UCI University of California, Irvine

CERN

Metrics, Measurements and Tools

Serhan Mete (UC Irvine) and Graeme Stewart (CERN)

Joint HSF/OSG/WLCG Workshop, VA, USA,
18-22 March 2019

Motivation

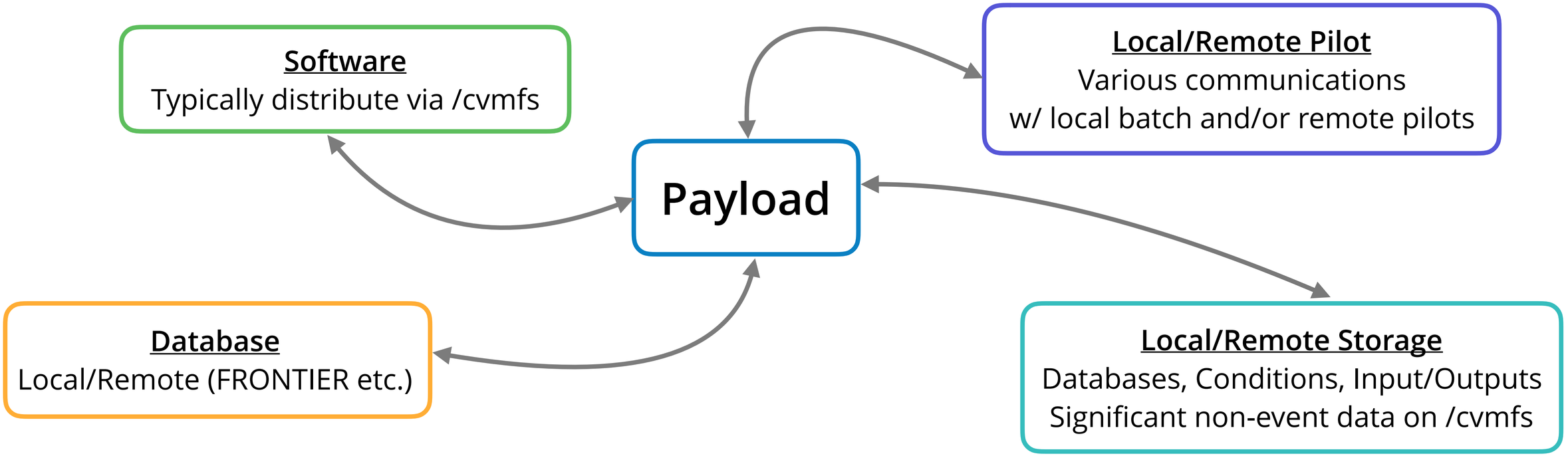
Systems Performance and Cost Modeling Working Group

Our community needs **metrics** that allow us **to characterise the resource usages of HEP workloads** in sufficient detail so that **the impact of changes in the infrastructure or the workload implementations can be quantified with a precision high enough to guide design decisions towards improved efficiencies**. This model has to express the resource utilisation of the workloads in terms of fundamental capabilities that computing systems provide, such as storage, memory, network, computational operations, latency, bandwidths etc.

- The WG mandate includes:
  - Bringing together workload and infrastructure experts (across sites/experiments)
  - *Identifying and agreeing on a set of reference workloads and benchmarking them*
  - *Building models and verifying them by predicting resource usage of the reference workloads*
  - Developing a strategy for mapping the model predictions to local costs and verifying this

WLCG Payloads in a Nutshell

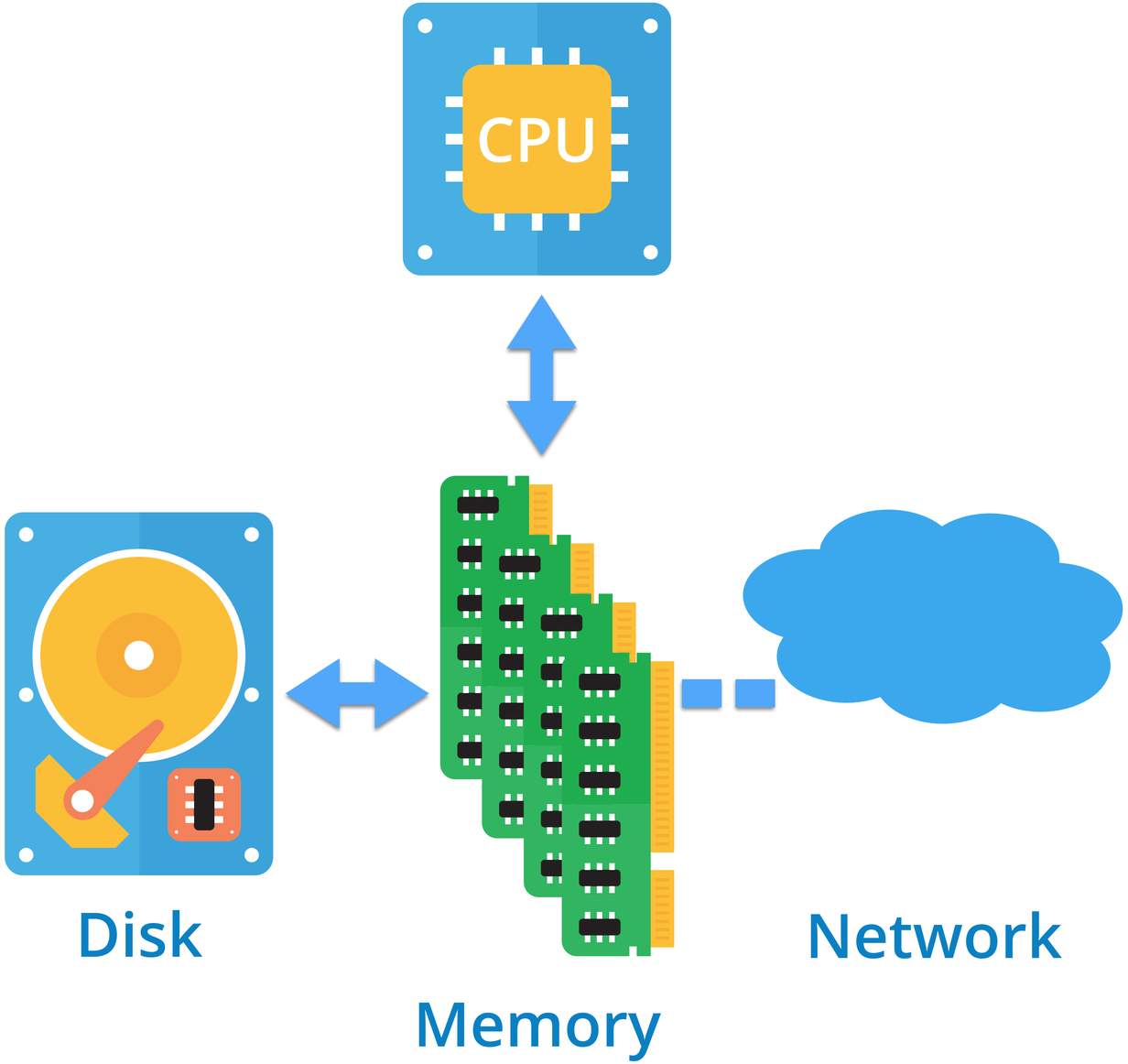
- In order to understand what metrics we need, we need to understand what we do



- Fairly non-local & complex due to the distributed nature of our computing model
- See hep-benchmarks/hep-workloads for a selection of LHC experiment workloads

What Goes within the Payload?

