GDB tests, CI & Buildbot BoF

Sergio Durigan Junior
sergiodj@redhat.com

License

- License: Creative Commons Attribution 4.0 International License (CC-BY-4.0)
- `https://creativecommons.org/licenses/by/4.0/`

Nomenclature

- Worker: The node that performs the "build". Usually one per physical machine/VM. For example, `fedora-x86_64-1` or `ubuntu-aarch64`.
- Factory: A recipe of how to perform a build.
- Builder: An instance of a factory. For example, `Fedora-x86_64-m64` or `Ubuntu-Aarch64-native-extended-gdbserver-m64`.
- Scheduler: Dispatches jobs to a set of builders. Can be triggered by specific events like a commit in a repository, a *try build* request or like a cronjob.

How was it?

- GDB Buildbot started in 2015 as a personal project.

How was it?

- GDB Buildbot started in 2015 as a personal project.
- We just had 2 machines serving 4 Fedora `x86_64` workers at the time. And no *try builds*!

How was it?

- GDB Buildbot started in 2015 as a personal project.
- We just had 2 machines serving 4 Fedora `x86_64` workers at the time. And no *try builds*!
- Initially it stored the test results in a git repository. This proved too inefficient over time...

And now?

- The master runs in a dedicated VM at OSCI (**O**pen **S**ource **C**ommunity **I**nfrastructure).

And now?

- The master runs in a dedicated VM at OSCI (**O**pen **S**ource **C**ommunity **I**nfrastructure).
- Most of our builders support *try builds*!

And now?

- The master runs in a dedicated VM at OSCI (**O**pen **S**ource **C**ommunity **I**nfrastructure).
- Most of our builders support *try builds*!
- 14 workers (11 machines):

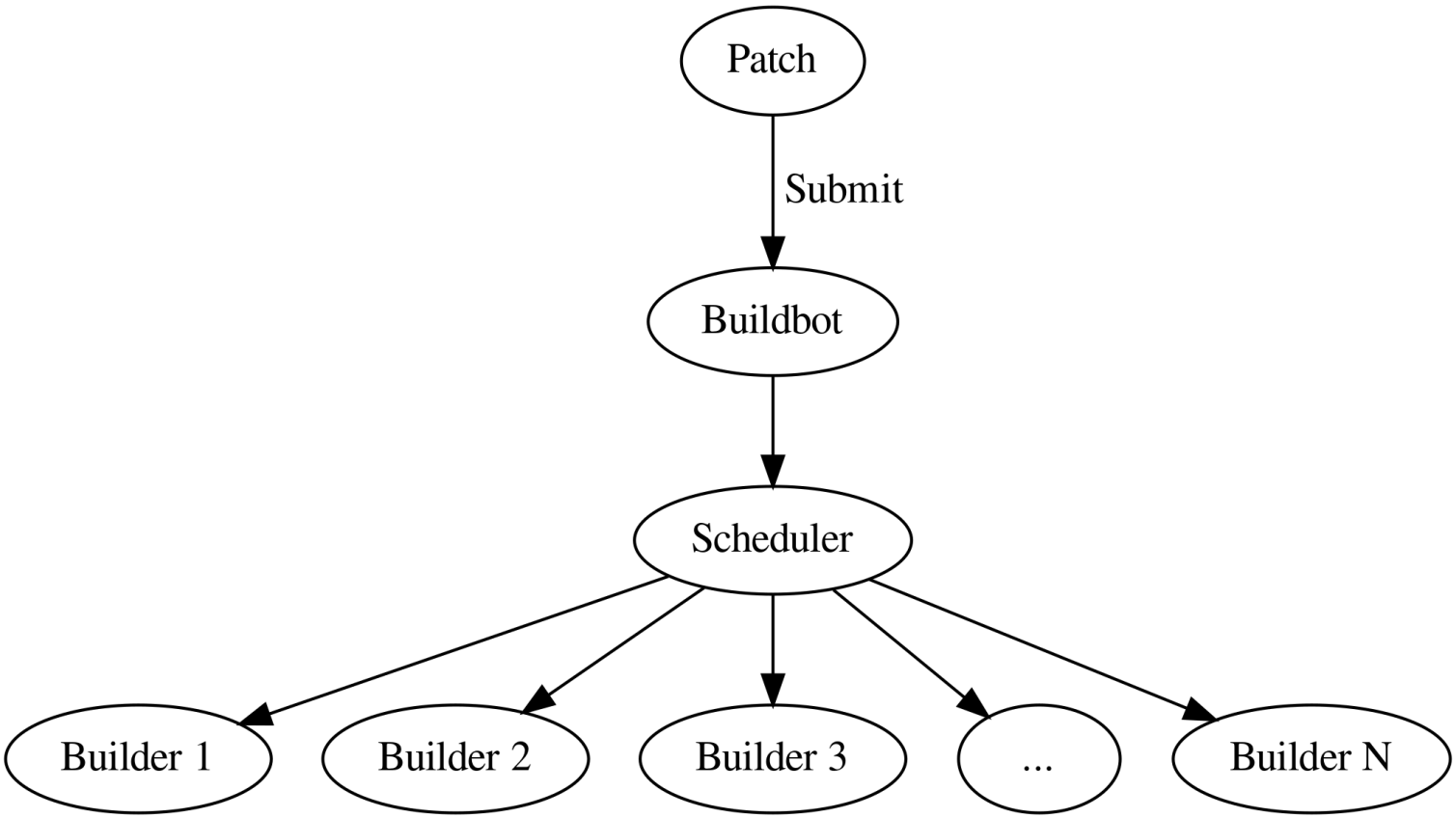
And now?

