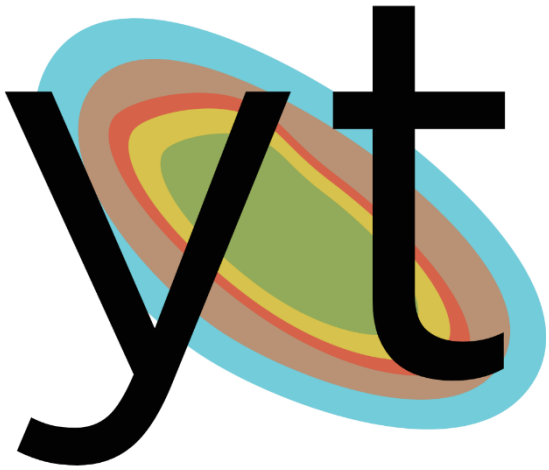
3/26/2018

Improve Test Coverage and Test Performance

Google Summer of Code, 2018



Abhishek Singh
UNIVERSITY OF MASSACHUSETTS AMHERST

Table of Contents

Abstract

This project focuses to improve yt's test framework. At present, yt's Python code coverage is only 25% (unit and answer testing) and the test runtime is approximately 45 minutes. The aim of this project is to increase code coverage and reduce test runtime.

First, I propose the use of Coveralls, which is a tool to monitor the code coverage and is free for open source repositories. This would not only help in analyzing the key areas that need immediate attention for coverage but will also help in maintaining higher code coverage for future developments.

yt's test suite could be divided into three areas, namely, Python unit tests, Cython test cases and answer testing. I will enhance the yt test suite by writing test cases for the flows that are not being tested currently. Runtime of tests could be improved by optimizing (or reducing) answer testing and image comparisons tests for visualization and volume rendering modules. This project also focuses on streamlining test cases for different geometries and data styles to improve the runtime of tests. Furthermore, the runtime of test suites varies on Linux and OSX, thereby giving us a scope of improvement.

Technical Details

1. Coveralls Integration

Coveralls is a tool to analyze code coverage with each pull request. The main advantage of this tool is that with each pull request one can know how much of the new code is being been tested.

I created a sample GitHub project to show Coveralls integration with Travis CI. Initially, I created two functions `add(a,b)` and `sub(a,b)` with a unit test case for only `add(a,b)` function. Due to this, the code coverage came out to be 86%. Then I created a PR to add the test case for `sub(a,b)` function. The effect of this PR could be easily analyzed at the coveralls build page.

