Pipeline dependencies and layered test suites

Jan Philipp Thiele[1]

[1]Weierstrass Institute Berlin

March 06, 2024, deRSE24 Würzburg

Motivation


my software has many features
and more and more users...
...with different OSes. And the full testsuite runs 2 hours
...with different OSes. And the full testsuite runs 2 hours
imgflip.com

The problem in more detail

It all boils down to time and ressources

- Ideally tests run on each pull request (PR = GitLab merge request)
- They have to run somewhere
  - Local GitLab - Own hardware is the limit
  - GitHub - Concurrent jobs, etc. limited by plan (free, pro, enterprise)
  - Hybrid – avoid GitHub limits by own hardware (Talk at 1:30 by J. Fritz & T. Gruber)
  - Hybrid alternatives: Jenkins CI, CDash, etc.
- Test results important for decision on PR
- New commits trigger new test runs
  - Large testsuite slows integration of features
  - Hardware needs to run everything everytime

Way out? Don‘t run everything on each occasion, but What? When? And how?

Layered Testsuites
The shorter answer

Test different parts or configurations at different points in time

Three typical layers

- **On each pull request automatically**
  - Quick(er) tests
  - Quick code quality checks: Formatting, linting, etc.
- **After human inspection**
  - Limit what gets tested on `private` hardware when linked to GitHub
  - Only things that are considered `ready to integrate`
- **Automatically test main branch on a fixed interval (e.g. every day at 0:00)**
  - Combines all merged PRs in set time interval
  - Full testsuite
  - On as many configurations as feasible and sensible
  - Notify on failing builds or tests and keep track
  - Targeted tracking of `offending` commits based on failed tests

Next steps

Analytical side: reducing complexity

How to decide what to test in which configuration?

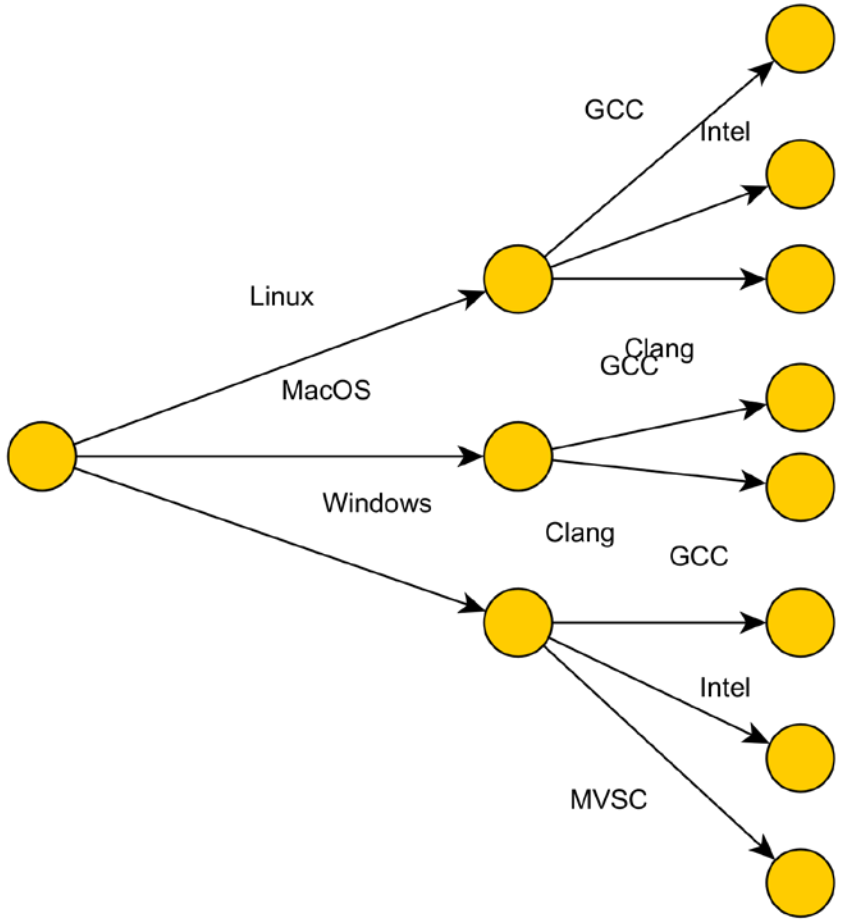
- **Be aware of combinatorial explosion**
- **Use parameter Matrices**
- **Code analysis**

Technical side

How to split up the testing?