- In the most basic form, a concise measurement needs to focus at least on:
  - CPU (i.e. how much CPU time do I use?)
  - Memory (i.e. do I page a lot, swap at all?)
  - Disk I/O (i.e. how much do I read/write?)
  - Network I/O (i.e. what are my access patterns?)
- Most crucial metrics can be collected from ProcFS
  - Linux implementation includes a directory for each running process, including kernel processes, in directories named */proc/<pid>*, where *<pid>* is the process number
- For a more qualitative analysis one needs to dive a bit more:
  - Hardware counters etc.



Fork me on GitHub

Process Monitor (prmon)

Process Monitor (prmon)

build passing | license Apache-2.0

DOI 10.5281/zenodo.2554202

The PRocess MONitor is a small stand alone program that can monitor the resource consumption of a process and its children. This is useful in the context of the WLCG/HSF working group to evaluate the costs and performance of HEP workflows in WLCG. In a previous incarnation (MemoryMonitor) it has been used by ATLAS for sometime to gather data on resource consumption by production jobs. One of its most useful features is to use smaps to correctly calculate the *Proportional Set Size* in the group of processes monitored, which is a much better indication of the true memory consumption of a group of processes where children share many pages.

prmon currently runs on Linux machines as it requires access to the `/proc` interface to process statistics.

Metrics	Source	Comments
Process #	pstree /proc/<pid>/task/<pid>/children	Multi-process applications
Thread #	/proc/<pid>/stat	Multi-thread applications
CPU	/proc/<pid>/stat	User and System usage
Memory	/proc/<pid>/smaps	VMEM, RSS, PSS
IO	/proc/<pid>/io	Total bytes read/written
Network	/sys/class/net/<device>/statistics/*	<device>-level information

Running

The `prmon` binary is invoked with the following arguments:

```
prmon [--pid PPP] [--filename prmon.txt] [--json-summary prmon.json] \
      [--interval 30] [--netdev DEV] \
      [-- prog arg arg ...]
```

- `--pid` the 'mother' PID to monitor (all children in the same process tree are monitored as well)
- `--filename` output file for timestamped monitored values
- `--json-summmary` output file for summary data written in JSON format
- `--interval` time, in seconds, between monitoring snapshots
- `--netdev` restricts network statistics to one (or more) network devices
- `--` after this argument the following arguments are treated as a program to invoke and remaining arguments are passed to it; `prmon` will then monitor this process instead of being given a PID via `--pid`

Outputs

In the `filename` output file, plain text with statistics written every `interval` seconds are written. The first line gives the column names.

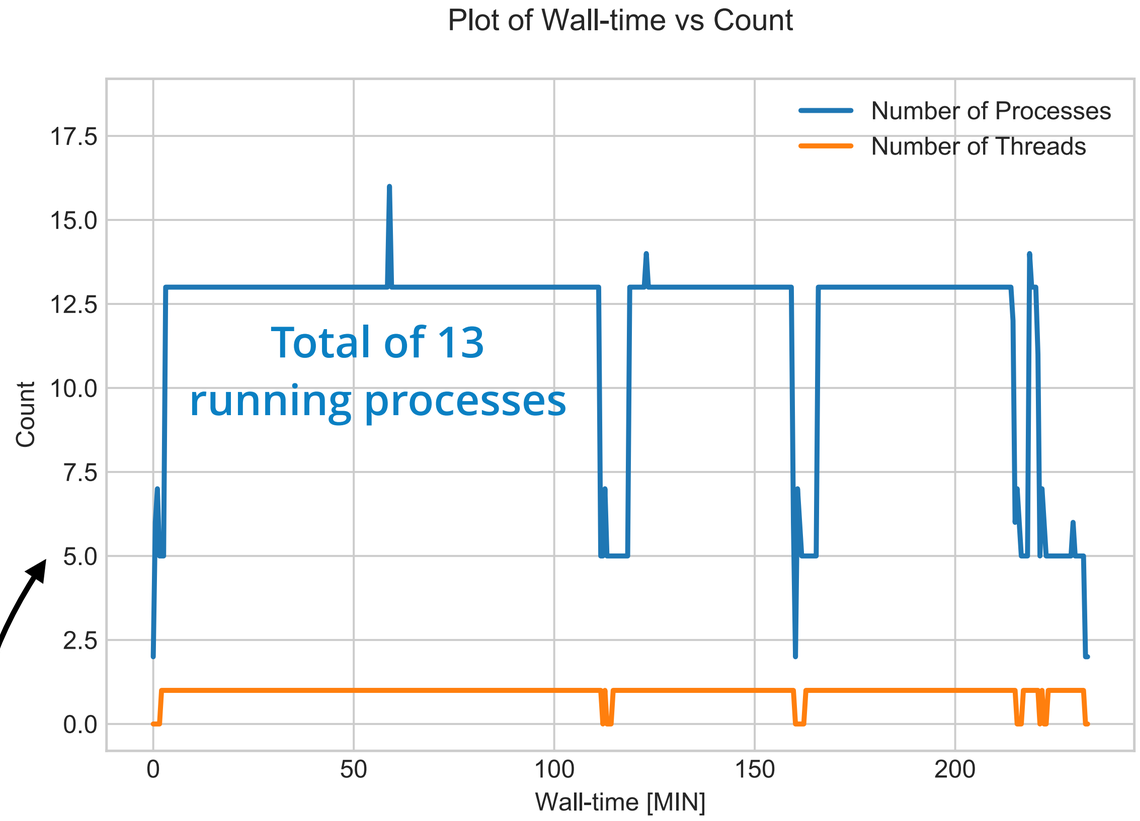
In the `json-summmary` file values for the maximum and average statistics are given in JSON format. This file is rewritten every `interval` seconds with the current summary values.

