Monitoring of Computing Resource Use of Active Software Releases in ATLAS

Antonio Limosani
On behalf of the ATLAS collaboration
• ATLAS is a diverse and globally distributed software project

• Millions of lines of codes distributed across more than 2500 packages built into shared libraries. Almost one package per every ATLAS author

• Diverse workflows to satisfy the goals of research

• Workflows deployed on heterogeneous computing systems
  • Tier0/1/2/3, WLCG, Cloud, HPC, laptops
  • Perform these tasks optimally (cheaply) as possible
  • Achieve maximum throughput of useful “physics events”
  • Precision physics requires billions of events
Resource monitoring

“If you can’t measure it, you can’t improve it”
Peter Drucker

CPU, Memory and Disk Storage are measured

• To track usage with respect to pre-defined directives and hardware limits
• Prevent bottlenecks entering production workflows
• Reduce incidence of job failures
• Give feedback to developers to remove bottlenecks and optimise their code
• Promote practice and culture of monitoring and optimisation

ATLAS Preliminary

ATLAS Offline Committers (by quarter)
PerfMon

- PerfMon is a toolkit developed in the context of the Athena framework
- Captures snapshots of the application's state for later analysis
- Provides detailed information of every algorithm executed
  - CPU and wall clock time,
  - Complete memory footprint of every algorithm executed and library loaded
- Uses hooks provided by the Gaudi framework to monitor the different stages of a job
  - configuration
  - initialisation
  - execute
  - finalise
- Retrieves data from Auditors e.g. ChronoStatAuditor, MemAuditor
• PerfMonSD = Semi-Detailed mode

Monitor stages of a job

• “ini” Initialisation

• “1st” the first event

• “cbk” callbacks

• “evt” event loop

• “fin” finalisation

• “dso” cost of loading libraries

• “preLoadProxy” cost of conditions data

Summary information

• Tests with PerfMonSD enabled are run on development software releases daily

• Little overhead ~ typically adding 1% to overall resources

```
PMonSD ================step ini ==================
PMonSD          n    cpu    vmem  malloc component
PMonSD          1  6390   24704   23623 GeoModelSvc
PMonSD          1  160  10240  10373 ToolSvc.MdtCalibDbTool

PMonSD ================step 1st ==================
PMonSD          n    cpu    vmem  malloc component
PMonSD          1  40  16384  17610 MooSegmentMaker
PMonSD          1  70  1024   779 MboyRec

PMonSD ================ step cbk ==================
PMonSD          n    cpu    vmem  malloc component
PMonSD          1  1880  56204  64841 TrackingGeometrySvc[0x14661258]+21
PMonSD          1  3280  22168  20806 MboySvc[0x12587000]+e732ae70

PMonSD =================step evt ====================
PMonSD          n    cpu    max@evt    vmem    max@evt  malloc    max@evt  component
PMonSD          249  153  3850@182  0      0@0       175  2009@182  MboyRec
PMonSD          249  102  110@1   0      0@0       145  152@182  MuonRpcRawDataProvider

PMonSD ================= step fin =====================
PMonSD          n    cpu    vmem  malloc component
PMonSD          1  60  0    -49214 DetectorStore
PMonSD          1  60  0    -36871 ToolSvc.MuonTrackingGeometryBuilder

PMonSD ================ step dso ======================
PMonSD          n    cpu    vmem  malloc component
PMonSD          1  120  33100  6 liblcg_OracleAccess.so
PMonSD          1  20  32236  5 libBFieldAth.so

PMonSD ================ step preLoadProxy ===============
PMonSD          n    cpu    vmem  malloc component
PMonSD          1  230  15692  11754 CondAttrListCollection[/MDT/T0]
PMonSD          1  210  8992  5573 CondAttrListCollection[/RPC/CABLING/MAP_SCHEMA]

PMonSD ================ special info ===============
PMonSD          n    cpu    wall    vmem  malloc component
PMonSD          1  250  379  379   -     - evtloop_time
PMonSD          1  200   -  127  103 leakperevt_evt51plus
PMonSD          1  676648 660290 501841
```
Monitoring webpages

