Monitoring of Computing Resource Use of Active Software Releases at ATLAS

Antonio Limosani on behalf of the ATLAS Collaboration
CERN CH-1211 Geneva 23 Switzerland and
University of Sydney, NSW, Australia
E-mail: antonio.limosani@cern.ch

Abstract. The LHC is the world’s most powerful particle accelerator, colliding protons at centre of mass energy of 13 TeV. As the energy and frequency of collisions has grown in the search for new physics, so too has demand for computing resources needed for event reconstruction. We will report on the evolution of resource usage in terms of CPU and RAM in key ATLAS offline reconstruction workflows at the Tier0 at CERN and on the WLCG. Monitoring of workflows is achieved using the ATLAS PerfMon package, which is the standard ATLAS performance monitoring system running inside Athena jobs. Systematic daily monitoring has recently been expanded to include all workflows beginning at Monte Carlo generation through to end-user physics analysis, beyond that of event reconstruction. Moreover, the move to a multiprocessor mode in production jobs has facilitated the use of tools, such as "MemoryMonitor", to measure the memory shared across processors in jobs. Resource consumption is broken down into software domains and displayed in plots generated using Python visualization libraries and collected into pre-formatted auto-generated Web pages, which allow the ATLAS developer community to track the performance of their algorithms. This information is however preferentially filtered to domain leaders and developers through the use of JIRA and via reports given at ATLAS software meetings. Finally, we take a glimpse of the future by reporting on the expected CPU and RAM usage in benchmark workflows associated with the High Luminosity LHC and anticipate the ways performance monitoring will evolve to understand and benchmark future workflows.

1. Introduction
ATLAS software is a diverse and globally distributed project consisting of millions of lines of code distributed across more than 2500 packages. In order to meet the research goals of the ATLAS experiment the software stack handles the many data processing steps shown in figure 1. These steps are deployed on the heterogeneous computing systems that make up the Worldwide LHC Computing Grid (WLCG). To both measure known processes with precision and search for rare phenomena, trillions of physics events must be processed. The aim is to achieve a maximum throughput of analysis of physics events within WLCG budgets of CPU, memory and disk storage. To that end detailed monitoring down to the level of resource usage of algorithms is systematically conducted to:

• ensure workflows respect the nominal memory limit of 2 GB per core;
• prevent CPU time bottlenecks from entering production workflows;
• reduce the incidence of job failures due to excessive demands on resources;
• give feedback to developers and allow them to target and confirm performance gains; and
• overall promote a culture of testing, monitoring, and optimisation.

Figure 1. Schematic representation of the Full Chain Monte Carlo production and real data processing.

2. PerfMon
PerfMon [1] is a toolkit developed in the context of the Athena framework [2], relying on its clear finite state machine to capture snapshots of the application’s state for later analysis:

• initialization stage
• physics event loop
• finalization stage

During each of these stages, various dedicated monitoring tools are scheduled by the PerfMonSvc, eavesdropping on all the available Gaudi components [2] during a job. This is achieved by relying on the already existing auditor set of entry points depicted in figure 2. PerfMon can be engaged through various modes, which specify the level of detail. By default in all jobs, PerfMon is run in the “Semi-Detailed” mode, denoted PerfMonSD [7]. PerfMonSD fills a gap between the two standard modes of PerfMon, namely the fast-monitoring mode which has a low resource overhead but no per-component monitoring (a component being an Athena Algorithm, Tool or Service) and the detailed-mode which has both per-component and per-event monitoring. PerfMon retrieves data using the Gaudi framework auditors services e.g. ChronoStatAuditor and MemAuditor. The Gaudi Auditor Service provides a set of auditors that can be used to provide monitoring of various characteristics of the execution of Algorithms. Each auditor is called immediately before and after each call to each Algorithm instance, and can track some resource usage of the Algorithm. Calls that are thus monitored are initialize(), execute() and finalize(). PerfMon’s Semi-Detailed mode features per-component but no per-evt storage of performance metrics. It accumulates the data in custom data structures optimised for low memory and CPU footprints and outputs a text summary at the end of the job, both a slightly truncated one inside the log-file and a complete one in a separate dedicated file. PerfMonSD doesn’t monitor as many variables as the standard PerfMon modes, but only CPU, VMEM and Malloc usage of each component. PerfMonSD adds negligible overhead to a given Athena job (preferably less than 0.1% of both memory and CPU), thus making it possible to
Figure 2. Monitoring hooks at specific entry points within a Gaudi job.

have it enabled by default. PerfMonSD uses the existing PerfMon infrastructure to be able to snapshot CPU and memory states before and after component methods such as initialize(), finalize(), and execute() are invoked. The following job stages are monitored for CPU, VMem and Malloc use:

- **ini** initialisation, where measurements for the calls to "::initialize()" of Algorithms, Tools and Services are made.