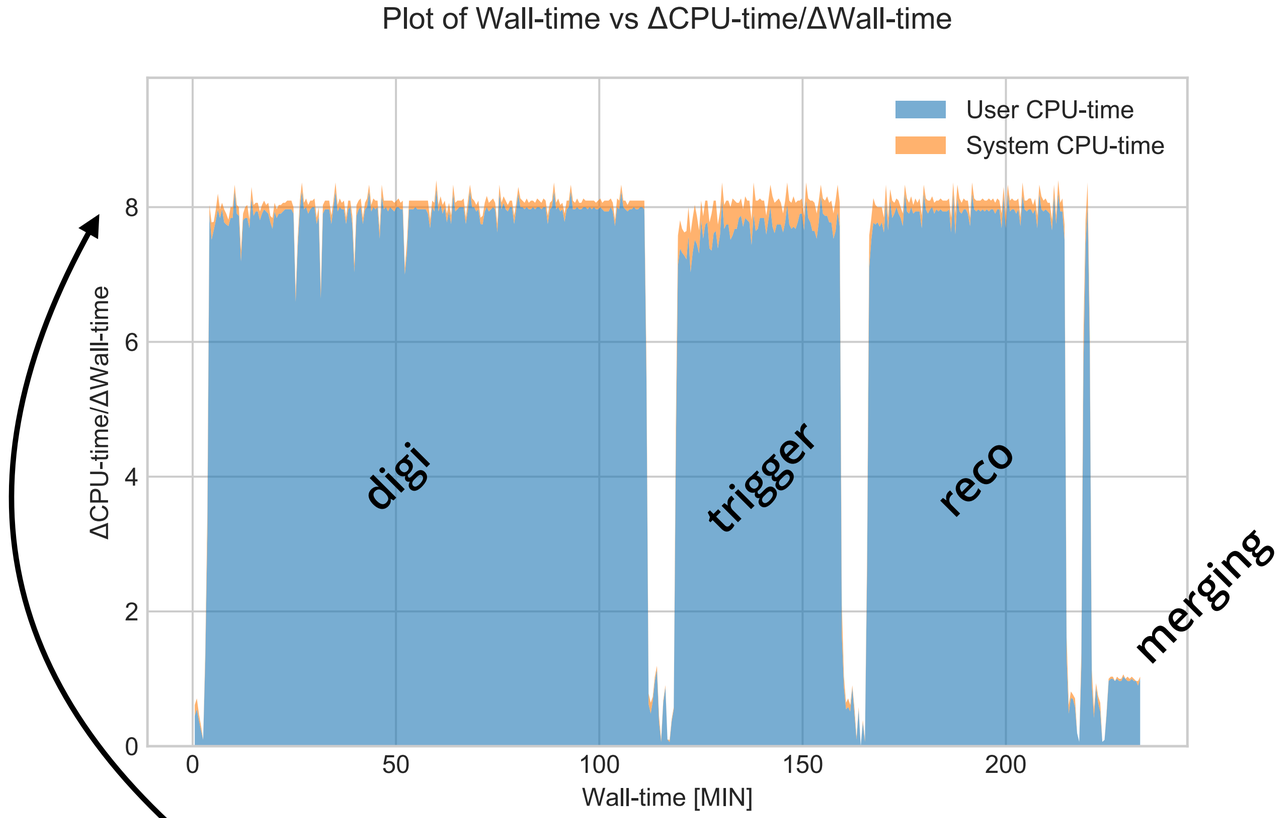
Monitoring of CPU, I/O and memory is reliably accurate, at least to within the sampling time. Monitoring of network I/O is **not reliable** unless the monitored process is isolated from other processes performing network I/O (it gives an upper bound on the network activity, but the monitoring is per network device as Linux does not give per-process network data by default).

Also include a script (*prmon_plot.py*) to visualize the results

ATLAS Workflow Example

- Running digitization, trigger simulation, reconstruction+ in 8 parallel processes





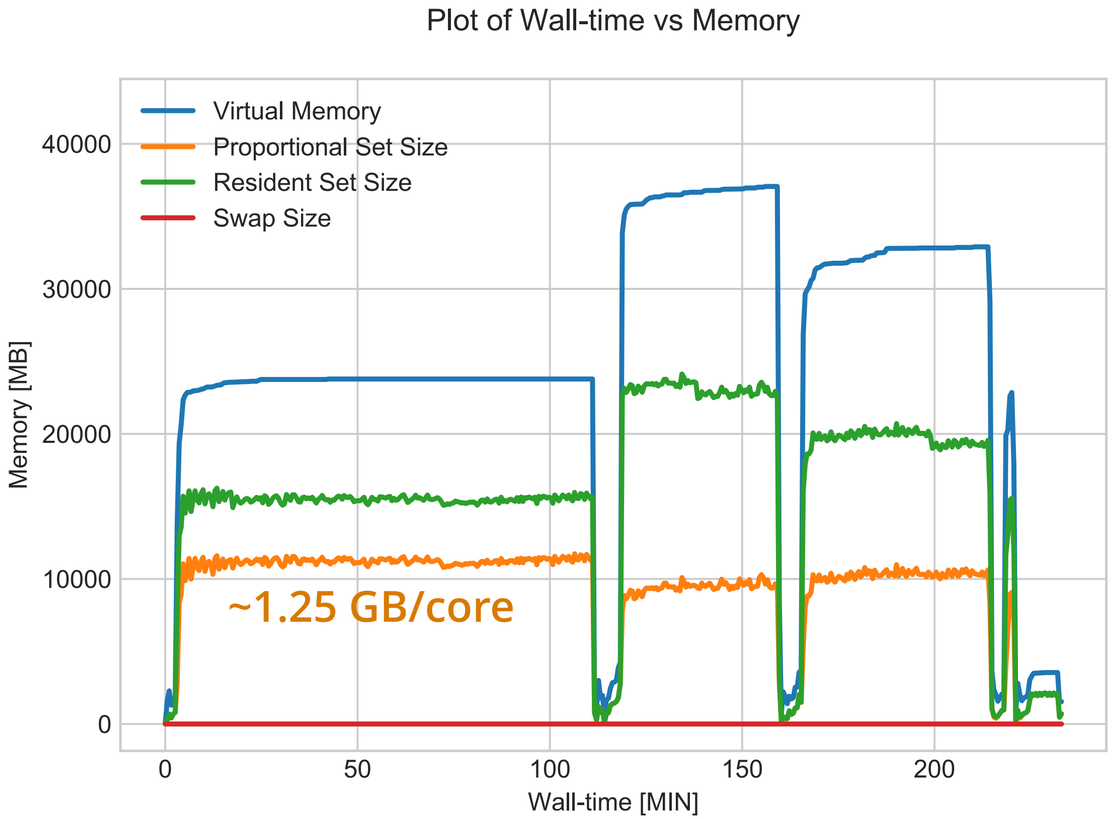
Total # of processes/threads

“offset”: 5 additional non-worker processes

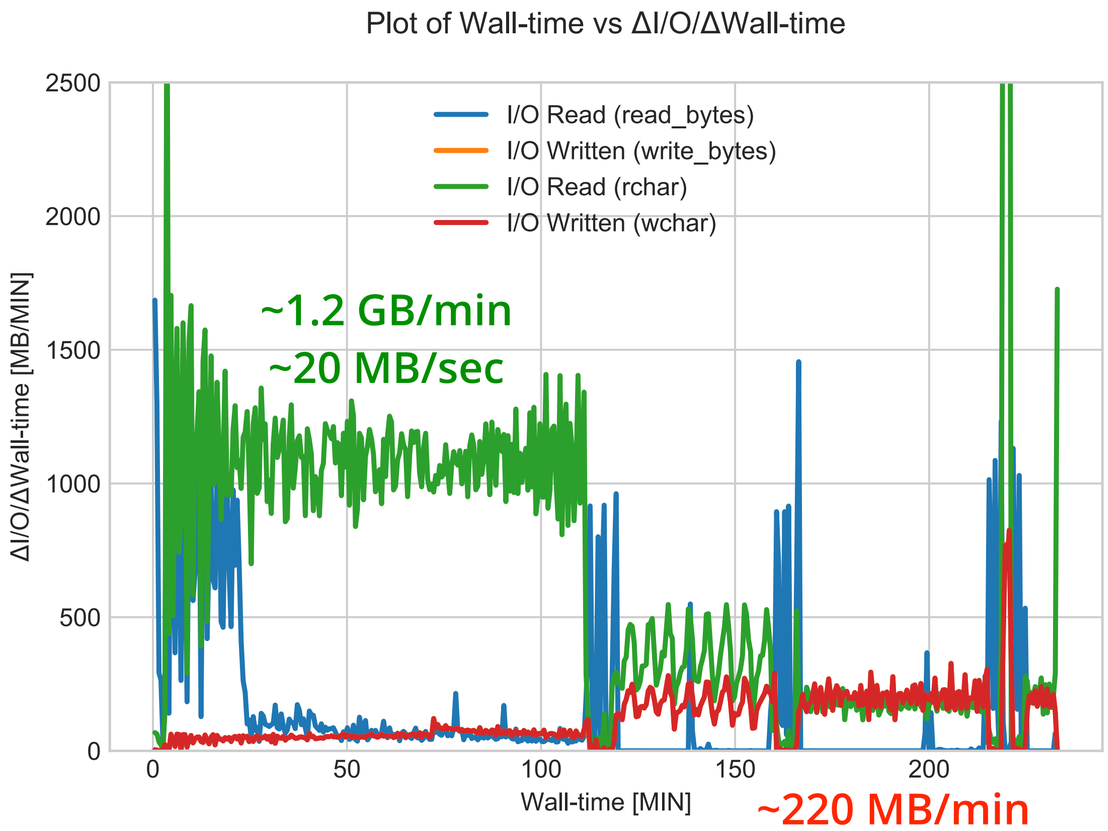
Effective CPU utilization

ATLAS Workflow Example (cont'd)