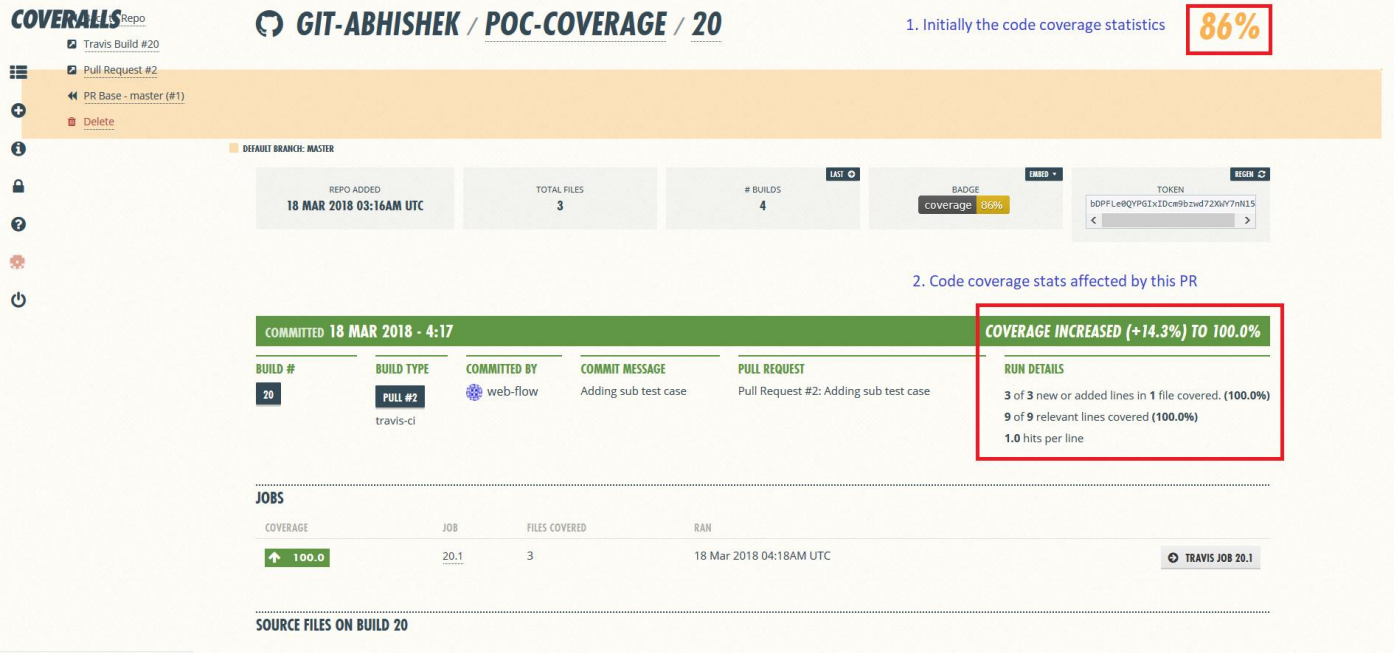


Fig 1. Coveralls build page for the PR

In addition, we also get a small summary at the pull request (PR) page depicting the overall code coverage owing to these changes. Using this we can easily track if the new changes are being properly tested or not, whether code coverage is improving which each commit or not.

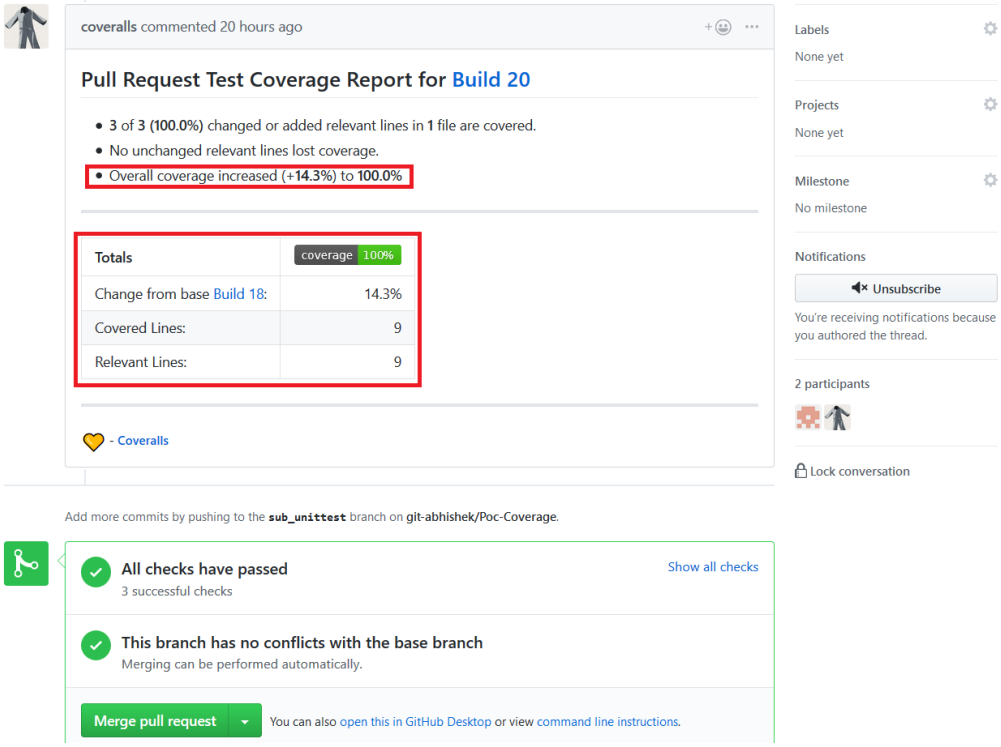


Fig 2. Coveralls summary at the PR page

2. Current Code Coverage Analysis and Improvement

Present code coverage (unit tests with answer testing in Python) of yt is 25%. The commands used for this purpose are as follows:

```
nosetests --with-coverage --with-answer-testing --cover-inclusive \
          --cover-erase --cover-html --cover-package=yt -d -v -s
```

yt is written in Python as well as Cython. Though there is already a testing framework for both, there are few key issues with them, described as follows:

- Expected Code coverage in both languages is less than 25%
- Previous attempts have been made for Cython code coverage but that has resulted in appreciably slowing down the build runtime (done by Kacper Kowalik)
- Answer testing takes a lot of time to run

With this analysis, I plan to identify yt modules that require more unit testing. This breakdown has been listed in Phase 1 and Phase 2 weekly schedule. An attempt to include as much Cython code as possible in the main test suite would be made, without affecting the runtime.

