

Analyzing historical DejaGNU test result data with the Bunsen toolkit

Serhei Makarov
<me@serhei.io>
for LPC/GNUTools Track
September 20, 2021

About me

- Software Engineer, Red Hat (~3.5yrs)
- Working on the SystemTap project (~5yrs)
- Working on tools to get a handle on our testing situation:
 - Similar to other tools projects: small development team, large DejaGNU testsuite, Buildbots for CI, changing dependencies
 - SystemTap is fairly robust to regressions causing fatal errors, but non-fatal functionality degradation creeps in over time....

In this talk

- Why keep a historical archive of test results?
- Overview of the Bunsen toolkit
- Examples of analysis on historical data:
 - Filtering recent regressions
 - Finding the origin of a regression
 - Identifying ‘flaky’ testcases

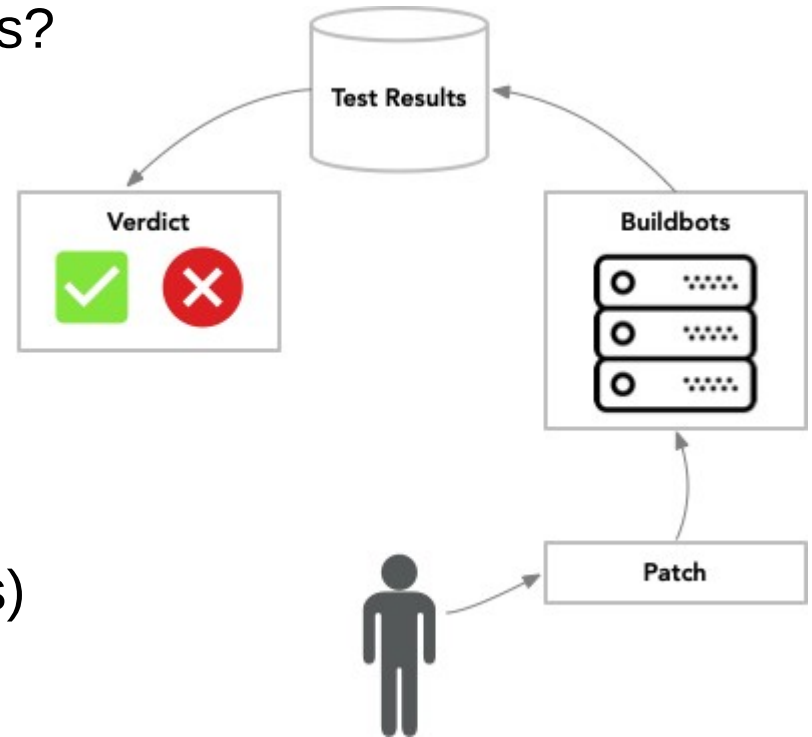


Overview of the problem

Why keep a historical archive of test results?

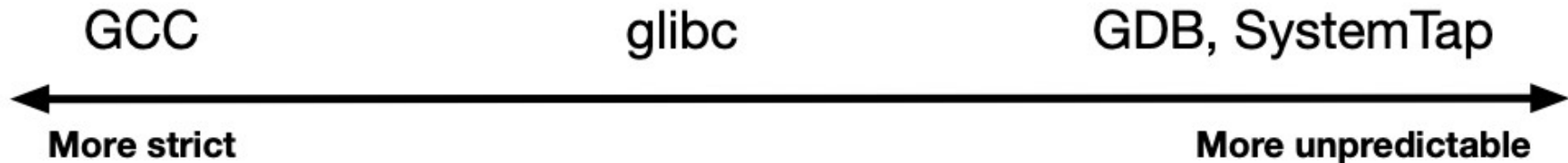
Overview of the problem

- Why keep a historical archive of test results?
- Typical CI blueprint:
 - Automatically test proposed patch
 - Compare result to 'known-good' version
 - Accept or reject based on regressions
- Accept or reject decision is very basic
- No need for long-term storage (>~2months)
- Solves a different problem from ours:
 - Reduce a firehose of incoming patches



Overview of the problem

- Can basic CI guard against regressions creeping in long-term?
- Based on several assumptions:
 - Testsuite is clean (no spurious failures, no nondeterministic behaviour)
 - CI covers all relevant system configurations
 - Regressions aren't caused by changes in OS and dependencies
- May not be true for some projects as they are now