- Running digitization, trigger simulation, reconstruction+ in 8 parallel processes



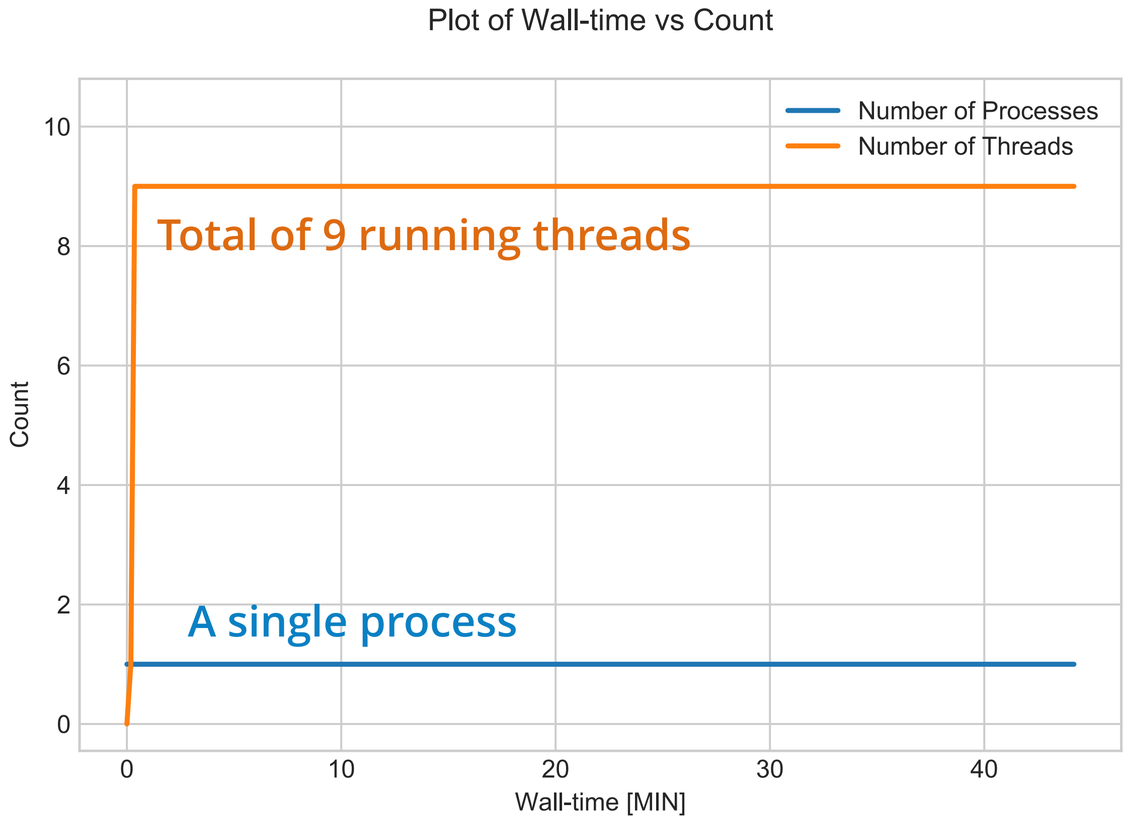
Total memory usage



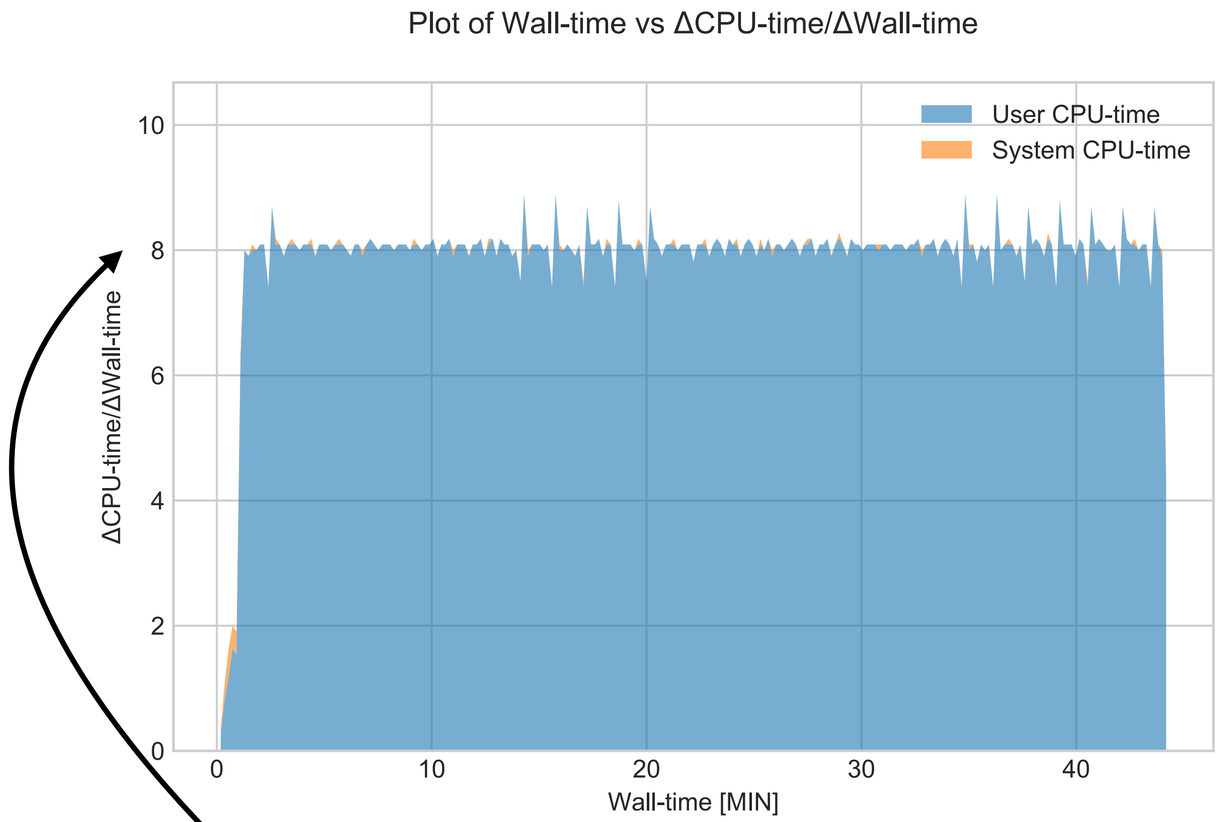
Disk I/O rate

CMS Workflow Example

- Running generation, simulation in 8 worker threads



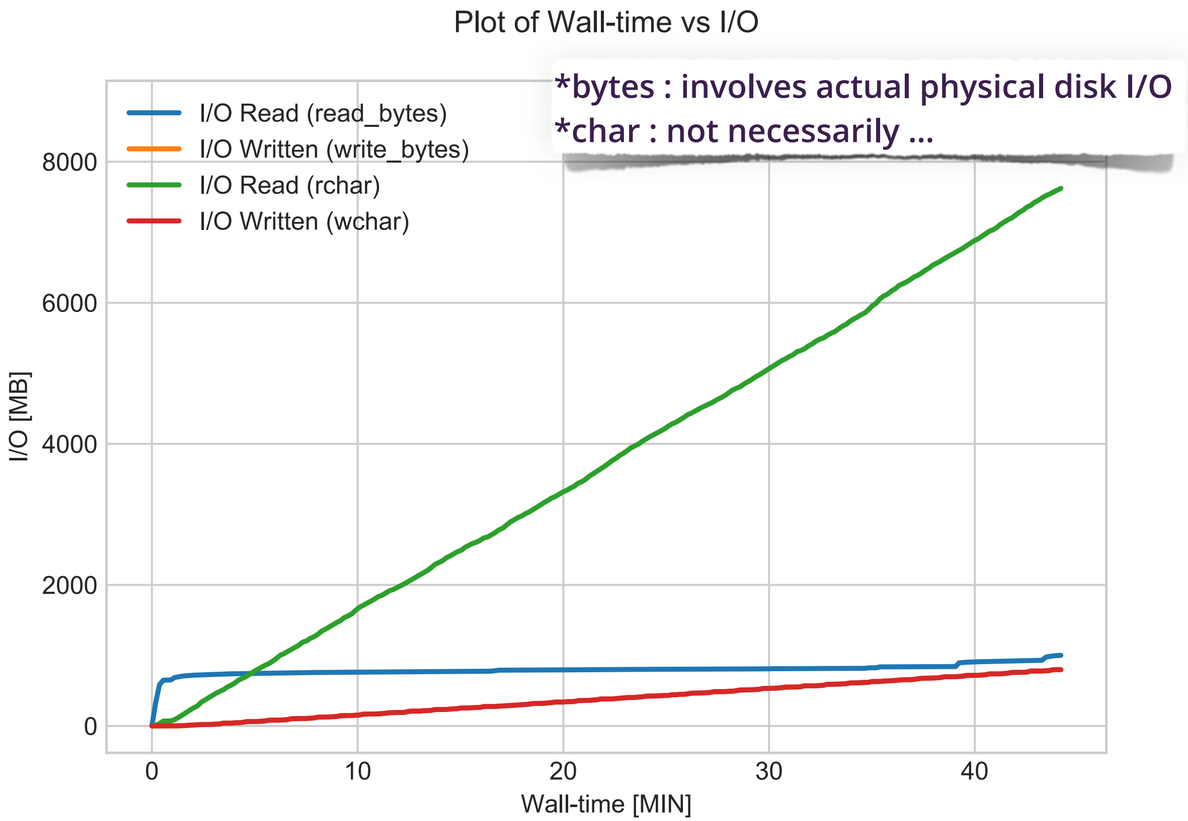
Total # of processes/threads



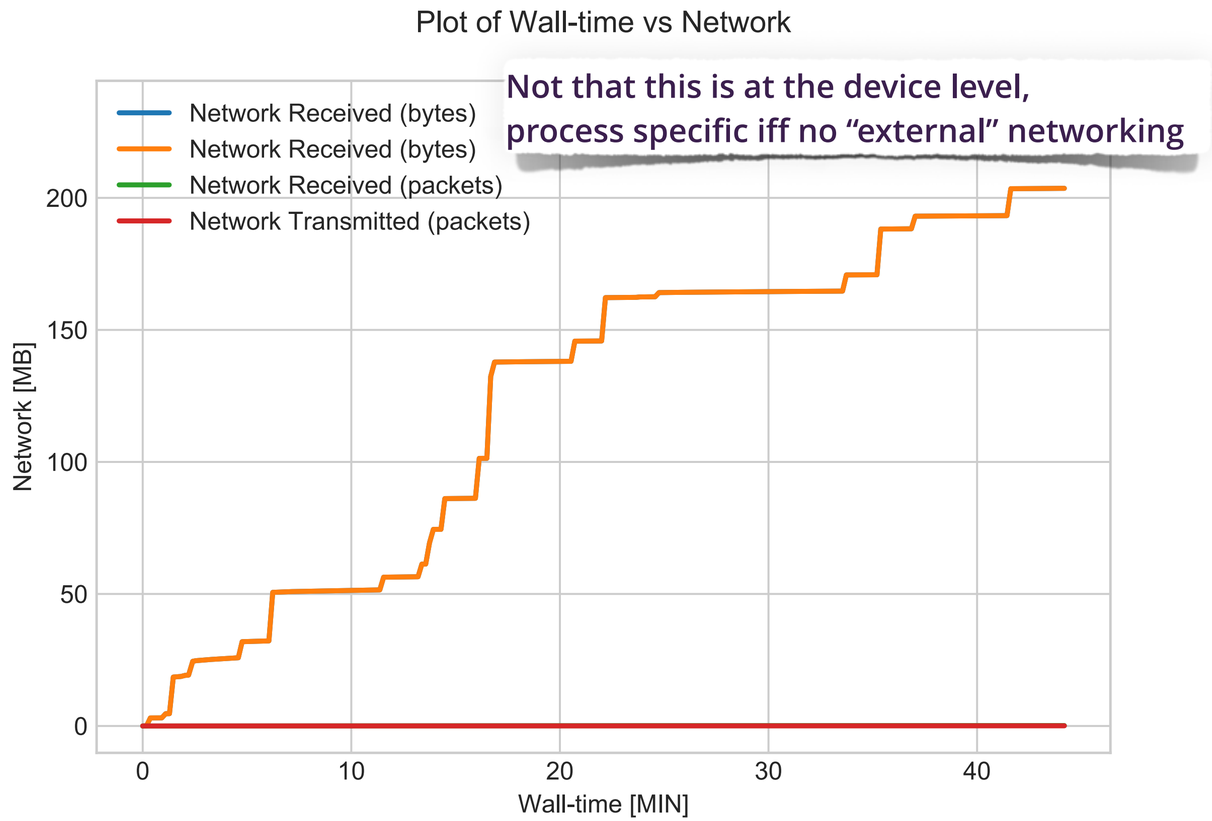