Starting point to expand test cases is by adding support for different geometries (cartesian, cylindrical, and spherical coordinates) and data styles (particle data, mesh data, including uniform resolution, octree, patch AMR, and unstructured meshes). Using the existing test functions `fake_random_ds`, `fake_amr_ds`, and `fake_particle_ds` I can define a `fake_test_datasets` generators. Using this a given functionality could be tested for different underlying geometries and data styles.

I aim to use Coverage and Nose Timer tools with the existing Nose framework. Coverage tells us which areas of code are untouched by a given code flow and thus helps in improving code coverage. Using nose-timer, we can get the runtime of a test case and thus it would help me in publishing before and after test runtime reports.

Note: Exact Cython code coverage analysis results could not be included in the report; even if we include this coverage with the Python's coverage the overall coverage would be less than 30%.

3. Improving Test Runtime

- Unit tests could be made faster by identifying the tests that are doing the heavy lifting and then simplifying them. For example, tests could be using big dataset where the same could be done by using a smaller dataset. Similarly, test cases could be using a dataset when not required.

- Besides the gain achieved by pruning the unit tests, **test runtime** could be reduced heavily by improving the answer testing and image comparison tests. Instead of these heavy tests, expected values of the function could be compared with a pre-computed value. For example:
  - Instead of pixel-by-pixel comparison of images, one could compare the perceptual hash values.

  - Using Matplotlib's image testing approach we can compare the gold images (known-correct images) with the generated image (ImageComparisonTest) and thus remove the current plot/image answer tests like FieldValuesTest, AllFieldValuesTest, ProjectionValuesTest, PixelizedProjectionValuesTest, GridValuesTest, VerifySimulationSameTest, GridHierarchyTest, ParentageRelationshipsTest, SimulatedHaloMassFunctionTest, AnalyticHaloMassFunctionTest, VRImageComparisonTest and others.

- Travis Parallelization: The entire test suite could be broken into three or more independent parts as follows:
  - simple and fast units tests
  - answer tests
  - Cython test cases

  With this split we can now utilize parallelization feature provided by Travis CI.

- As pointed by Colin Marc, OSX runtime is more than the Linux environment. One of the reasons for this is due to the `before_install` section of the travis.yml file. In OSX it takes 277.30s however, in Linux it takes 7.13s (for a random run). Thus a small amount of time could be spent on this task to see if it is feasible to reduce and get immediately a gain of 4 minutes in each build.

Schedule of Deliverables

Community Bonding Period

- April 23 - May 14
  - Connect with mentors and other community members
  - Discuss more on yt's testing framework
  - Research more on effective testing frameworks
  - Understand test code flow for different geometries and data styles
  - Understand in detail yt's testing framework
  - Set up a blog post and write the first blog

Common Task Each Week

- Write blog post showing progress, code flow understanding (1 per week)
- Maintain a public Wiki listing tasks done each day (every day)
- Write narrative documents in yt repository (if needed)
- Work on yt's bug/issue list (optional, if time permits)

Phase 1

Deliverables

```
1. Coveralls Integration
2. Finish up writing test cases for currently untouched code, increasing code
   coverage from 25% to maximum possible
3. Improve the Stream frontend by making it compatible with different
   geometries and data types
4. Enhancements in data_objects, geometry, frontends, units and fields module
```

- Week 1: [May 14 - May 18]
  - Integrate Coveralls to yt
  - Work on `data_objects` module
  - image_array.py, region_expression.py, particle_filters.py, analyzer_objects.py, grid_patch.py, unstructured_mesh.py, particle_io.py, selection_data_containers.py, derived_quantities.py, profiles.py, particle_trajectories.py, octree_subset.py, time_series.py, static_output.py, data_containers.py, construction_data_containers.py

- Week 2: [May 21 - May 25]
  - Work on `geometry` module
  - oct_geometry_handler.py, unstructured_mesh_handler.py, particle_geometry_handler.py, spec_cube_coordinates.py,

cartesian_coordinates.py, spherical_coordinates.py, cylindrical_coordinates.py, coordinate_handler.py, grid_geometry_handler.py, object_finding_mixin.py, geographic_coordinates.py, geometry_handler.py

- Cython Code: fake_octree.pyx, grid_container.pyx, grid_visitors.pyx, oct_container.pyx, oct_visitors.pyx, particle_deposit.pyx, particle_oct_container.pyx, particle_smooth.pyx, selection_routines.pyx

