LINUX TESTING WITH TAPPER
Complexity in a nutshell

Steffen Schwigon, AMD Operating System Research Center
October 28, 2011
Public
OVERVIEW | Abstract

The Operating System Research Center (OSRC), a global AMD Research organisation headquartered in Dresden, Germany, acts as a bridge between the OS development community and the worldwide AMD processor design community.

At the OSRC we run a test infrastructure to test Linux in many orthogonal dimensions: hardware generations, software visible features, kernel branches, Linux-based distributions, virtualization with upstream or distro-specific Xen and KVM, multi-machine scenarios, and running in simulators. Inside of those dimensions we cover regression, functional, and stress tests, benchmarks, guest migration, and reboot and suspend/resume tests.

This talk will give an overview of our test infrastructure (codename "Tapper") and dive deeper into some interesting technical topics like the machine scheduler and the query interface, show the combination of open-source standard protocols and tools to glue everything together, and how we break down that complexity into easy but powerful, scriptable APIs with no client-side toolchain dependencies for the users.
OVERVIEW | Context

- Operating System Research Center (OSRC)
- We developed and run a test infrastructure called Tapper
  - Automated testing of operating systems and virtualization (Xen/KVM)
  - Published as open source in 2011
    - Mailing list: [http://www.amd64.org/mailman/listinfo/tapper](http://www.amd64.org/mailman/listinfo/tapper)
    - Source: [http://github.com/amd](http://github.com/amd)
OVERVIEW | Agenda

- Overview
  - Mission
  - Test approaches
  - Test infrastructure
  - Automation
  - Web GUI

- Testing
  - Understanding the test protocol
  - From visible simplicity to hidden complexity

- Automation

- Result evaluation
  - From hidden complexity back to visible simplicity
SCOPE | Mission

- OS distribution testing
  - Partner distributions: Novell (SLES), Red Hat (RHEL)
  - Community distributions: openSuse, Fedora, (Debian, Ubuntu)
  - Windows, as guest
- Linux kernel
  - OSRC contributions
  - Regressions (“Attack of the alien patches”)
- Virtualization
  - Xen
  - KVM
- AMD hardware

→ Combinations of all the above
Functional testing of Linux kernel

- “Classic QA”
- For OSRC enablement work
- Iterate Linux kernel + hardware + developer repositories
TEST APPROACHES | 2/3: Virtualization matrix

- Virtualization matrix
  - ➔ Find new problems
  - Distro Xen/KVM vs. upstream releases
  - Huge matrix of host/guest combinations
  - Stress system (use benchmarks in guests)
  - Iterate Xen/KVM + hardware
Testplans

- **Ensure no regressions**
- Dedicated scenarios for points of interest
- Bridge to “TaskJuggler” planning software
  - Bi-directional: scheduling + reporting
- On top of the other approaches
OVERVIEW | Test infrastructure
BASIC PRINCIPLES

- Test infrastructure “Tapper” -- basic principles:
  - Zero overhead to write tests and report results
  - Flexible evaluation: easy web GUI + scriptable API
  - Optional but advanced automation
BASIC PRINCIPLES

- Test infrastructure "**Tapper**" -- basic principles:
  - Zero overhead to write tests and report results
  - **Flexible evaluation:** easy web GUI + scriptable API
  - Optional but advanced automation

```
<table>
<thead>
<tr>
<th>Automation Framework</th>
<th>Test Suites</th>
<th>Reports Framework</th>
</tr>
</thead>
</table>
```

TAP
**BASIC PRINCIPLES**

- Test infrastructure “**Tapper**” -- basic principles:
  - Zero overhead to write tests and report results
  - Flexible evaluation: easy web GUI + scriptable API
  - Optional but advanced automation
BASIC PRINCIPLES | Zero overhead

- Zero overhead to write tests
  - Just respect the test protocol (“TAP”)

    1..3
    ok - feature 'foo' available
    ok - expected return value
    not ok - memory cleaned up

- Zero overhead to report results
  - “Fire & forget” into socket

    test_program.sh | netcat tapper 7357
BASIC PRINCIPLES | Advanced automation

- Boot machines from network and set up from scratch
  - Optionally reuse machines via SSH
- Track serial console output
- Hardware reset on epic fails
- Virtualization setups
- Complex timeout handling
- Optimize machine utilisation
  - Bandwidth-driven multiplexing of “too many” use-cases on “not enough” machines
- Allow multi-machine scenarios
  - Network performance, guest migration
- Benchmarking infrastructure
- Scriptable query interface
- Web interface
The central idea is not a technology but a protocol.
The central idea is not a technology but a protocol.