Effective CPU utilization

CMS Workflow Example (cont'd)

- Running generation, simulation in 8 worker threads



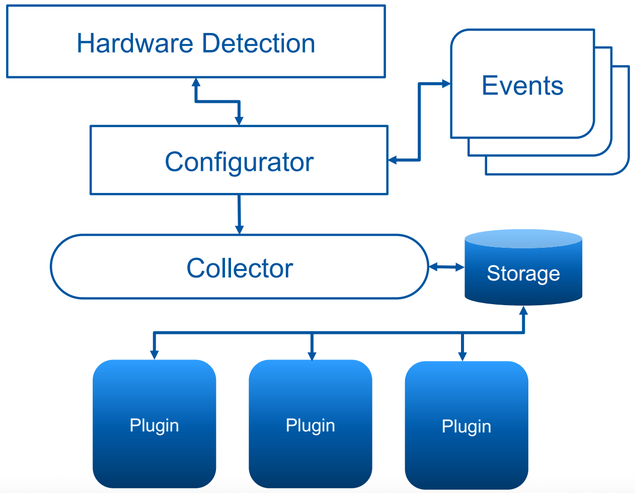
Disk I/O read/write



Network received/transmitted

Trident

Trident is a qualitative analysis tool that can look at various low level metrics with respect to the Core, Memory and I/O to highlight performance bottlenecks during the execution of an application.