- Week 3: [May 28 - June 1]
  - Work on `frontends` module
  - fields.py, data_structures.py, definitions.py, data_structures.py, fields.py, fields.py, simulation_handling.py, fields.py, fields.py, utilities.py, misc.py, fields.py, data_structures.py, fields.py, io.py, fields.py, data_structures.py, fields.py, util.py, fields.py, config.py, misc.py, io.py, fields.py, simulation_handling.py, io.py, fields.py, fields.py, io.py, io.py, data_structures.py, data_structures.py, io.py, fields.py, fields.py, hilbert.py, io.py, io.py, io.py, io.py, io.py, owls_ion_tables.py, io.py, data_structures.py, io.py, fields.py, io.py, fields.py, io.py, data_structures.py, data_structures.py, fields.py, fields.py, data_structures.py, data_structures.py, particle_handlers.py, fields.py, io.py, misc.py, data_structures.py, io.py, data_structures.py, io.py, field_handlers.py, data_structures.py, io.py, io.py, io.py, io.py, data_structures.py, simulation_handling.py, io.py, data_structures.py, data_structures.py, io.py, io.py, data_structures.py, simulation_handling.py, data_structures.py, io.py, data_structures.py, data_structures.py, data_structures.py, data_structures.py, data_structures.py, io.py, data_structures.py, data_structures.py, data_structures.py, data_structures.py, data_structures.py, data_structures.py
  - Cython Code: artio/_artio_caller.pyx

- Week 4: [June 4 - June 8]
  - Work on `units` module
  - pint_conversions.py, unit_registry.py, dimensions.py, unit_systems.py, equivalencies.py, unit_object.py, yt_array.py

- Week 5: [June 11 - June 15]
  - Work on `fields` module
  - cosmology_fields.py, astro_fields.py, field_plugin_registry.py, interpolated_fields.py, geometric_fields.py, vector_operations.py, fluid_fields.py, local_fields.py, magnetic_field.py, field_exceptions.py, field_detector.py, field_info_container.py, particle_fields.py, species_fields.py, derived_field.py, xray_emission_fields.py

Phase 2

Deliverables

1. Improve volume rendering and visualization modules answer testing and image comparison by an expected answer of a test case
2. Improvement in utilities Python and Cython modules with respect to higher code coverage and reduction in test runtime

- Week 6: [June 18 - June 22]
  - Work on `visualization` module
  - particle_plots.py, line_plot.py, tick_locators.py, base_plot_types.py, streamlines.py, image_writer.py, fixed_resolution.py, color_maps.py, fits_image.py, plot_container.py, plot_window.py, profile_plotter.py, plot_modifications.py

- Week 7: [June 25 - June 29]
  - Work on `visualization_volume_rendering` module
  - volume_rendering.py, blenders.py, off_axis_projection.py, zbuffer_array.py, create_spline.py, glfw_inputhook.py, utils.py, interactive_vr_helpers.py, image_handling.py, transfer_function_helper.py, camera_path.py, camera.py, lens.py, scene.py, transfer_functions.py, render_source.py, old_camera.py

- Week 8: [July 2 - July 6]
  - Work on `utilities` module
  - hierarchy_inspection.py, chemical_formulas.py, orientation.py, particle_generator.py, png_writer.py, controller_system.py, metadata.py, utilities.py, nodal_data_utils.py, periodic_table.py, amr_kdtools.py, file_handler.py, initial_conditions.py, linear_interpolators.py, writer.py, configure.py, flagging_methods.py, logger.py, performance_counters.py, mesh_code_generation.py, libconfig.py, cosmology.py, rpdb.py, io_handler.py, fortran_utils.py, io_runner.py, task_queue.py, parameter_file_storage.py, encode.py, math_utils.py, on_demand_imports.py, amr_kdtree.py, minimal_representation.py, conversion_athena.py, exceptions.py, parallel_analysis_interface.py, sdf.py, command_line.py

- Week 9: [July 9 - July 13]
  - Work on Cython `utilities` module
  - allocation_container.pyx, alt_ray_tracers.pyx, amr_kdtools.pyx, autogenerated_element_samplers.pyx, basic_octree.pyx, bitarray.pyx,

bounding_volume_hierarchy.pyx, contour_finding.pyx, cosmology_time.pyx, depth_first_octree.pyx, distance_queue.pyx, element_mappings.pyx, fnv_hash.pyx, fortran_reader.pyx, geometry_utils.pyx, grid_traversal.pyx, image_samplers.pyx, image_utilities.pyx, interpolators.pyx, lenses.pyx, line_integral_convolution.pyx, marching_cubes.pyx, mesh_triangulation.pyx, mesh_utilities.pyx, misc_utilities.pyx, origami.pyx, particle_mesh_operations.pyx, partitioned_grid.pyx, pixelization_routines.pyx, points_in_volume.pyx, primitives.pyx, quad_tree.pyx, ragged_arrays.pyx, write_array.pyx