(a standard one with already existing technology)
TAP INTRO | Philosophy

- No obligatory API how to write test
- But standard protocol to declare test results

“Test Anything Protocol” (TAP)
- Easy to generate
  - OS testing - be prepared to have nothing
  - Still easy when printf/printk/echo is everything you have
  - However, lots of toolchains optionally available
- Scales from simplicity to complexity
  - http://testanything.org
TAP INTRO / Basics

- Line-based
- Starts with a plan ("1..3") – how many test lines expected
- Some “ok” test lines
- Some “not ok” test lines
- Directives “# TODO” / “# SKIP” on test lines
- Comment lines starting with “#”
- Unrecognized lines are ignored
TAP INTRO | Synopsis

1..3
ok
ok
not ok

- Plan and ok/not ok lines
TAP INTRO | Synopsis

1..3

ok - established connection
ok - checksum
not ok - transfer completed

- Plan and ok/not ok lines
- Test line descriptions
1..3

ok - established connection
ok - checksum
not ok - transfer completed
# got error message "Bummer!"

- Plan and ok/not ok lines
- Test line descriptions
- Comment lines
TAP INTRO | Synopsis

1..3

ok - established connection
ok - checksum
not ok - transfer completed # TODO we know it fails
# got error message "Bummer!"

- Plan and ok/not ok lines
- Test line descriptions
- Comment lines
- Directives # TODO
1..3

ok - established connection
ok - checksum  # SKIP no md5sum available
not ok - transfer completed  # TODO we know it fails
# got error message "Bummer!"

- Plan and ok/not ok lines
- Test line descriptions
- Comment lines
- Directives # TODO / # SKIP
1..3

ok - established connection
ok - checksum # SKIP no md5sum available
not ok - transfer completed # TODO we know it fails

# got error message "Bummer!"
Hello? I am a statement lost in code, help me out!

- Plan and ok/not ok lines
- Test line descriptions
- Comment lines
- Directives # TODO / # SKIP
- Unrecognized lines are ignored
1..3

ok 1 - established connection
ok 2 - checksum # SKIP no md5sum available
not ok 3 - transfer completed # TODO we know it fails

# got error message "Bummer!"

Hello? I am a statement lost in code, help me out!

- Plan and ok/not ok lines – optionally numbered
- Test line descriptions
- Comment lines
- Directives # TODO / # SKIP
- Unrecognized lines are ignored
1..3

**ok 1** - established connection

**ok 2** - checksum # SKIP no md5sum available

**not ok 3** - transfer completed # TODO we know it fails

# got error message "Bummer!"

Hello? I am a statement lost in code, help me out!
TAP INTRO | Embedded data

1..4

ok 1 - established connection
ok 2 - checksum # SKIP no md5sum available
not ok 3 - transfer completed # TODO we know it fails
# got error message "Bummer!"
Hello? I am a statement lost in code, help me out!
ok - transfer benchmarks
1..4

ok 1 - established connection

ok 2 - checksum # SKIP no md5sum available

not ok 3 - transfer completed # TODO we know it fails
# got error message "Bummer!"

Hello? I am a statement lost in code, help me out!

ok - transfer benchmarks

---

benchmarks:

    pass1: 1234.56
    pass2: 999.99

...
TAP INTRO | Transport Tapper meta-information
ok 1 - established connection
ok 2 - checksum # SKIP no md5sum available
not ok 3 - transfer completed # TODO we know it fails
# got error message "Bummer!"
Hello? I am a statement lost in code, help me out!

ok - transfer benchmarks

---

benchmarks:
  pass1: 1234.56
  pass2: 999.99
...
1..4

# Tapper-Suite-Name: hello-world

ok 1 - established connection
ok 2 - checksum # SKIP no md5sum available
not ok 3 - transfer completed # TODO we know it fails

# got error message "Bummer!"
Hello? I am a statement lost in code, help me out!

ok - transfer benchmarks

---

benchmarks:
  pass1: 1234.56
  pass2: 999.99
...

TAP INTRO | Transport Tapper meta-information
TAP INTRO | Transport Tapper meta-information

1..4

# Tapper-Suite-Name: hello-world
# Tapper-Reportgroup-Testrun: 244122

ok 1 - established connection

ok 2 - checksum # SKIP no md5sum available

not ok 3 - transfer completed # TODO we know it fails

# got error message "Bummer!"

Hello? I am a statement lost in code, help me out!

ok - transfer benchmarks

---

benchmarks:
  pass1: 1234.56
  pass2: 999.99
  ...