- **1st** contributions from algorithms in the first event, where typically various forms of on-demand or delayed initialisation schemes are triggered.

- **cbk** a callback is typically triggered when some conditions data change and calibration constants have to be updated in components which have "registered" an interest in affected conditions.

- **evt** event loop, where measurements of calls to Algorithms "::execute()" method from the second event and beyond are called.

- **fin** finalisation, where measurements for the calls to "::finalize()" of Algorithms, Tools and Services are made.

- **dso** cost of loading libraries.

**preLoadProxy** the first client who retrieves certain conditions data will automatically trigger a retrieval of that data (and the filling of related data structures in memory). The overhead of this retrieval is measured.

3. **Systematic Performance Monitoring Infrastructure**

Systematic performance monitoring of ATLAS workflows can be broken down into five components:

**Benchmark job** ATLAS has numerous concurrent open nightly releases to serve developments in each of the data processing steps shown in figure 1. For each of these steps either one or
several benchmark jobs are defined. These benchmark jobs are chosen to be representative
of the given step being monitored.

**Job scheduler** Open nightly releases are built in their entirety once a day, and as such
benchmark jobs are scheduled to run on the given release. The job scheduler of choice
has shifted from acron [3] (CERN AFS cron service) to ATLAS’ Real Time Tester (RTT)
framework [4] in order to make use of the extensive bookkeeping and report production
facility that RTT provides. However it is foreseen with the ATLAS nightly build system
migrating to the use of the open-source software application, Jenkins [6], such that jobs in
future will have to rely also on Jenkins.

**Output** PerfMon is providing the results of monitoring in the output log of the job and in
ROOT format. Graphs and histograms are also produced in PDF files.

**Database** Currently the results from each job are saved into a file-based database,
which files the details and output of a job in a directory structure defined as
/\dd/mm/yyyy/release/platform/jobname. The database will be migrated to EOS [5],
and moreover for ease of processing and interpretation it will be extended to include an
SQLite component.

**Reports** Reports in this context refer to the production of plots and subsequent web pages,
which graphically display the evolution of the performance of the release and associated sub-
components as a time-series and also comparisons are made between production releases in
order to understand changes over longer time periods. Currently the analysis and production
of webpages is done using python, with plots made using the matplotlib module. With data
being stored in SQLite, much of the current code performing the sorting and categorisation
can be replaced with succinct SQL commands, and moreover the results can be published to
the web in the form of dashboards to allow for a more interactive user experience. Example
plots published in static web pages are shown in figure 3.

**Actions** The most important task is to act on information gathered from the reports. Actions
taken span from the manual production of JIRA bug issues to presentations in relevant
meetings, for example, to highlight that a particular algorithm or software domain has
been responsible for memory, CPU time or output file-size increases.

A typical plot is shown in figure 3, where CPU time categorized into software domain is
tracked over a two-week period in a given ATLAS nightly software release. Note the consistency
and therefore ability to resolve differences on a day-to-day basis in the software performance.
This consistency is reliant on the same benchmark machine being reserved for the job and also
on not overloading the machine with too may jobs at the same time. A series of such plots are
organised into web pages. Typically a given job type will have a dedicated webpage. Jobs are
defined on the basis of the step engaged in the data processing workflow. The steps monitored
are defined as follows:

- Generation of physics events producing “EVNT” files
- Simulation of the physical ATLAS detector response using GEANT to the physics event
  (input EVNT files, output HITS files)
- Digitisation, determine the signals read out from the detector (input HITS files to RDO
  (Raw Data Objects) files)
- Trigger, determine the trigger response ( RDO to RDO (Raw Data Objects) files)
- Reconstruction (Reconstruct physics objects components) RDO to ESD (Event Summary
  Data) to AOD (Analysis Object Data)
- Analysis Derivations (Slim, Skim, and Thin events for dedicated physics analysis streams)
  AOD to DxAOD (Derived from AOD)
The resource use measured in each step of the processing of physics events is summarised in Table 1.