- **Use labels or other manual triggers**
- **Hunting the `offending` commit**
- **Keeping track**
- **Notifying everyone about failing regression tests**

Combinatorial Explosion



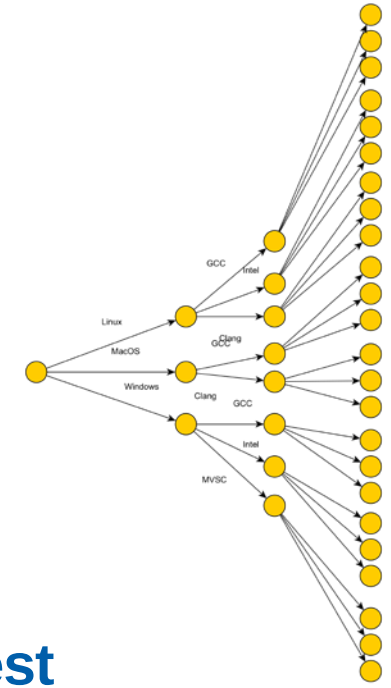
Each set of options we want to support/test multiplies the number of setups by its size.

Some common options:

- **3 operating systems (OS) = 3**
- **2-3 compilers per OS = 8**
- **3 compiler versions each = 24**
- **3 versions of 1 SW library = 72**
- **Other dependencies?**
- **Build parameters?**
- **Sequential + OpenMP + MPI?**
- **GPU support?**
- **Build Tools (Make, Ninja, etc.)?**

➢ **A huge amount of options to test**

Does each make sense? Possibly not



Parameter/Configuration Matrices

An example for a configuration matrix

		GCC			Clang			MVSC
OS	Parallel config	11.4	12.3	13.2	15	16	17	
Linux	Sequential							Not available
	OpenMP							
	MPI			Your PC				
Windows	Sequential							
	OpenMP							
	MPI							
MacOS	Sequential							Not available
	OpenMP							
	MPI							

Modify as needed
e.g. add rows or cols

Use in planning/survey
- Common combinations?
- Unavailable options?
- Compare timings to see which configs can be tested often

Use in result analysis
- Commonalities?
- System specific?
- Compiler specific?

Code Analysis
Tooling and automation

Sometimes we can anticipate which core functions we should test.

Profilers and code coverage analysis can help us choose and design tests

Profilers

- Provides timings and call graphs
- Which function is called often and how long does it take?
- Caveats:
  - Needs actual application code(s)
  - Has a sample bias

Code coverage tools

- Which lines and functions are covered by tests?
- Assists manual minmaxing:
  - Find minimal set of tests with maximum (sensible) coverage
  - Not only number but also runtime of tests

The human factor
Survey the user community

It‘s easy to overengineer the whole process!

Be mindful of the actual needs of the community

- Use user interaction (e.g. in issues or surveys)
  - Which software is actually in use
  - Are there issues with getting the software to run somewhere?
- Make your software citable
  - Check citing papers for use cases (Best case: shared user code)
  - Find out who is using your software
- Be open about (current) limitations
  - Use GitHub badges to show which configs are supported by tests
  - Use the Readme to inform about specific dependencies
  - Tell users how to suggest changes

Next steps

Analytical side: reducing complexity

How to decide what to test in which configuration?

- Be aware of combinatorial explosion
- Use parameter Matrices
- Code analysis

Technical side

How to split up the testing?

- **Use labels or other manual triggers**
- **Hunting the `offending` commit**
- **Keeping track**
- **Notifying everyone about failing regression tests**

Starting tests on pull requests based on triggers

Maybe don‘t allow every PR to run on self-hosted CI

- Costs your own ressources
- PR is in a `sensible` state
- Some person has to look at it

Jobs can be set to manual, but that has two problems

- Needs manual restart on new push
- Each job has to be started individually

Alternative: Conditional Expressions on GitHub and GitLab

- Most common: ‚on: push‘, ‚on: pull request‘
- But more fine grained options with labels and if statements
  - Have a specific label, e.g. „ready to test“
  - Statement - if: ${{ github.event.label.name == 'ready to test‘ }}
  - Many jobs can share this specific trigger
  - Has to be set only once per PR

Having a dedicated regression testing server

Many tools like CDash allow you to setup and track many configurations.