TAP INTRO / Run and evaluate

- Developer, locally

```bash
$ prove my_feature.sh  # run + evaluate
```
Developer, locally

$ prove my_feature.sh  # run + evaluate
my_feature.sh .. ok
All tests successful.
Files=1, Tests=1,  3 wallclock secs ( ... )
Result: PASS
### TAP INTRO | Run and evaluate

- **Developer, locally**

  ```
  $ prove my_feature.sh                    # run + evaluate
  my_feature.sh .. ok
  All tests successful.
  Files=1, Tests=1, 3 wallclock secs ( ... )
  Result: PASS
  ```

- **Inside Tapper**

  ```
  $ prove -e cat static_tap_results.tap   # just evaluate
  $ prove --formatter=TAP::Formatter::HTML # render to HTML
  ```
Everyone who can do TAP can participate.
Now put technology around it.
Testing

Automation Framework → Test Suites → Reports Framework

TAP
TAP SUPPORT | Re-using and writing tests

- Using tests: `autotest`
- Writing tests: `Tapper-autoreport`
**TAP SUPPORT | Using tests - autotest**

- **Autotest** is a test project targeting the Linux kernel
- Wraps lots of existing test and benchmark suites
- AMD contributed TAP support as of autotest v0.13
  - Convert test results and data to TAP + embedded YAML
  - Bundle everything in TAP::Archive (.tar.gz of TAP + meta data)

  ```
  autotest --tap tests/hackbench/control
  ```

- Generic wrapper Tapper::TestSuite::AutoTest
  - Downloads + installs autotest client from github or other URL
  - Run and upload TAP to Tapper server

  ```
  tapper-testsuite-autotest --test hackbench
  ```

  - [https://github.com/amd/Tapper-TestSuite-AutoTest](https://github.com/amd/Tapper-TestSuite-AutoTest)
**TAP SUPPORT | Writing tests - tapper-autoreport (2)**

- **Tapper-autoreport**
  - A shell “include” (*source*) file
    - You do what you normally do to test from shell
  - Will make your script magically behave like a Tapper testsuite
    - Sends a TAP report to Tapper server
    - Includes meta-information
    - Uploads files
    - No send+upload when run via “prove” for local test development
- Lots of ways to influence behaviour
  - “Do What I Mean” parameters
  - Environment variables
  - Uses Tapper automation environment (e.g., Testrun-ID)
- [https://github.com/amd/Tapper-autoreport](https://github.com/amd/Tapper-autoreport)
SYNOPSIS – shortest usage

```bash
#!/bin/bash

# your testing here
. tapper-autoreport $? /tmp/my.log /tmp/results.dat
```

“Do What I Mean” params

- ` $? ` ... integers are interpreted as success/fails – useful for one-liners
- Existing filenames are attachments to be uploaded
SYNOPSIS – shortest usage

```bash
#!/bin/bash

# your testing here
.tapper-autoreport $? /tmp/my.log /tmp/results.dat
```

“Do What I Mean” params
- `$?` ... integers are interpreted as success/fails – useful for one-liners
- Existing filenames are attachments to be uploaded
SYNOPSIS – shortest usage

```bash
#!/bin/bash

# your testing here
. tapper-autoreport $? /tmp/my.log /tmp/results.dat
```

“Do What I Mean” params

- `$?` ... integers are interpreted as success/fails – useful for one-liners
- **Existing filenames are attachments to be uploaded**
SYNOPSIS – use utility functions

```bash
#!/bin/bash
.tapper-autoreport --import-utils
# your testing here
.tapper-autoreport $? /tmp/my.log /tmp/results.dat
```

Utility functions like

- `ok $? “some description”`
- `negate_ok $? “some description”`
- `require_cpu_feature “cpb”`
- `require_family_range 0x12 0x15`
- `has_kernel_config CONFIG_SENSORS_FAM15H_POWER`
“Vendor ID” – the issue