Configurator detects the hardware and looks for event configuration

Collectors records hardware counters for the corresponding metrics

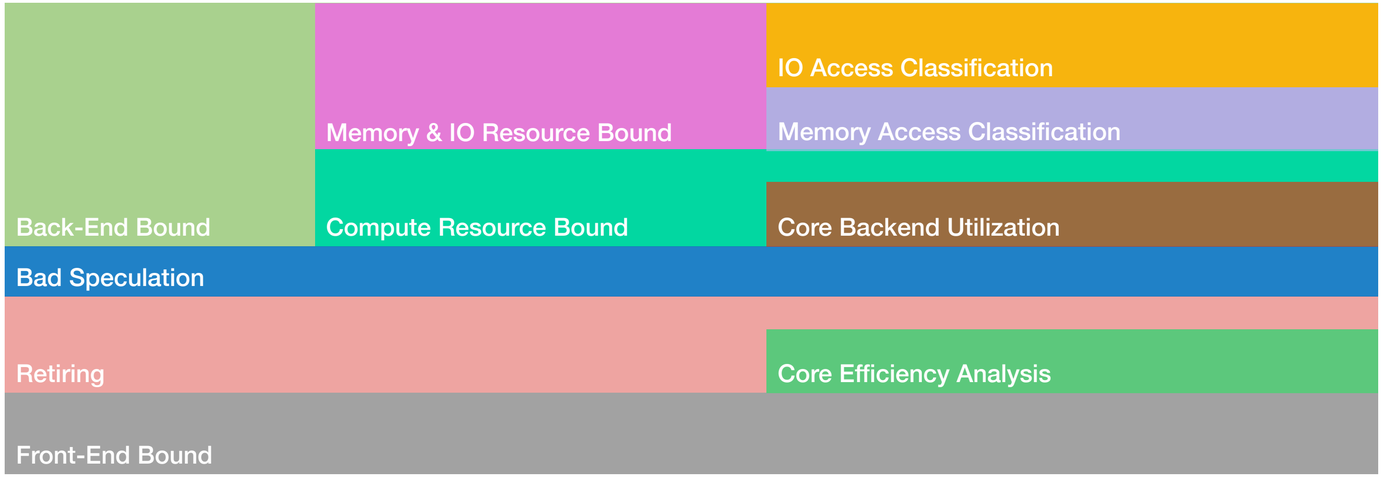
- One based on **perf** to collect memory and core information
- The other is a wrapper that captures IO counters from ***ProcFS***

Core results are presented as histograms

- Instruction Per Cycle (IPC)
- Execution Unit Port Utilization

Top-Down Analysis (IO/memory classification)

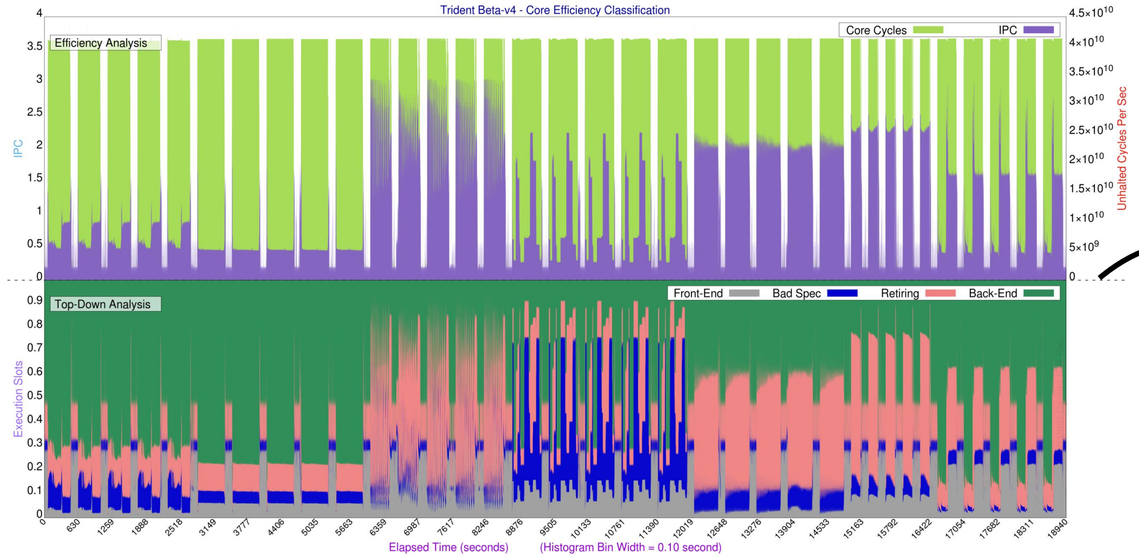
- Bandwidths, transfer rates etc.



Extended Top Down Analysis

Trident in Action

HS06



Examples based on HEPSPEC06 Benchmark Suite

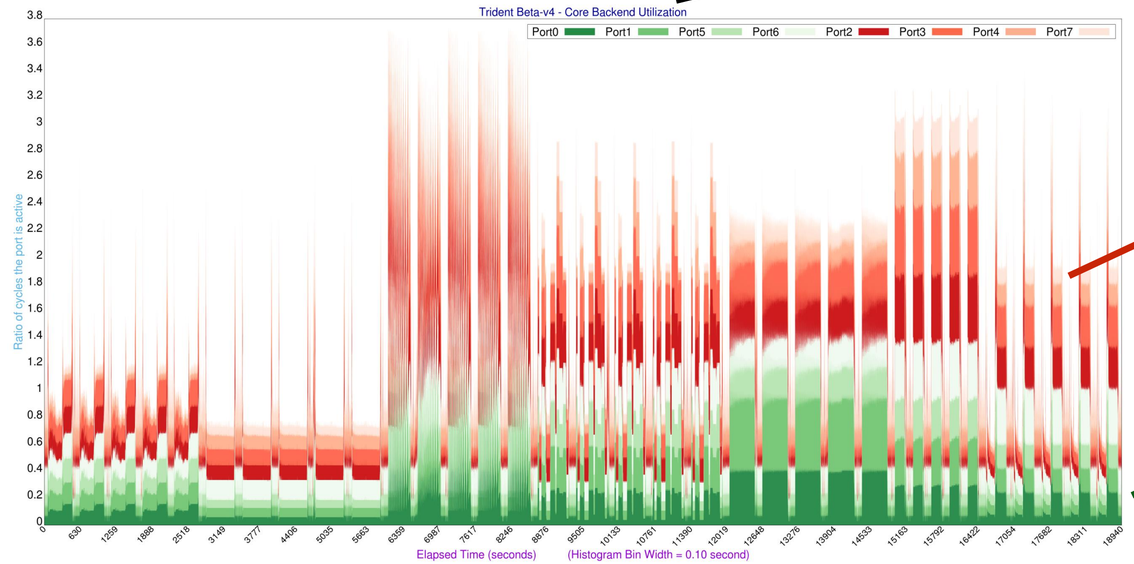
- 450.soplex, 471.omnetpp, 447.dealII, 473.astar, 444.namd, 453.povray, 483.xalancbmk

Core Efficiency Classification measures:

- Core cycles and IPC
- Break-down of execution slots:
  - Front/Back-End, Bad Speculation, Retiring