- The master runs in a dedicated VM at OSCI (**O**pen **S**ource **C**ommunity **I**nfrastructure).
- Most of our builders support *try builds*!
- 14 workers (11 machines):
  - Sergio (Red Hat): 2 machines (Fedora `x86_64`)
  - Alan Hayward (ARM): 2 machines (Ubuntu `ARM` 32 and 64)
  - Rainer Orth (CeBiTec.Uni-Bielefeld.DE): 2 machines (Solaris `amd64` and `sparcv9`)
  - David Edelsohn: 3 machines (RHEL 7.1 `s390x`, AIX `POWER8` and Debian Jessie `s390x`)
  - Edjunior Machado: 1 machine (CentOS 7 `PPC64LE`)
  - Mark Wielaard: 1 machine (Fedora `s390x`)
  - Kamil Rytarowski: 1 machine (NetBSD `amd64`)

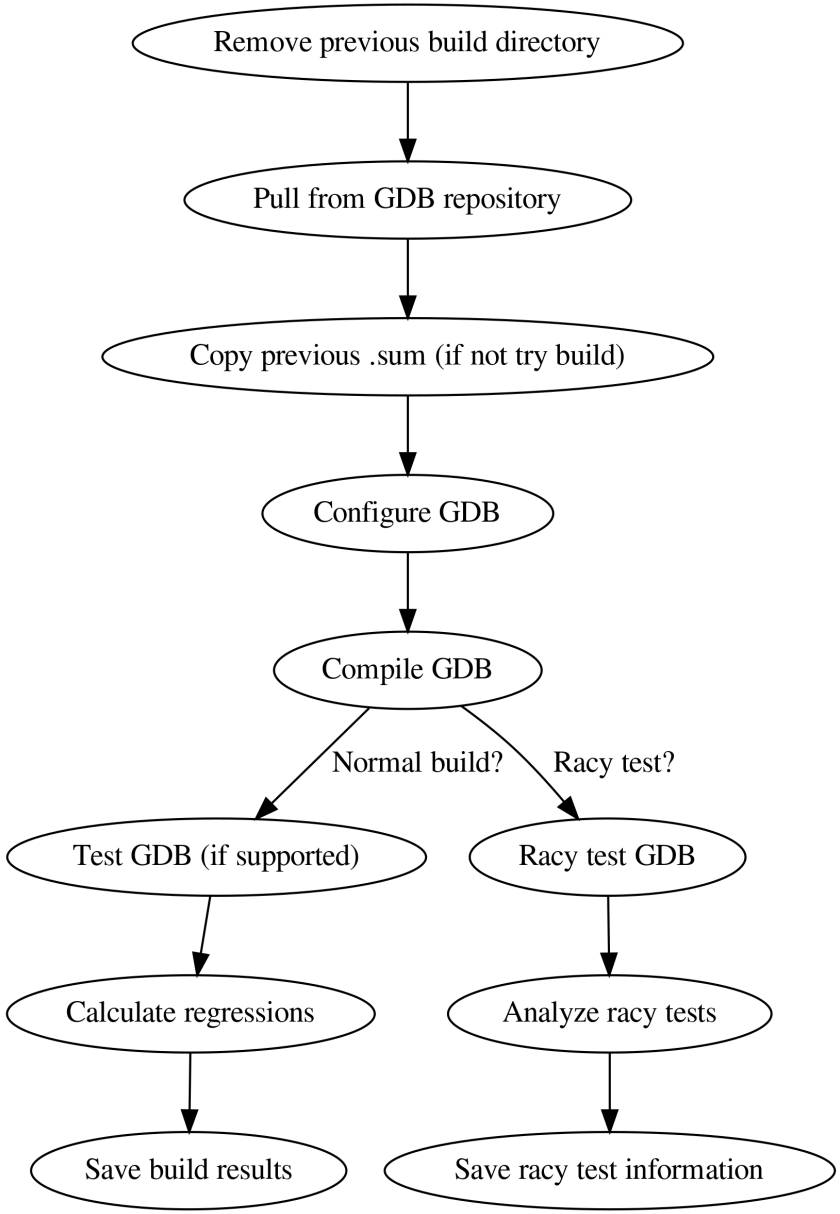
And now?

- The master runs in a dedicated VM at OSCI (**O**pen **S**ource **C**ommunity **I**nfrastructure).
- Most of our builders support *try builds*!
- 14 workers (11 machines):
  - Sergio (Red Hat): 2 machines (Fedora `x86_64`)
  - Alan Hayward (ARM): 2 machines (Ubuntu `ARM` 32 and 64)
  - Rainer Orth (CeBiTec.Uni-Bielefeld.DE): 2 machines (Solaris `amd64` and `sparcv9`)
  - David Edelsohn: 3 machines (RHEL 7.1 `s390x`, AIX `POWER8` and Debian Jessie `s390x`)
  - Edjunior Machado: 1 machine (CentOS 7 `PPC64LE`)
  - Mark Wielaard: 1 machine (Fedora `s390x`)
  - Kamil Rytarowski: 1 machine (NetBSD `amd64`)
- Test results are stored directly on-disk, and "garbage-collected" every week (tests older than 4 months are deleted).

How does it work?



How does it work? [2]



Racy tests handling (or an attempt to)

- We keep a list of racy tests (detected weekly through the racy build analysis).

Racy tests handling (or an attempt to)

- We keep a list of racy tests (detected weekly through the racy build analysis).
- When a racy build finishes, we include the racy tests in the `xfail` file for that builder.

Racy tests handling (or an attempt to)

- We keep a list of racy tests (detected weekly through the racy build analysis).
- When a racy build finishes, we include the racy tests in the `xfail` file for that builder.
- We then ignore them when doing normal test builds. However... whac-a-mole.

Test analysis (a.k.a. finding regressions)

- Transform the current `.sum` file into a Python dict:
  - `{ 'gdb.base/test1.exp: name1' : 'PASS',`
    `'gdb.base/test1.exp: name2' : 'FAIL', ...}`

Test analysis (a.k.a. finding regressions)

- Transform the current `.sum` file into a Python dict:
  - `{ 'gdb.base/test1.exp: name1' : 'PASS',`
    `'gdb.base/test1.exp: name2' : 'FAIL', ...}`
- Do the same for the previous `.sum` file.