CPU time

- http://atlas-pmb.web.cern.ch/atlas-pmb/

Memory

• Other 2129
• RecoSvc:egammaMVATool
• RecoSvc:AligAlg
• RecoSvc:TBRecalch/tcalib
• EDprintCentralAlg
• RecoSvc:MuonStaTool
• UA7_11RecalkClusterMaker:CaloRunClusterCorrectionsAccalev7_11
• RecoSvc:MCAligTool
• EventSelector
• TauDiscriminant
• InDetSVT:ConditionsSummarySvc
• TauDiscriminant:taueleBDT
• RooSvc
• RecoSvc:LooseLHSelector
• AG010toGeoSvc
Software Monitoring evolution / plan

- **AFS cron**
- **AFS project area**
- **Static Webpage**
- **JIRA issues**
- **Savannah**

**Workflow**
- **.job scheduler**
  - **Jenkins/Gitlab**
- **Database**
  - **SQLite**
- **Generate Reports**
  - **Interactive plots / Dashboards**

**Output**
- **PerfMonSD**
- **Pool/ROOT files**
- **logs**
Monitoring of workflows in daily builds

- Generation (of physics events) EVNT
- Simulation (Produce GEANT4 hits) EVNT to HITS
- Digitisation (Digitise hits as detector readout) HITS to RDO (Raw Data Objects)
- Reconstruction (Reconstruct physics objects components) RDO to AOD (Analysis Object Data)
- Analysis Derivations (Slim, Skim, and Thin events for dedicated physics analysis streams) AOD to DxAOD
- Final Analysis (Calibrations and plot making leading to physics measurements)

Wall clock time fraction for grid and HPC jobs
July 2015 - July 2016

<table>
<thead>
<tr>
<th>Step</th>
<th>Time per event (sec)</th>
<th>PSS/RSS (GB)</th>
<th>Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVNT to HITS</td>
<td>180</td>
<td>1.7</td>
<td>19.2</td>
</tr>
<tr>
<td>HITS to RDO (4-core)</td>
<td>32</td>
<td>6.1/8.4</td>
<td>20.7</td>
</tr>
<tr>
<td>RDO to RDOMonitor (4-core)</td>
<td>10</td>
<td>6.7/12.8</td>
<td>20.7</td>
</tr>
<tr>
<td>RAW to ESD (4-core)</td>
<td>13</td>
<td>7.2/11.9</td>
<td>20.7</td>
</tr>
<tr>
<td>ESD to AOD (4-core)</td>
<td>0.3</td>
<td>5.1/7.9</td>
<td>20.7</td>
</tr>
<tr>
<td>AOD to DxAOD MC15ExotJetM</td>
<td>0.4</td>
<td>2.4</td>
<td>20.7</td>
</tr>
<tr>
<td>AOD to DxAOD Data15ExotJetM</td>
<td>0.15</td>
<td>1.7</td>
<td>20.7</td>
</tr>
<tr>
<td>Physics Main $\langle \mu \rangle \approx$ 30 (4-core)</td>
<td>11</td>
<td>11.0/18.3</td>
<td>20.7</td>
</tr>
<tr>
<td>BS to ESD (4-core)</td>
<td></td>
<td>7.7/11.7</td>
<td>20.7</td>
</tr>
<tr>
<td>ESD to AOD</td>
<td>0.9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Monitoring of production workflow @ Tier0

• New program: **RunTier0Tests.py** where developer updates are tested for changes in file outputs, memory, CPU time and log file messaging in both simulated and real data workflows.