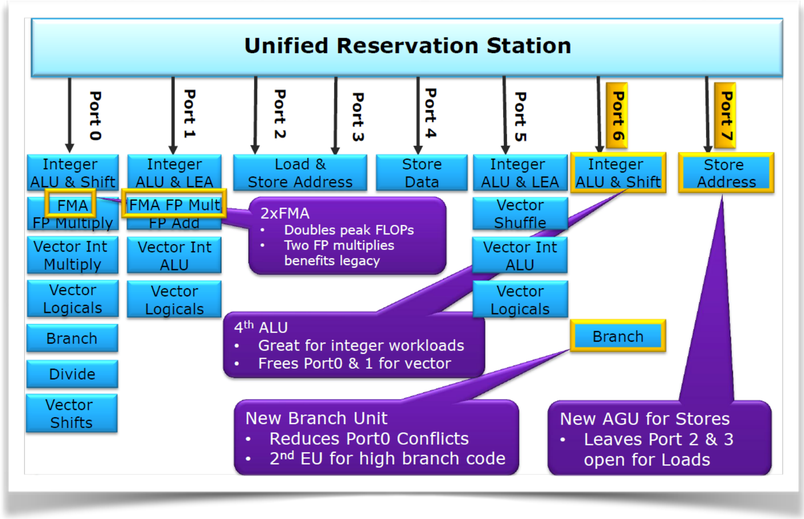
Core Backend Utilization is broken down to ports

HS06



Memory

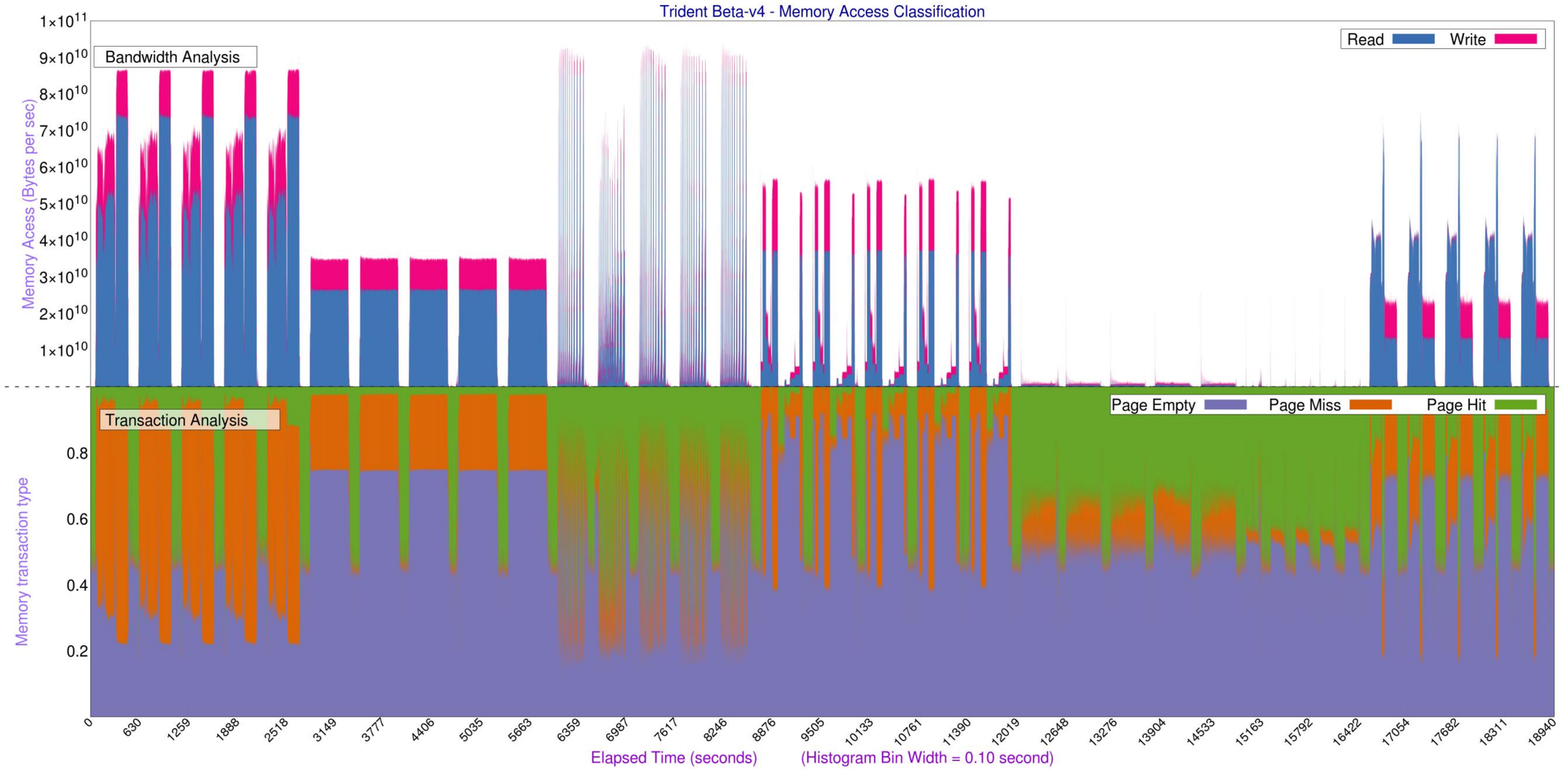
Arithmetic



Intel Haswell EP

Trident in Action (cont'd)

