Metrics, Measurements and Tools

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

  - **Software**
    Typically distribute via /cvmfs

  - **Database**
    Local/Remote (FRONTIER etc.)

  - **Local/Remote Pilot**
    Various communications w/ local batch and/or remote pilots

  - **Local/Remote Storage**
    Databases, Conditions, Input/Outputs
    Significant non-event data on /cvmfs

• 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.
Process Monitor (prmon)

PrMon is a small standalone 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 HEPA workflows in WLCG. In a previous incarnation (MemoryMonitor) it has been used by ATLAS for some time 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.

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

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

Total of 13 running processes

Plot of Wall-time vs Count

Plot of Wall-time vs ΔCPU-time/ΔWall-time

User CPU-time

System CPU-time

digi

trigger

reco

merging
ATLAS Workflow Example (cont’d)

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

**Total memory usage**

- ~1.25 GB/core

**Disk I/O rate**

- ~1.2 GB/min
- ~20 MB/sec
- ~220 MB/min
- ~3.5 MB/sec
CMS Workflow Example

- Running generation, simulation in 8 worker threads

Total # of processes/threads

Effective CPU utilization

Plot of Wall-time vs Count

Plot of Wall-time vs ΔCPU-time/ΔWall-time

Total of 9 running threads

A single process
CMS Workflow Example (cont’d)

- Running generation, simulation in 8 worker threads

**Plot of Wall-time vs I/O**

- I/O Read (read_bytes)
- I/O Written (write_bytes)
- I/O Read (rchar)
- I/O Written (wchar)

*bytes: involves actual physical disk I/O
*char: not necessarily...

**Plot of Wall-time vs Network**

- Network Received (bytes)
- Network Received (bytes)
- Network Received (packets)
- Network Transmitted (packets)

Not that this is at the device level, process specific iff no “external” networking

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.
Trident in Action

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

Memory

Arithmetic

Unified Reservation Station

- Integer ALU & Shifter
- 2x FMA
- Vector Int. Multiply
- Vector Int. Add
- Vector Logicals
- Branch
- Divide
- Vector Shifts

- Load & Store Address
- Load & Store Data
- Store & Load
- Integer ALU & LEA
- New AGU for Stores
- New Branch Unit
- New Branch
- 2nd EU for high branch code
- Improves Port B & 1 for vector
- 4th AU
- Great for integer workloads
- Improves Port C
- Leaves Port 2 & 3 open for loads
- Doubles peak IPCs
- Two PIM processors
Trident in Action (cont’d)
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 assess (prmon) the impact on ATLAS/CMS payloads

Sensitivity to swap usage

Corentin Bugnot (University of Geneva)

Sensitivity to latency

Corentin Bugnot (University of Geneva)
Prmon Workflow Examples

**ATLAS**

```bash
export ATENA_PROC_NUMBER=8
```

**CMS**

```bash
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
```