Final Week

- Week 10: [July 16 - July 20]
    - Buffer Week (Phase 1 or 2 tasks overflow)
- Week 11: [July 23 - July 27]
    - Buffer Week (Phase 1 or 2 tasks overflow)
- Week 12: [July 30 - August 3]
    - Buffer Week (Phase 1 or 2 tasks overflow)
    - Wrap up all the code development
    - Finish the final report to be submitted

Students Submit Code and Final Evaluations

- Week 13: [August 6 - August 14]
    - Submit the overall code contribution and reports to GSoC

Development Experience

I have 3 years 3 months of work experience as a software developer engineer at various technology companies. During these years, I worked on many different technologies ranging from Java, Python, C/C++/C#, ASP.NET, SQL to JavaScript, jQuery, HTML/CSS and learned many frameworks. This experience has not only made me a better developer but also taught me to reach for coding excellence. It exposed me to the skills that make a developer complete like working in global teams, timely delivery of production code in large codebases and taking product ownership. Thus, I feel I am competent enough to work on this project.

As part of my coursework in Neural Networks, I implemented deep learning library in Python. For the course Applied Information Retrieval, I created a toy search engine in Java that could efficiently index the corpus and return ranked query results. This semester in AI I am implementing search algorithms like A*, constraint satisfaction problems, minimax adversarial search and others. In Intelligent Visual Computing I am writing Matlab code for

3D shape reconstruction, deepnet eye detector, and marching cubes. Furthermore, I am implementing a simulator based robot in C with 9 degrees of freedom. Since all these projects are forbidden to be shared publicly, I do not have them on GitHub. (I will be required to take professors' permission, in case I need to send it privately to the reviewers.)

yt is my first open source community and thus I do not have any contributions that are publicly available. However, I have made 2 PRs for yt till now fixing issues #1680 and #1599.

Why this project?

For any software product with increasing userbase, stability and reliability are utmost. Furthermore, no organization wants to ship more features at the cost of robustness. This makes effective test suite one of the most important backbones of any product. Better testing framework not only leads to early bug detection but also contributes to rapid code development. At present, the code coverage of yt is only one-third and there is a lot of scope to improve the runtime which makes me highly interested in this project.

Increasing code coverage at the same time reducing proportional test runtime makes this an interesting problem. This demands highly optimal test code and intelligent handcrafting of test cases. This makes it a challenging problem requiring extensive analysis and well-thought-out solution.

Furthermore, this project touches the entire codebase of yt. It would not only help me understand yt better but would also make me competent to keep contributing in future. I believe when one knows the end-to-end of a system, then one can think of innovative and polished solutions of the shortcomings in the system.

Given an opportunity to work on this project, I am confident with my possessed skills and the right mentorship, that I will be able to enhance yt making its developer more confident and at the same time increasing the penchant of its userbase. I believe this would be a wonderful learning opportunity for me and would be my pleasure to give back to the open source community.

References

I deeply thank Nathan Goldbaum and Colin Marc for their valuable feedback on this project proposal. I appreciate their help and efforts!

Appendix

1. Personal Details

- Name: Abhishek Singh
- Current Degree Program: Master's in Computer Science
- Current University: University of Massachusetts, Amherst
- Past Degree Program: Master's in Mathematics, Bachelor of Engineering in Computer Science
- Past University: Birla Institute of Technology and Science, Pilani
- Phone: +1 413-824-1385
- Email/Google Hangout: abhisheksing@umass.edu
- GitHub: https://github.com/git-abhishek
- LinkedIn: https://www.linkedin.com/in/asingh690/
- Resume

2. Relevant Programming Skills

- Python, Java, C/C++
- Git and GitHub
- Cython (Beginner)

3. Work Experience

- Software Robotics Corp. (Junior Platform Engineer, 4 months)
- Citi Bank (Application Developer Supervisor, 30 months)
- Amazon (SDE Intern, 5 months)

4. Summer'18 Commitment

- Except for GSoC, I do not have any other engagements for the summer. I will be working fulltime (40 hrs/week) on this project.