HS06



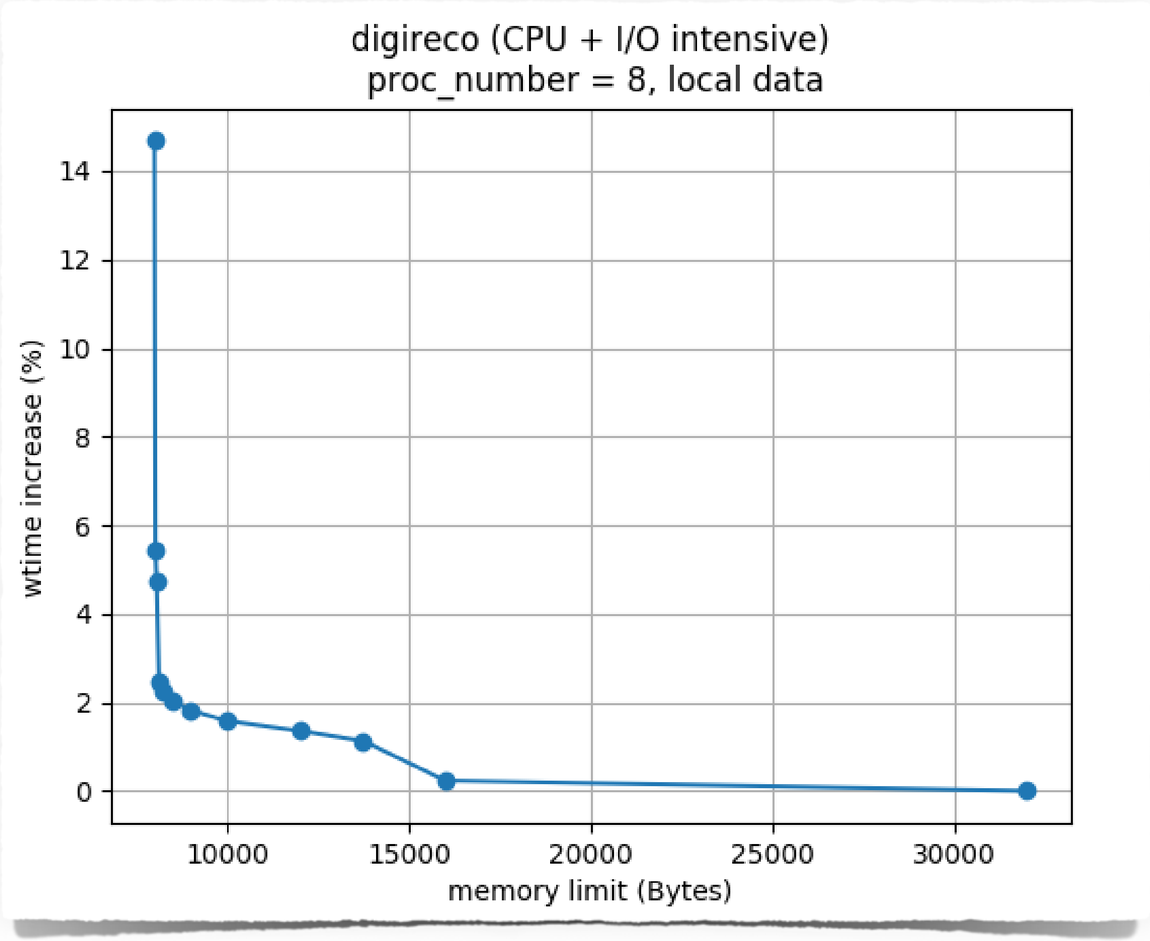
Conclusions

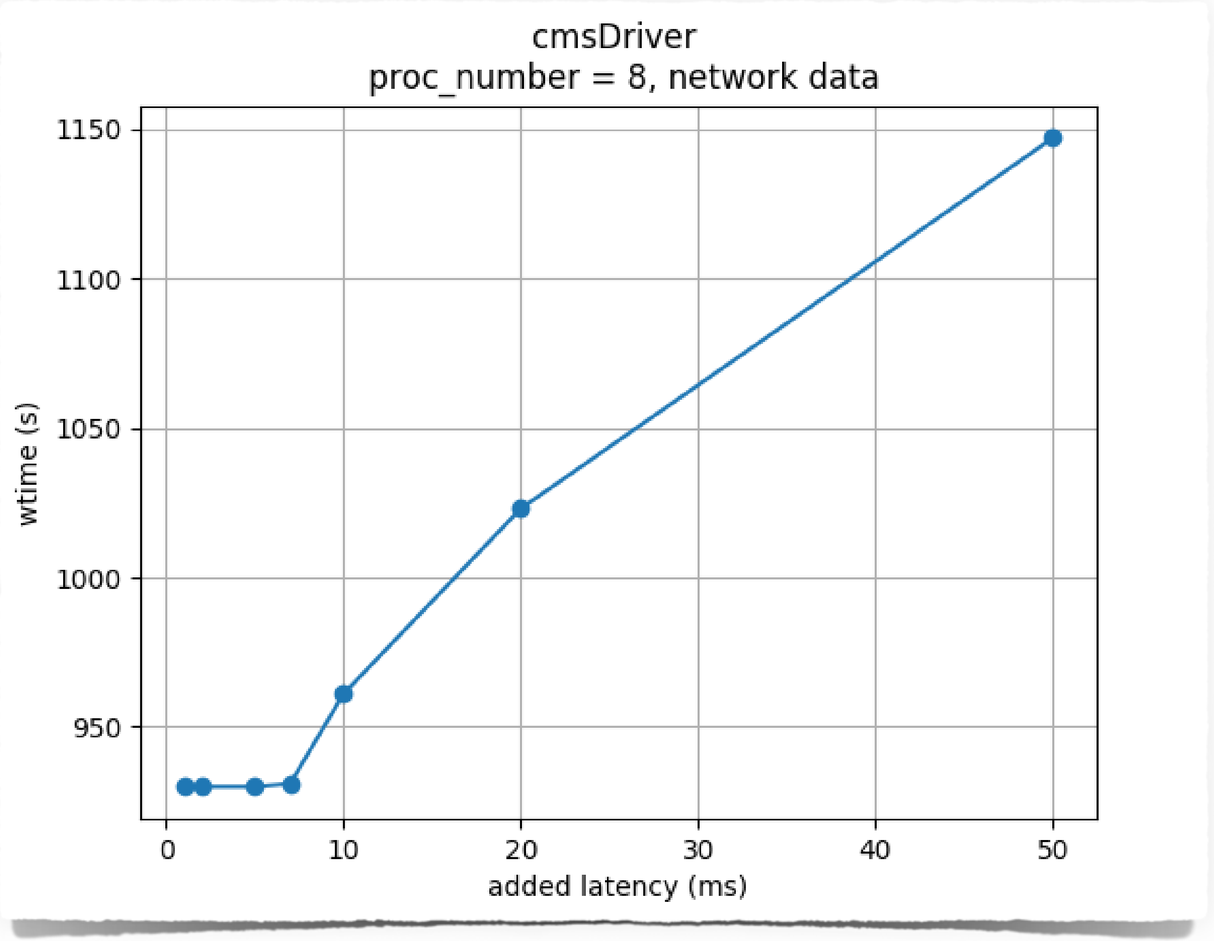
- Detailing and measuring key performance metrics is essential to understand any workflow/workload
- Tools designed by HEP people for HEP people are already available, some are presented here
- It’s important to distinguish performance monitors from profilers
  - Former needs to be common, lightweight and bring minimal (no) overhead
    - prmon is designed and implemented to do just this w/o administrative privileges
  - Trident can provide further insight to the inner workings of an application
    - By accessing hardware counters and presenting digestible information on possible bottlenecks
    - More of a software developer oriented tool
- It’s very important to:
  - Constantly monitor the workloads to avoid performance degrading changes
  - Understand and assess the resource needs to guide decision making
  - Correlate the information between monitors, profilers and the software to help improve the latter

BACKUP

Workloads under Network/Memory Limitations

Emulate network/memory limitation (via cgroups) to asses (prmon) the impact on ATLAS/CMS payloads





See this talk for more information

Prmon Workflow Examples

ATLAS

```
export ATHENA_PROC_NUMBER=8
prmon --interval 30 -- /usr/bin/time Reco_tf.py –inputHITSFile=… --maxEvents="2000" --postInclude "default:PyJobTransforms/
UseFrontier.py" --preExec "all:rec.Commissioning.set_Value_and_Lock(True);from AthenaCommon.BeamFlags import
jobproperties;jobproperties.Beam.numberOfCollisions.set_Value_and_Lock(20.0);[…]" --inputHighPtMinbiasHitsFile="HITS.
10701335._002545.pool.root.1,HITS.10701335._002546.pool.root.1,HITS.10701335._002547.pool.root.1,HITS.10701335._002548.pool.root.1" --
inputLowPtMinbiasHitsFile="HITS.10701323._008539.pool.root.1,HITS.10701323._008545.pool.root.1,HITS.10701323._008550.pool.root.1,HITS.
10701323._008552.pool.root.1,HITS.10701323._008556.pool.root.1,HITS.10701323._008557.pool.root.1" […]
```

CMS

```
THREADS=8
EVENTS=1000
prmon -i 10 -- cmsDriver.py TTbar_13TeV_TuneCUETP8M1_cfi --conditions auto:phase1_2018_realistic -n $EVENTS --era Run2_2018 --
eventcontent RAWSIM --relval 9000,50 -s GEN,SIM --datatier GEN-SIM --beamspot Realistic50ns13TeVCollision --fileout step1.root --
nThreads $THREADS > stdout 2> stderr
```