Overview of the Bunsen toolkit

Collect, store, and analyze DejaGNU test result data

Bunsen toolkit

- Collect DejaGNU test results, parse them, produce a JSON index
- Store the test results and index in a Git repo
- Use Git de-duplication to achieve a high compression factor
 - 2019, GDB: 8758 testruns to 3.2GB (~200x from 648GB raw)
 - 2019, SystemTap: 1562 testruns to 2.7GB (~38x from 103GB raw)
 - 2021, SystemTap: 4158 testruns in 7.8GB (~42x from ~333GB raw)
- Python library for querying the indexed test results
- **This talk:** Analysis scripts to extract information from the Bunsen repo

Bunsen data model

- Each testrun has *commit IDs*, a *system configuration* and a list of *testcases*
 - `version` or `source_commit`: Git commit id of **source code** in **project repo**
 - Optional -- could be replaced by ID of submission in a CI system.
 - Or any other `version` that allows us to sort project versions chronologically.
 - `bunsen_commit_id`: Git commit id of **test results** in **Bunsen Git repo**
 - Commit stores the original DejaGNU `.sum` and `.log` uploaded to Bunsen
 - Together with auxiliary log files (e.g. system diagnostics, `dmesg`)
- System configuration is a set of key-value pairs, most commonly:
 - `osver`: Linux distribution (fedora-35-rawhide, fedora-34, ubuntu-20.04, etc.)
 - `arch`: hardware architecture (x86_64, aarch64, ppc64le, etc.)
 - Ideally also: versions of dependent components (`kernel_ver`, `gcc_ver`, etc.)

Bunsen data model

- Each testrun has *commit IDs*, a *system configuration* and a list of *testcases*
- Testcases are (name, outcome, subtest) tuples matching DejaGNU semantics
 - name is the name of the .exp testcase
 - subtest is the string “identifying” the subtest (ambiguous in many testsuites)
 - outcome is PASS, FAIL, XPASS, XFAIL, etc.
- Two options for storing testcase data: full, and compact
 - Full format stores a testcase entry for every subtest of every testcase
 - Compact format stores only FAIL subtests separately, and consolidates any .exp testcase with a PASS outcome into a single entry
 - In general, analysis assumes “absence of failure” means “no problem”

Why historical analysis?

- There are simple utilities that diff a pair of DejaGNU test results
- But diffing against a 'known-good' version doesn't give a full picture:
 - No 'known-good' version with a clean testsuite
 - Regression occurs on configuration that isn't tested regularly
 - Regression caused by changes in OS and dependencies
- Results must be interpreted against historical context



LINUX September 20-24, 2021
PLUMBERS
CONFERENCE

Example analysis

Filtering recent regressions

Level 1: ‘grid view’ analysis

Based on earlier scripting developed by Martin Cermak:

- For each .exp testcase in the testsuite
 - For each system configuration
 - For each `source_commit` / `version` in specified range
 - Report + if the testcase outcome is PASS (for all subtests)
 - Report - if the testcase outcome is FAIL (for some subtest)
 - Report ? if this combination was not tested
 - Output the result as a table (configuration x `source_commit`)

Level 1: 'grid view' analysis

systemtap.syscall/syscall_consistency.exp

last	first	arch	osver	988f439	1a4c755 f3cc67f	7615cae 4711113	afe0acf 8fcc756	7db5419	91087d8 3df5453	f75ec91
+	+	x86_64	None							
+	+	x86_64	rhel-8	+	+			+		
-	+	x86_64	rhel-7	-77				-77		
+	+	x86_64	fedora-33-rawhide							
+	+	x86_64	fedora-31	+				+		
+	+	x86_64	fedora-34					+	+	
-	+	s390x	rhel-8					-74		
-	+	i686	rhel-6				-65	-65		
+	+	aarch64	rhel-8	+			+	+		
+	+	x86_64	fedora-33	+	+			+	+	
-	+	ppc64le	rhel-8							
-	+	i686	fedora-30							
-	+	x86_64	ubuntu-18-04	-78	-78		-78	-78		
-	+	x86_64	rhel-6				-70	-70		

...

