ART - ATLAS Release Tester using the Grid

Tulay Cuhadar Donszelmann[1,*], *Walter* Lampl[2,**], and *Graeme A* Stewart[3,***]

[1]University of California Irvine, Irvine CA, United States of America
[2]University of Arizona, Tucson AZ, United States of America
[3]CERN, 1211 Geneva 23, Switzerland

Abstract. The ART (ATLAS Release Tester) system is designed to run test jobs on the Grid after a nightly release of the ATLAS offline software has been built. The choice was taken to exploit the Grid as a backend as it offers a huge resource pool, suitable for a deep set of integration tests, and running the tests could be delegated to the highly scalable ATLAS production system (PanDA). The challenge of enabling the Grid as a test environment is met through the use of the CVMFS file system for the software and input data files. Test jobs are submitted to the Grid by the GitLab Continuous Integration (gitlab-ci) system, which itself is triggered at end of a release build. Jobs can be adorned with special headers that inform the system how to run the specific test, allowing many options to be customised. The gitlab-ci system waits for exit status and output files are copied back from the Grid to an EOS area accessible by users. All gitlab-ci jobs run in ART's virtual machines, using docker images for their ATLAS setup. ART jobs can be tracked by using the PanDA system. ART can also be used to run short test jobs locally. It uses the same ART command-line interface, where the back-end is replaced to access a local machine for job submission rather than the Grid. This allows developers to ensure their tests work correctly before adding them to the system. In both the Grid and local machine options, running and result copying are completely parallelized. ART is written in python, complete with its own local and Grid tests to give approximately 90% code coverage of the ART tool itself. ART has been in production for one year and fully replaces and augments the former ATLAS testing system.

ATL-SOFT-PROC-2020-012
16 February 2020

1 Introduction

The offline software of the ATLAS experiment [1], Athena [2, 3], is jointly developed by more than hundred members of the ATLAS collaboration. This collaborative effort is enabled by modern software infrastructure like git for version control, continuous integration and nightly builds [4]. The Athena framework is used for most workflows needed for the ATLAS physics programme, such as simulation, reconstruction as well as online applications. To ensure that

*e-mail: tcuhadar@cern.ch
**e-mail: walter.lampl@cern.ch
***e-mail: graeme.andrew.stewart@cern.ch

the software is functional for all these use cases several levels of automated tests are done. This paper focuses on the ART (ATLAS Release Tester) testing system: ART tests can be relatively heavy, involving the processing of hundreds or even thousands of events. They are run on every nightly build. The results are checked by release coordinators to spot possible regressions with respect to a fixed reference or a previous nightly release.

2 ART Project Overview

The main purpose of ART is to test a wide array of ATLAS workflows on nightly builds. This differs from the normal use case of testing in software development (e.g., through the Python `unittest` framework [5]) in that the objective is to provide a *scalable* system for the demands of the ATLAS experiment, where running on a single node would take many days. Additionally, it requires the high level of integration with ATLAS own production and data management systems, which means that no off-the-shelf solutions exist (even amongst LHC experiments there is no common solution).

ART replaces two former ATLAS testing systems, AtNight (ATN) and RunTimeTester (RTT) which ran, respectively, short and long tests. The two systems did similar things, but had a different syntax to define tests. In particular RTT could not be used any longer because it relied on a local batch system that CERN decommissioned (Platform LSF) as well as the AFS file-system, which CERN is also phasing out. Based on the limitations identified from previous systems we derived the initial requirements for the ART system:

- Define a simple and uniform test-description.
- Run tests on a local machine or in a Grid job.
- Automate test submission for every nightly build.
- Handle a large number of tests and release branches.

ART provides these required features and, further, creates a unified command-line tool-set for users and for nightly test submissions triggered by a build of the ATLAS Athena release(s). Submission of these nightly jobs and copying of results to EOS [6] for evaluation by users are automated. The GitLab Continuous Integration (gitlab-ci) system [7] is used to orchestrate the nightly submission. Additionally, several standalone ART scripts have been created to manage the input data for the nightly tests, to automatically clean the EOS area and to perform post-processing checks on output files.

3 ART Tests

The tests to be run in ART can be shell scripts or Python scripts. They should be adorned by ART headers, which are key-value pairs written as comments in the scripting language. The headers prefixed with the number-sign (#) are typed (number, string) and may be repeated to produce lists (arrays). ART headers can be used to designate the type of test (grid, local), the input for the test and the number of cores to use for the test. Each header is strictly typed and checked for this type. Faulty headers are reported and ignored. An example of some the supported key-value pairs is:

```
# art-type:  grid | build   To run on the Grid or locally
# art-include:  <String>    Nightly releases that the script must run on
# art-input:  <String>      Name of the dataset to be read in the Grid
# art-nfiles:  <Int>        Number of files to be read from the dataset
```

The `art validate` command can be used to check the validity of the ART headers. An ART test can be constructed from several steps, usually the first being the actual command to run the ATLAS Athena software, with the next steps aiming to evaluate the outputs, such as making histograms or regression tests.