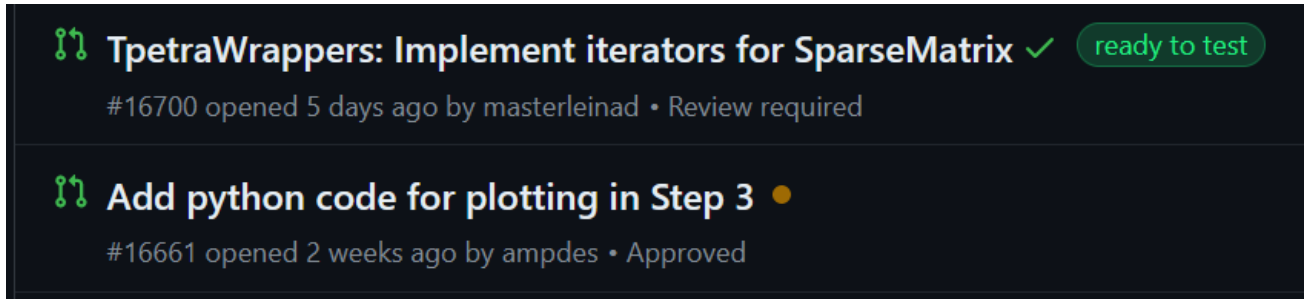
- Full test suite runs can be scheduled
- Shows results for each configuration and tested commit
  - By stage: configure, build, test
  - By result: warning, error, pass, fail, not run
- Automate hunting for the offending commit
  - You have previous result information – New error or not?
  - Targeted searching – Only run the failing tests on older commits
- Use REST API to open issues
  - Notification that tests on configuration have failed
  - Include useful information
    - Which commit(s) are the likely culprits?
    - Who authored them?
  - Close issues when the tests pass again

Layered test suites
Example: deal.II

A C++ library implementing these measures is deal.II (github.com/dealii/dealii)

There are three layers

1. GitHub Actions (most with GCC compiler and quick testsuite)
   - Static checks: clang-tidy and indent/formatting with clang-format
   - Linux debug: parallel, intel oneapi, cuda, cuda clang; release: serial (sequential)
   - OSX: serial, parallel with 64bit indices
   - Windows: MSVC 2019, MSVC 2020
2. Jenkins server (full testsuite)
   - GCC serial, GCC parallel, CUDA, OSX
   - Only runs when „ready to test“ label is set