Test analysis (a.k.a. finding regressions)

- Transform the current `.sum` file into a Python dict:
  - `{ 'gdb.base/test1.exp: name1' : 'PASS',`
    `'gdb.base/test1.exp: name2' : 'FAIL', ...}`
- Do the same for the previous `.sum` file.
- Iterate over the current `.sum` file's dictionary and do:

Test analysis (a.k.a. finding regressions)

- Transform the current `.sum` file into a Python dict:
  - `{ 'gdb.base/test1.exp: name1' : 'PASS',`
    `'gdb.base/test1.exp: name2' : 'FAIL', ...}`
- Do the same for the previous `.sum` file.
- Iterate over the current `.sum` file's dictionary and do:
  - If the current key is `XFAIL`'ed (i.e., a racy test), ignore it.

Test analysis (a.k.a. finding regressions)

- Transform the current `.sum` file into a Python dict:
  - `{ 'gdb.base/test1.exp: name1' : 'PASS',`
    `'gdb.base/test1.exp: name2' : 'FAIL', ...}`
- Do the same for the previous `.sum` file.
- Iterate over the current `.sum` file's dictionary and do:
  - If the current key is `XFAIL`'ed (i.e., a racy test), ignore it.
  - If the current key exists in the new dict:
    - If it has the same value, good (not a regression).
    - If it changed from `PASS` to `FAIL`, bad. Report as a regression.
    - If it changed from `FAIL` to `PASS`, good. Update the baseline.

Test analysis (a.k.a. finding regressions)

- Transform the current `.sum` file into a Python dict:
  - `{ 'gdb.base/test1.exp: name1' : 'PASS',`
    `'gdb.base/test1.exp: name2' : 'FAIL', ...}`
- Do the same for the previous `.sum` file.
- Iterate over the current `.sum` file's dictionary and do:
  - If the current key is `XFAIL`'ed (i.e., a racy test), ignore it.
  - If the current key exists in the new dict:
    - If it has the same value, good (not a regression).
    - If it changed from `PASS` to `FAIL`, bad. Report as a regression.
    - If it changed from `FAIL` to `PASS`, good. Update the baseline.
  - If the current key doesn't exist in the new dict:
    - If it's a `PASS`, good. Update the baseline.
    - If it's a `FAIL`, bad. Report as a new failure.

Notifications

- To *gdb-testers*: whenever we detect a possible regression in an upstream commit.

Notifications

- To *gdb-testers*: whenever we detect a possible regression in an upstream commit.
- To the *author*: on *try builds*, or when his/her commit broke GDB.

Notifications

- To *gdb-testers*: whenever we detect a possible regression in an upstream commit.
- To the *author*: on *try builds*, or when his/her commit broke GDB.
- To *gdb-patches*: when a commit breaks GDB.

Notifications

- To *gdb-testers*: whenever we detect a possible regression in an upstream commit.
- To the *author*: on *try builds*, or when his/her commit broke GDB.
- To *gdb-patches*: when a commit breaks GDB.
- Breakage notifications are usually reliable. Regression notifications are not (just look at *gdb-testers*).

Problems and challenges

- Racy testcases. Perhaps the most difficult/persistent problem?

Problems and challenges

- Racy testcases. Perhaps the most difficult/persistent problem?
- Lots of test messages are non-unique. This makes it really hard to compare test results and find regressions.

Problems and challenges

- Racy testcases. Perhaps the most difficult/persistent problem?
- Lots of test messages are non-unique. This makes it really hard to compare test results and find regressions.
- Better way to store and retrieve test results (current way is "enough" for what we need, but it can certainly be improved). See Serhei's work and Keith's work.

Problems and challenges

- Racy testcases. Perhaps the most difficult/persistent problem?
- Lots of test messages are non-unique. This makes it really hard to compare test results and find regressions.
- Better way to store and retrieve test results (current way is "enough" for what we need, but it can certainly be improved). See Serhei's work and Keith's work.
- `make -jN`, racy tests and `gdb.threads`.