Tests are normally recursively searched for in a given script directory. All tests for a package are submitted at once.

4 Data Management for ART Tests

The input data can either be a number of files from a dataset available on the Grid [8] or a file from the CVMFS file system [9]. In the former case, the dataset name and the number of files to be read can be pointed to by an ART header. In the latter case, the input data needs to be copied to a specific EOS directory, which is then distributed on the Grid via CVMFS. Input files are located in a well defined directory structure. The `art-share` script handles the copying from EOS to CVMFS. As these files tend to be large, to avoid duplicates, all data is copied to a single folder and named with their respective SHA1 (Secure Hash Algorithm 1) sum [10]. The real file in the directory is renamed to a symbolic link to the corresponding SHA1 file. Any copy will just be another symlink, thereby avoiding data duplication. If all symlinks to a SHA1 file are removed, the SHA1 file is deleted. To the user (and for their test) the symlinked data files look exactly as before.

5 Submission of Nightly Tests

Both grid and local tests can be submitted from the gitlab-ci system. Nightly short tests are run on local virtual machines, while nightly long tests are run on the Grid. Nightly jobs are initiated by a gitlab trigger which normally comes after a release is built. Input for the tests is pre-distributed via the mechanism described in Section 4. For both short and long tests the actual gitlab-ci jobs wait for the result. After the result is received output files are copied to an EOS area for users to look at.

5.1 Grid Tests

Tests to run on the Grid are submitted via the `art submit` or `art grid` commands. The former is called by gitlab-ci while the latter can be invoked by any user to run tests on the Grid. Both commands use the ATLAS pathena [11] utility to deal with the actual submission. A number of flags are set by ART, but some flags are settable using ART headers in the tests themselves. Depending on whether the tests jobs all run with the same options, specifically with the same input, the submission of the tests is done either in `batch` or in a `single` mode. The tests, reading input from CVMFS, are grouped and submitted at once as one job to the Grid in so-called `batch` mode. This job gets split into a number of tests, and each test runs as a separate job on the Grid. The output container accommodates as many tar files as the number of jobs. The `single` mode is used when each test needs to read a separate dataset from the Grid. Each test is submitted as an individual job. This takes more time than in the `batch` mode and is only invoked when needed. In this case the output and log files have a distinct name. The output container of a `single` job consists of a single tar file.

The tar file of each job consists of output files that are specified in each test script via the `art-output` header. This is done to avoid copying unnecessary files to EOS. The outputs are downloaded to the virtual machines using the Rucio command line interface [12], then copied to EOS using the xrdcp utility [13].

The status of the grid jobs can be seen in the PanDA system [11]. ART polls PanDA to find out when a job has finished. Results and log files are copied afterwards.

5.2 Local Tests

Tests to run locally are handled by ART with the `art run` command. The tests are queued in a Python `ThreadPoolExecutor`. This executor schedules a number of tests to run in parallel. When a test finishes, a new one is taken from the queue. Some flags can be used to specify the maximum number of tests to run in parallel. Tests are run in separate processes to be able to cancel them any time when the `art run` command is interrupted, or when a test timeout occurs. The status of the tests is monitored by the `art run` command. When tests are finished their results and log files are copied to EOS using the xrdcp utility [13].

6 Output Presentation and Evaluation of Results

Once a test finishes, with good or bad exit code, the results and log files are copied to a designated area, normally some EOS area reachable by users. As tests can take a long time and can run in parallel, the results and log files are copied as soon as they are available, even if other tests in the same submission or run have not yet finished.

As indicated in Section 3, an ART test might consist of more than one command. When a test finishes, the exit code of the test is reported. To allow the tests to report error codes from each command, a special `art-result` header can be written in the log file. The log file is then scanned as soon as the test completes, and the error codes are kept for the user.

Finally, a status file is written in JSON format [14], for all the submitted tests, to show error codes and some additional information about the nightly release that was run.

7 ART Software

The ART system itself is implemented in Python 2, with a port to Python 3 underway. It depends on a number of libraries, such as Docopt [15] to handle command line processing and Requests [16] for web access. ART is implemented in several classes. Inheritance is used to factor out grid and local running, where an abstract base class handles all common functionalities. All interactions with Rucio and other systems are handled in separate classes. All classes are called from a suite of Python scripts making up the ART tool-set.