Layered test suites
Example: deal.II

3. CDash server 23 different configurations
   - Overview page with different filters

Regression tests *23 builds*

		Update	Configure		Build		Test			
Site	Build Name	Revision	Error	Warn	Error	Warn	Not Run	Fail	Pass	Start Time
tester-tos	Clang-16.0.6-master-autodetection-20230507	1d56f8	0	0	0	0	0	1 +1	15661 -1	Feb 28, 2024 - 20:34 UTC
tester-tng	GNU-11.3.0-master-ubuntu-lts-22.04	1d56f8	0	1	0	0	0	0	15444	Feb 28, 2024 - 18:03 UTC
tester-tng	GNU-9.4.0-master-ubuntu-lts-20.04	1d56f8	0	0	0	2 +2	0	0	46	Feb 28, 2024 - 18:03 UTC
tester-tos	Clang-16.0.6-master-taskflow-20230708	1d56f8	0	0	0	0	0	0	15664	Feb 29, 2024 - 03:01 UTC
tester-tos	Clang-16.0.6-master-sanitizer-20230622	1d56f8	0	0	0	0	0	0	7794	Feb 29, 2024 - 01:48 UTC
tester-tos	GNU-13.1.1-master-64bit_indices-20230507	1d56f8	0	0	0	0	0	0	15848	Feb 29, 2024 - 00:02 UTC
tester-tos	GNU-13.1.1-master-standalone_kokkos-20230507	1d56f8	0	0	0	0	0	0	46	Feb 28, 2024 - 23:52 UTC
tester-tos	GNU-13.1.1-master-minimal-20230116	1d56f8	0	0	0	0	0	0	7926	Feb 28, 2024 - 23:05 UTC
tester-tos	Clang-16.0.6-master-unity_build-20230622	1d56f8	0	0	0	0	0	0	48	Feb 28, 2024 - 22:54 UTC
tester-tos	Clang-16.0.6-master-std=c++20-20230626	1d56f8	0	0	0	0	0	0	48	Feb 28, 2024 - 22:41 UTC
tester-tos	Clang-13.0.1-master-autodetection-20230116	1d56f8	0	0	0	0	0	0	48	Feb 28, 2024 - 22:31 UTC
tester-tos	Clang-14.0.6-master-autodetection-20230116	1d56f8	0	0	0	0	0	0	48	Feb 28, 2024 - 22:20 UTC
tester-tos	Clang-15.0.6-master-autodetection-20230116	1d56f8	0	0	0	0	0	0	48	Feb 28, 2024 - 22:09 UTC
tester-tos	GNU-13.1.1-master-std=c++20-20230626	1d56f8	0	0	0	0	0	0	48	Feb 28, 2024 - 20:22 UTC
tester-tos	GNU-10.4.1-master-autodetection-20230116	1d56f8	0	0	0	0	0	0	48	Feb 28, 2024 - 20:07 UTC
tester-tos	GNU-11.3.1-master-autodetection-20230116	1d56f8	0	0	0	0	0	0	48	Feb 28, 2024 - 19:53 UTC
tester-tos	GNU-12.2.1-master-autodetection-20230116	1d56f8	0	0	0	0	0	0	48	Feb 28, 2024 - 19:40 UTC
tester-tng	GNU-12.2.0-master-ubuntu-23.04	1d56f8	0	0	0	0	0	0	15478	Feb 28, 2024 - 18:02 UTC
tester-tng	GNU-12.2.0-master-debian-10	1d56f8	0	0	0	0	0	0	46	Feb 28, 2024 - 18:02 UTC
tester-tng	GNU-12.2.0-master-debian-11	1d56f8	0	0	0	0	0	0	46	Feb 28, 2024 - 18:01 UTC
tester-tng	GNU-12.2.0-master-debian-12	1d56f8	0	0	0	0	0	0	15478	Feb 28, 2024 - 18:01 UTC
tester-tng	GNU-12.3.0-master-debian-13	1d56f8	0	0	0	0	0	0	15478	Feb 28, 2024 - 18:00 UTC
tester-tos	GNU-13.1.1-master-autodetection-20230507	1d56f8	0	0	0	0	0	0	15670	Feb 28, 2024 - 18:00 UTC

Layered test suites
Example: deal.II

- Automatically opens an issue on fail
  - List revisions (runs)
  - Failed configurations
  - Possible culprits (merges)
  - Tags authors and mergers
  - Issue gets pinned
- Automatically closes on clean run
- Automatically reopens if closed and new revisions don‘t fix

Regression tester regressed 1d56f8 #16701

Closed tamiko opened this issue 5 days ago · 0 comments · Fixed by #16702

tamiko commented 5 days ago • edited Member

[start automated regression testsuite report]

Dear all,

this is the automated regression testsuite reporting a new regression between

- new revision: 1d56f85 regression testsuite results
- old revision: 8a6690c regression testsuite results

Summary:

- new build warnings introduced for configuration GNU-9.4.0-master-ubuntu-lts-20.04 on site tester-tng
- new test failures introduced for configuration Clang-16.0.6-master-autodetection-20230507 on site tester-tos

I have identified the following pull requests as possible candidates:

- Fix issue with colorized cylinder shell #16694 (authored-by: @blaisb; merged-by: @kronbichler)
- Template TpetraWrappers::SparseMatrix::vmult on the vector type. #16614 (authored-by: @kinnewig; merged-by: @masterleinad)

Notes:

- The regression reported above is a subset of the following full set of regressions compared to the baseline:
  - build warnings for configuration GNU-9.4.0-master-ubuntu-lts-20.04 on site tester-tng
  - test failures for configuration Clang-16.0.6-master-autodetection-20230507 on site tester-tos
- I will close and unpin this issue automatically once a full run is complete and compares cleanly to the baseline.
- If closed I will reopen the issue if the testsuite run identified an additional regression compared to what I have reported so far.

[end automated regression testsuite report]

Thank you for your attention!
Questions?