Continuous Integration at Google Scale

By John Micco

Developer Infrastructure
Google™ Speed and Scale

- >10,000 developers in 40+ offices
- 5000+ projects under active development
- 17k submissions per day (1 every 5 seconds)
- Single monolithic code tree with mixed language code
- Development on one branch - submissions at head
- All builds from source
- 20+ sustained code changes per minute with 60+ peaks
- 50% of code changes monthly
- 100+ million test cases run per day
1. Continuous Integration Goals

2. Continuous Integration at Google

3. Practical Matters
Continuous Integration

- Provide real-time information to build monitors
  - Identify failures fast
  - Identify culprit Changes
  - Handle flaky tests

- Provide frequent green builds for cutting releases
  - Identify recent green builds
  - Show results of all testing together
  - Allow release tooling to choose a green build
  - Handle flaky tests
Develop Safely

- Sync to last green changelist
- Identify whether change will break the build before submit
- Submit with confidence
- Handle flaky tests
Standard Continuous Build System

- Triggers builds in continuous cycle
- Cycle time = longest build + test cycle
- Tests many changes together
- Which change broke the build?
Google Continuous Build System

- Triggers tests on every change
- Uses fine-grained dependencies
- Change 2 broke test 1
● Identifies failures sooner
● Identifies culprit change precisely
  ○ Avoids divide-and-conquer and tribal knowledge
● Lowers compute costs using fine grained dependencies
● Keeps the build green by reducing time to fix breaks
● Accepted enthusiastically by product teams
● Enables teams to ship with fast iteration times
  ○ Supports submit-to-production times of less than 36 hours for some projects
Costs

- Requires enormous investment in compute resources (it helps to be at Google) grows in proportion to:
  - Submission rate
  - Average build + test time
  - Variants (debug, opt, valgrind, etc.)
  - Increasing dependencies on core libraries
  - Branches

- Requires updating dependencies on each change
  - Takes time to update - delays start of testing
Developing Safely - presubmit

- Makes testing available before submit
- Uses fine-grained dependencies
  - Recalculate any dependency changes
- Uses same pool of compute resources at high priority
- Avoids breaking the build
- Captures contents of a change and tests in isolation
  - Tests against head
  - Identifies problems with missing files
- Integrates with
  - submission tool - submit iff testing is green
  - Code Review Tool - results are posted to the review thread
Pending CL 30795386 : Presubmit Still Running

▼ Still Running (1)

排名第 1 仍在运行

/javatests/com/google/payments/testing/malbec/scenarios/fromconsole/sellersignup:LargeTapTests

▼ Newly Failing (1)

排名第 1 新失败

/javatests/com/google/moneta/storedvalue/service:LargeTests

▼ Newly Passing (1)

排名第 1 新通过

/javatests/com/google/checkout/external/virtualproxy/card/servers:RpcFunctionalTests

▼ Still Passing (1366)

排名第 1366 仍通过

▼ Skipped (223)

排名第 223 跳过
System assumes tests pass or fail reliably given code
  ○ Tests that don't have this property are "flaky"

Sources of test flakiness:
  ○ Infrastructure
    ■ machine failure
    ■ environment / setup problems
    ■ leakage - one test impacting another
    ■ Overloading resources
  ○ Tests
    ■ race conditions
    ■ external dependencies
    ■ timeouts
  ○ Code-under-test
    ■ memory problems
    ■ order dependence (e.g. hash tables)
Flaky Tests (cont)

- **Causes**
  - Inability to find changes breaking the build - false positives
  - Inability to identify green builds for releases
  - Wasted work for build monitors
  - Wasted compute resources
  - Inappropriately failing presubmits - wasting developer time

- **Solutions (Google does all of these):**
  - Fix them!!!
    - Difficult - requires developer time
  - Hide them
    - Retry causing delays
    - Identify infrastructure flakes
    - Use metrics to ignore
  - Track them
    - Provide metrics to prioritize fix / hide
• **Sources of growth in test execution time**
  ○ More developers = increased submission rate
  ○ More tests
  ○ Longer running tests
  ○ Tests consuming more resources (threading)

• **Examine the growth trends**
  ○ Predict compute needs
  ○ Look for any build system features required
Test Growth

- **Problems**
  - Quadratic execution time growth
  - Ultimately cannot run every affected test @ every change
  - Low latency results still top requirement

- **Solution**: Just in time scheduling (JIT)
Continuous Integration:

- Run every test affected at every changelist.

In Production:

- Build and run tests concurrently on Google’s distributed build and test backend.
JIT Scheduling

Schedule tests to run only when system has capacity.

Produce project-wide results at periodic changelists.
Same User Experience; Lower Cost

Culprit finding

- Failures / breaks between changes may be more difficult to localize to the offending change.

- **Short-term:** Command-line tool to find culprits

- **Longer Term:** Integrated automatic culprit finding
Same User Experience; Lower Cost

Flaky Tests
- Tests which only pass some of the time could cause fewer green statuses for projects.

- **Short Term**: Optionally retry failed tests

- **Longer Term**: Tightly integrated flake mitigation and automatic / manual re-running of suspected flakes
Q & A

For more information:

- [http://www.youtube.com/watch?v=b52aXZ2yi08](http://www.youtube.com/watch?v=b52aXZ2yi08)
- [http://google-engtools.blogspot.com/](http://google-engtools.blogspot.com/)
- [https://www.youtube.com/watch?v=KH2_sB1A6I&feature=youtube_gdata_player](https://www.youtube.com/watch?v=KH2_sB1A6I&feature=youtube_gdata_player)