The gitlab-ci system is used to schedule jobs and wait for their output. A number of virtual machines installed with gitlab-runners [17] take care of the submission. In local mode a separate cluster of virtual machines handles the running of these jobs. In both cases Docker [18] images are used to load the correct operating system and environment. The ART architecture is shown in Figure 1.

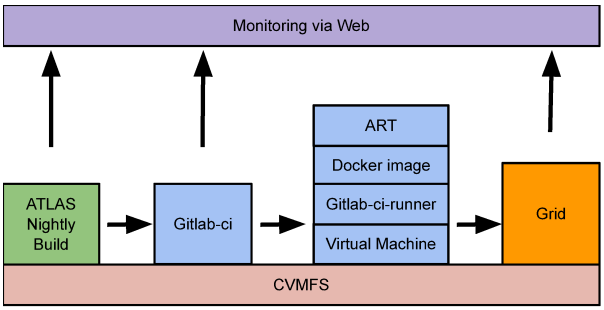

Figure 1. ART architecture for Grid submission.

A number of unit and continuous integration tests are setup to test the integrity of ART itself. These tests range from simple unit tests to full tests which submit to the Grid, wait for the output and check the results. The tests use the Python Unit Test Framework and, in order to see if the whole code is tested, the Python `coverage` [19] tool is used.

The ART project is published as an OpenSource project available not only for ATLAS but also other experiments and users [20].

8 Future Developments

The ART test system is now quite mature, but some future developments are anticipated.

First, once tests run in multi-threaded mode, some additional ART headers could be used to influence the way the local running in parallel should be handled. At this moment ART imposes a maximum number of cores to be used by a test, but once tests are multi-threaded themselves this needs to be corrected to avoid overcommitting resources.

Second, the dependency on the gitlab-ci system and the long wait for test results introduces a fragility: if the gitlab-ci system goes down, the waiting process is interrupted and is not restarted, so the copy of results is not done. It can be foreseen to use the gitlab-ci application programming interface [21] to schedule a job to run some time after the job submission has finished. This additional job can check the status of the tests, copy results if finished, or reschedule itself if not, so that a potential outage of gitlab-ci can be recovered from.

9 Conclusions

The ART (ATLAS Release Tester) is designed to test the workflows of the ATLAS nightly builds. It provides a set of tools for automatic nightly submissions via gitlab-ci, triggered at the end of nightly builds, and also supports user prompted testing. The system uses the Grid for long tests as it offers a huge resource pool. Developers have the option to run their tests on the Grid or locally using a command-line interface. The system relies on EOS and CVMFS for the input and output file management. ART has been in production more than a year and has been improved over time. It has fully replaced the former testing system and augmented its functionality.

References

[1] ATLAS Collaboration, JINST **3**, S08003 (2008), https://doi.org/10.1088/1748-0221/3/08/S08003

[2] Athena, https://doi.org/10.5281/zenodo.2641997

[3] C. Leggett et al., J. Phys.: Conf. Ser. **898**, 042009 (2017), https://doi.org/10.1088/1742-6596/898/4/042009

[4] E. Ritsch et al., J. Phys.: Conf. Ser. **1085**, 032033 (2018), https://doi.org/10.1088/1742-6596/1085/3/032033

[5] Python Unit Test Framework, https://docs.python.org/3/library/unittest.html

[6] A. Peters and L. Janyst, J. Phys.: Conf. Ser. **331**, 052015 (2011), https://doi.org/10.1088/1742-6596/331/5/052015

[7] Gitlab-ci, https://docs.gitlab.com/ee/ci/

[8] The LCG Editorial Board: LHC Computing Grid Technical Design Report. LCG-TDR-001, CERN-LHCC-2005-024, https://cds.cern.ch/record/840543, June (2005)

[9] J. Blomer et al., J. Phys.: Conf. Ser. **331**, 042003 (2011), https://doi.org/10.1088/1742-6596/331/4/042003

[10] SHA1 Sum, https://en.wikipedia.org/wiki/Sha1sum
[11] T. Maeno et al., J. Phys.: Conf. Ser. **898**, 052002 (2017), https://doi.org/10.1088/1742-6596/898/5/052002
[12] M. Barisits et al., Comput. Softw. Big. Sci. **3**, (2019), https://doi.org/10.1007/s41781-019-0026-3
[13] XRootD, https://xrootd.slac.stanford.edu/doc/xrdcl-docs/www/xrdcldocs.html
[14] JSON, https://www.json.org
[15] Docopt, https://docopt.org
[16] Requests, https://requests.readthedocs.io
[17] Gitlab-runner, https://docs.gitlab.com/runner/
[18] Docker, https://www.docker.com
[19] Python Coverage, https://coverage.readthedocs.io
[20] ART gitlab-ci project, https://gitlab.cern.ch/art
[21] gitlab-ci API, https://docs.gitlab.com/ee/api/