![Figure 3. CPU time reports for ATLAS software algorithms.](image)

**Table 1.** CPU time and memory use in ATLAS workflows.

| Step            | Time per event (sec) | Memory per core | Number of Cores |
|-----------------|----------------------|-----------------|-----------------|
| EVNT to HITS    | 180                  | 1.7             | 1               |
| HITS to RDO     | 32                   | 1.5             | 4               |
| RDO to RDOTrigger | 10                  | 1.6             | 4               |
| RDO to ESD      | 13                   | 1.8             | 4               |
| ESD to AOD      | 0.3                  | 1.2             | 4               |
| AOD to DxAOD    | 0.4                  | 2.4             | 1               |

4. Monitoring of production workflow at Tier0

The CERN Tier0 processes data live as recorded by ATLAS from collisions in the LHC. Therefore the ATLAS release used at Tier0 is continually updated to meet varying requirements in response to changes in LHC conditions and knowledge gained from previous data processing runs. To improve the robustness and re-build frequency of the Tier0 release has been developed to test for changes to:

- file outputs
- memory consumption
• CPU time
• frequency of log file messages

in both simulated and real data workflows. The script is configured to run simultaneous tests on the same machine with and without the proposed code changes. It is a “set and forget” test that takes around 1 hour to complete on a typical interactive node (Intel Core Processor 2.4 GHz - Haswell). There is option to reduce the time by half if addition cpu cores can be accessed. Other special options include comparisons against past production releases and supplementary job configuration instructions. This script has had the effect of preventing disastrous memory leaks and new bottlenecks entering into the code base in imminent production releases.

The move to running jobs on multi-core in order for forked processes to share memory has necessitated the development of the “MemoryMonitor” tool [8], which reports the total memory consumed by the job taking into account the sharing. Specifically the crucial metric reported, as a function of time, is that of the Proportional Set Size (PSS) memory, in addition to the usual metrics of Virtual Memory, Resident Set Size memory, and Swap Space for a job process ID and all of its forked children process IDs. It does so by probing the “smaps” file for the given process IDs in the proc file system. The tool has also been extended to measure the amount of data read and written by examining the “io” file. The tool is an invaluable addition to monitoring real workflow at Tier0 and on the WLCG.

5. Optimisation studies

The current monitoring and benchmarking framework has allowed developers to:

• optimise code performance, see Table 2 and figure 4, where the results of compiler and memory allocator studies are presented, respectively.
• anticipate resource usage based on current workflows, see figure 5.
• plan resource requests and define limits for future workflows.
• react to and measure the impact of external changes such as significant code changes, for example, the move from ROOT5 to ROOT6, migration from ATLAS’ home grown Configuration Management Tool (CMT) to CMake and migration from CLHEP vectors to Eigen vectors.

| Workflow        | CPU time per event |
|-----------------|--------------------|
|                 | -O2 (default) | -Os | -O3 |
| Simulation      | 156           | 171 | 140 |
| Digitization    | 22            | 25  | 20  |
| Trigger         | 7.9           | 11.2| 7.6 |
| Reconstruction  | 10.0          | 11.4| 9.4 |

6. Conclusion

In a distributed multi-developer environment software monitoring is essential to optimisation of ATLAS workflows. The 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. The performance monitoring infrastructure has continued to evolve to help ATLAS make optimal use of available computing resources.
Figure 4. CPU time and memory consumption as a function of the different memory allocators in the workflow of ATLAS reconstruction.

Figure 5. Full reconstruction time per event as a function of the number of interactions per bunch crossing $\mu$.

References

[1] “Performing Monitoring at ATLAS” http://twiki.cern.ch/twiki/bin/view/AtlasComputing/MonitoringMemory
[2] “ATLAS Athena Framework” http://atlas-proj-computing-tdr.web.cern.ch/atlas-proj-computing-tdr/Html/Computing-TDR-21.htm
[3] “ACRON-Service” http://cern.service-now.com/service-portal/service-element.do?name=ACRON-Service
[4] “The ATLAS RunTimeTester User Guide” http://atlas-rtt.cern.ch/prod/docs/userguide/
[5] “Large Disk Storage at CERN” http://eos.web.cern.ch/
[6] “Jenkins User Handbook” http://jenkins.io/user-handbook.pdf
[7] “Semi-Detailed mode of PerfMon” https://twiki.cern.ch/twiki/bin/view/AtlasComputing/PerfMonSD
[8] “MemoryMonitor” https://twiki.cern.ch/twiki/bin/view/ITSDC/ProfAndOptExperimentsApps#MemoryMonitoringTool