• Configured to run simultaneous tests on the same machine with and without the proposed code changes

• Around an hour to complete on typical CERN/lxplus interactive nodes

• Options for speedup, include running all tests simultaneously but need multi-core machine with free slots

• Special options include comparisons against production releases

• Has had the effect of preventing disastrous memory leaks and new bottlenecks entering into the code base in imminent production releases
Multicore memory monitoring

• To save memory we use “Athena” framework in multi-processor mode “AthenaMP”

• Absolutely essential to take into account memory shared across processors

• Use MemoryMonitoringTool (N. Rauschmayr, R. Seuster) and which records evolution of VMEM, Resident Set Size, Proportional Set Size and Swap memory usage for a process and all its child processes. Probes /proc/<process_ID>/smaps files.

• Needed to monitor and track improvements in memory usage

• Here comparing memory using 1 multicore job using 8 cores versus 8 serial jobs using 1 core each

• Advanced and comprehensive studies of memory usage and patterns. See

Rauschmayr & Kama “Identifying memory allocation patterns in HEP software” @ CHEP 2016
Rauschmayr & Kama Find Obselete Memory
• Current monitoring and benchmarking framework

• has allowed developers to optimise code performance

• anticipate resource usage based on current workflows

• plan resource requests and define limits for future workflows

• react to and measure impact of external changes i.e. ROOT5 to ROOT6, CMT to CMake, CLHEP versions, migration from CLHEP to Eigen
Compiler Optimisation studies

- gcc compiler has multiple optimisation levels
- Benchmark and PerfMon allow for systematic studies across multiple workflows
- gcc4.9.3
- Average CPU time per event in seconds

<table>
<thead>
<tr>
<th>Workflow</th>
<th>-O2</th>
<th>-Os</th>
<th>-O3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation</td>
<td>156 (1)</td>
<td>171 (1.10)</td>
<td>140 (0.90)</td>
</tr>
<tr>
<td>Digitisation</td>
<td>22 (1)</td>
<td>25 (1.14)</td>
<td>20 (0.91)</td>
</tr>
<tr>
<td>Trigger</td>
<td>7.9 (1)</td>
<td>11.2 (1.42)</td>
<td>7.6 (0.96)</td>
</tr>
<tr>
<td>Reconstruction</td>
<td>10.0 (1)</td>
<td>11.4 (1.14)</td>
<td>9.4 (0.94)</td>
</tr>
</tbody>
</table>
Memory allocator optimisation studies

- Multiple memory allocators are available were investigated for use in the reconstruction workflow (tc: Google tcmalloc)

<table>
<thead>
<tr>
<th>Memory Allocator</th>
<th>CPU (milli-sec)</th>
<th>RSS_MEAN (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>stcmalloc</td>
<td>1'150</td>
<td>3.1</td>
</tr>
<tr>
<td>LCG tc</td>
<td>2'300</td>
<td>3.3</td>
</tr>
<tr>
<td>tc default</td>
<td>3'450</td>
<td>3.5</td>
</tr>
<tr>
<td>tc32</td>
<td>4'600</td>
<td>3.8</td>
</tr>
<tr>
<td>tc small</td>
<td></td>
<td></td>
</tr>
<tr>
<td>concur</td>
<td></td>
<td></td>
</tr>
<tr>
<td>umem</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jemalloc</td>
<td>3'450</td>
<td>3.8</td>
</tr>
<tr>
<td>hoard</td>
<td></td>
<td>3.6</td>
</tr>
</tbody>
</table>
Conclusions

• Other tools used: AthMemoryAuditor (R. Seuster) quick, lightweight memory leak checker; Valgrind (callgrind, memcheck, massif); Heap Profiles using jemalloc; FOMTool (N. Rauschmayr & S. Kama), gperftools, igprof.org

• Software monitoring is essential to optimisation of ATLAS workflows in a distributed multi-developer environment

• Existing framework is being upgraded to handle current and future workflows

• SQL will provide a powerful means of analysing performance data and will allow greater flexibility in tracking changes.

• Move to multi-threading algorithms will present considerable challenge

G. Stewart “How To Review 4 Million Lines of ATLAS Code” @ CHEP