ee34127	90f9123	8a7e5b0	e3529b3	bc86fc8	72f1927	5f74d0d	55de61e	feb0327	41040e6	b0cccb3	d3afb38	6bb4e12	2643825	c481017	a708b90	875469f	c71391b	5f564f4	4fe0f0
							+	+	+			+		+	+	+	+	+	+
+	+					+		+	+			+		+			+	+	+
7	-77	-77			+			+	+			+		+			+	+	+
								+	+			+		+			+	+	+
									+								+		
74	-74	-74			+		+	+	+			+							+
65	-65	-65			+		+	+	+			+	+	+	+		+	+	+
+	+	+					+	+	+			+							
+	+	+			+			+	+				+	+			+	+	+
-74	-74	-74					+	+	+										
														+			+	+	+
-78	-78	-78					+	+	+			+		+					
-70	-70	-70			+		+	+	+			+	+	+	+		+	+	+

Level 2: 'time cube' filtering

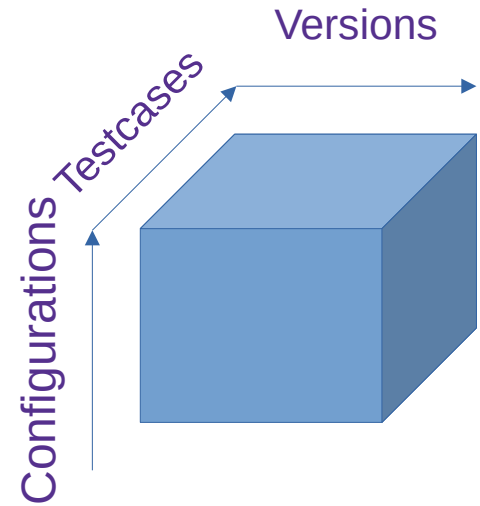
Grid view required lots of scrolling and coffee to assess all testcases.

- For improved filtering, assemble a grid as in the previous script
 - keys $\rightarrow$ (testcase x config x version)
 - values (+ - ?) $\rightarrow$ set(name x outcome x subtest)
- For each (testcase x config x version) in chronological order:
 - *“Use prior history to decide if this test result is noteworthy”*
- For each testcase with noteworthy test results:
 - List (config x version) $\rightarrow$ (name x outcome x subtest) in reverse chronological order

Level 2: 'time cube' filtering

Detail of 'time cube' data structure:

```
class Timecube:
    def __init__(self,...):
        self.outcomes_grid = {} # grid_key -> PASS,FAIL
        self.subtests_grid = {} # grid_key -> set of (name,outcome,subtest)
        self.prev_tested = {} # grid_key -> grid_key for prev tested version
        self.next_tested = {} # grid_key -> grid_key for next tested version
    def grid_key(self, testcase, configuration, commit):
        """Returns string ID of the specified grid cell."""
    def scan_commits(self):
        """Populate the grid."""
    def iter_scan_commits(self):
        """Populate the grid while yielding (commit, testruns)."""
    def iter_commits(self):
        """Yield (commit, testruns) in chronological order."""
    def iter_testcases(self):
        """Yield testcase names."""
    def commit_dist(self, v1, v2):
        """Returns distance in # of commits between v1 and v2."""
```



Current scanning performance: slightly tolerable (~454*15*143 cells in ~76sec)

Level 2: ‘time cube’ filtering

“Use prior history to decide if the test result is noteworthy”: some options:

- Report every change PASS ↔ FAIL (too much information)
- Report first change PASS → FAIL and last change FAIL → PASS
- Consider changes between consecutive runs of PASS/FAIL and in-between ‘flaky’ periods where changes are more frequent
 - When encountering a change, update the last ‘solid’ period of identical results, and the prior ‘flaky’ period of unstable results
- Other options that correlate results across different configurations?

Level 2: 'time cube' filtering

commit_id: 90f9123bb1406aca27c8135b99549f6c1ed8d42c

summary: PR26015: Make syscall arguments writable again

gitweb_info: [commit](#), [commitdiff](#)

baseline	latest	testcase	info
FAIL	PASS	systemtap.apps/busybox.exp	seen on 1 configurations
INIT	PASS	systemtap.base/probewrite.exp	seen on 8 configurations
FAIL	PASS	systemtap.base/proc_exec.exp	seen on 1 configurations
PASS	FAIL	systemtap.base/target_set.exp	seen on 3 configurations
FAIL	PASS	systemtap.base/temp-directory.exp	seen on 1 configurations
PASS	FAIL	systemtap.base/utrace_p5.exp	seen on 1 configurations
PASS	FAIL	systemtap.base/vma_vdso.exp	seen on 2 configurations
PASS	FAIL	systemtap.context/usymbols.exp	seen on 3 configurations
PASS	FAIL	systemtap.printf/print_user_buffer.exp	seen on 3 configurations
FAIL	PASS	systemtap.server/client.exp	seen on 1 configurations

systemtap.syscall/syscall_consistency.exp

baseline	latest	commit_id	summary	info
PASS	FLAKE	acd978b	stp_task work: don't busy poll in stp_task_work_exit()	seen on 1 configurations
PASS	FAIL	90f9123	PR26015: Make syscall arguments writable again	seen on 6 configurations
INIT	PASS	49e0a88	testsuite: fix buildok perms	seen on 1 configurations
INIT	PASS	e85ffdc	PR26755 temporary kprobes_onthefly.exp: also disable m* on ppc	seen on 2 configurations
INIT	PASS	2f7e379	PR26755 kprobes_onthefly.exp: skip lock_* tracepoints pending investigation	seen on 2 configurations
INIT	PASS	aad9a0e	stapbpff: fix module name	seen on 1 configurations
INIT	PASS	046fa01	RHBZ1847676 cont'd: one more uprobes-inode/onthefly	seen on 1 configurations



Example analysis

Finding the origin of a regression

Finding the origin of a regression

If we are interested in the history of a particular testcase:

- For each (testcase x config x version) in chronological order:
- Assemble the time cube for one testcase only:
 - keys $\rightarrow$ (config x version)
 - values (+-?) $\rightarrow$ set(name x outcome x subtest)
- For version in reverse chronological order
 - Report any changes compared to the earlier version.



Example analysis

Identifying 'flaky' testcases

Identifying ‘flaky’ testcases

Use (source_commit x configuration) tuples for which there are multiple testruns in the Bunsen repo to identify nondeterministic testcases:

- For each (source_commit x configuration) in the Bunsen repo
 - all_testcases = {}
 - For each testrun matching (source_commit x configuration)
 - For each (name x subtest x outcome) tuple in testrun
 - Append testrun to all_testcases[name x subtest x outcome]
 - For each (name x subtest x outcome) tuple in all_testcases
 - If tuple didn't appear in all Testruns, flag name as nondeterministic
- Output the list of testcase names that were flagged

Conclusions

- Historical analysis gives clarity about the state of the testsuite
- More & nastier development work needed to make Bunsen adaptable
- Bunsen development roadmap:
 - High priority – more work on the DejaGNU parser(s)
 - High priority – more options for getting logs into the Bunsen repo
 - High priority – more configurations (no ‘modify script to configure’)
 - High priority – repo bookkeeping (update/delete/gc testruns)
 - Medium priority – email reporting
 - Medium priority – better web interface (with Flask framework)
 - Low priority – hardening the web interface

Conclusions & Questions

- Brief overview of parser & test result availability for projects:
 - SystemTap: solid parser; (private) CI active; (private) archives.
 - Some info on our CI system in [blog post 1](#), [blog post 2](#)
 - GDB: solid parser (thanks to Keith Seitz); CI defunct; no archives.
 - <https://gdb-buildbot.osci.io/#/>
 - glibc: no parser; CI in-progress; no archives.
 - <https://sourceware.org/glibc/wiki/CICDDesign>
 - gcc: no parser; CI active; no archives.
 - <https://gcc.gnu.org/jenkins> → huge firehose of data.
- ‘Best practice’ for extracting data from CI on ongoing basis?
- Usefulness of retroactive testing?



Thank You

- Bunsen project
 - Examples for this talk → <https://people.redhat.com/~smakarov/2021-lpc-talk/>
 - <https://sourceware.org/git/?p=bunsen.git;a=summary>
 - <mailto:bunsen@sourceware.org>
 - <https://github.com/serhei/bunsen>
- SystemTap project
 - <https://sourceware.org/systemtap/>
 - <https://sourceware.org/git/gitweb.cgi?p=systemtap.git;a=summary>
 - #systemtap on OFTC
- Special thanks to: Keith Seitz, Frank Ch. Eigler, Martin Cermak