- Some data structure overflowed into the vendor ID in `/proc/cpuinfo`
- Sloppily check we are on AMD and skip all if not
- Check whether the full vendor string is correct
“Vendor ID” – the test

```bash
#!/bin/bash # vendor-id.sh

tapper-autoreport --import-utils

TICKETURL='https://osrc/bugs/show_bug.cgi?id=901'

require_vendor_amd
grep -q 'vendor.*AuthenticAMD' /proc/cpuinfo
ok $? “vendor string in /proc/cpuinfo”

tapper-autoreport
```
“Vendor ID” – the report

```
1..6
# Tapper-suite-name: vendor-id
# Tapper-machine-name: bascha
# Tapper-ticket-url: https://osrc/bugs/show_bug.cgi?id=901
# Tapper-uname: Linux bascha 2.6.35 #59-Ubuntu SMP Tue Aug 30 19:00:03 UTC 2011 x86_64 GNU/Linux
# Tapper-osname: Ubuntu 10.10
# Tapper-kernel: 2.6.35
# Tapper-changeset: Linux version 2.6.35-30-generic (buildd@allspice) (gcc version 4.4.5 (Ubuntu... root=UUID=6990cb5e-1a77-40b8-ba05-919f6c928607 ro quiet splash
# Tapper-flags: 2 cores [AMD Athlon(tm) 64 X2 Dual Core Processor 6000+]
# Tapper-ram: 2007
# Tapper-starttime-test-program: Tue, 11 Oct 2011 14:48:04 +0200
ok - autoreport
ok - exitcode
   ---
   exitcode: 0
   ...
ok - success
ok - require_vendor_amd
ok - vendor string in /proc/cpuinfo
# File upload: '/boot/config-2.6.35-30-generic'
# File upload: 'vendor-id.sh'
# File upload: '/proc/cpuinfo'
# File upload: '/proc/devices'
# File upload: '/proc/version'
```
Zero overhead to submit test results.
REPORT RESULTS | “Fire & forget”

- Report via “fire & forget” into socket

```
$ test_program.sh  | netcat  tapper  7357
```

- Easy
- On crashes you get as much as possible
- Still recognize the crash (planned vs. counted test lines)
AUTOMATION

Automation Framework → Test Suites → Reports Framework

TAP
Optional

- Set up machines from scratch, over network
  - Unpack prepared images (.iso, .tgz)
  - Or run kickstart/autoyast/d-i distro installers
  - Inject any other requirements

- Allow virtualization setups
  - Xen, KVM
  - Inject into guests

- Optionall via SSH
  - Already prepared machines
  - E.g., simnow, just inject kernel
AUTOMATION | Overview (2)

- Track serial console output, cover early boot problems
- Hardware reset on epic fails
- Time-out handling
  - Virtualization-aware
- Suspend/Resume support
- Benchmarking infrastructure
AUTOMATION | Overview (3)

- Advanced scheduling
  - Optimize machine utilisation
    - “Too many” use-cases on “not enough” machines
    - Use-case queues with bandwidths/priorities
    - Different types of bandwidths (“official” vs. “non-official” to fill under-used machines)
    - Choose host by complex feature expressions
      - mem >= 4096 and vendor eq “AMD”
  - Bind hosts to queues
  - Allow multi-machine scenarios
    - Network performance
    - Guest migration
AUTOMATION | Use-case bandwidths
AUTOMATION | Synopsis (1)

SYNOPSIS

$ tapper-testrun newhost --name grizzly --active --queue simnow

$ tapper-testrun listhost --verbose

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Active</th>
<th>Testrun ID</th>
<th>Comment</th>
<th>Queues</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>blibb</td>
<td>active</td>
<td>249876</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>blobb</td>
<td>active</td>
<td>249862</td>
<td></td>
<td>xen-unstable-pvops-64</td>
</tr>
<tr>
<td>27</td>
<td>blubb</td>
<td>active</td>
<td>249529</td>
<td>testplan experimenting</td>
<td>AdHoc</td>
</tr>
<tr>
<td>22</td>
<td>grizzly</td>
<td>active</td>
<td>free</td>
<td></td>
<td>simnow</td>
</tr>
</tbody>
</table>

$ tapper-testrun listqueue

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>AdHoc</td>
<td>1000</td>
</tr>
<tr>
<td>98</td>
<td>autoinstall-bare-rhel-6.2-64</td>
<td>200</td>
</tr>
<tr>
<td>71</td>
<td>autoinstall-bare-sles-11.2-64</td>
<td>200</td>
</tr>
<tr>
<td>21</td>
<td>xen-4.0-testing-32</td>
<td>50</td>
</tr>
<tr>
<td>22</td>
<td>xen-4.0-testing-64</td>
<td>50</td>
</tr>
<tr>
<td>75</td>
<td>xen-4.1-testing-pvops-32</td>
<td>300</td>
</tr>
<tr>
<td>76</td>
<td>xen-4.1-testing-pvops-64</td>
<td>300</td>
</tr>
</tbody>
</table>
AUTOMATION | Synopsis (2)

**SYNOPSIS**

```
$ tapper-testrun listqueue --name AdHoc

Id: 10
Name: AdHoc
Priority: 1000
Active: yes
Bound hosts: blubb, affe, zomtec
Queued testruns (ids): 238772, 238773, 238774, 238785, 238786, 238787

$ tapper-testrun list --id 249532

id: 249532
topic: track-workload-stress-opensuse_11.4_32
state: schedule
queue: AdHoc
requested hosts: blubb
auto rerun: no
precondition_ids: 224057, 224058, 224059, 224060, 224061
```
$ tapper-testrun freehost --name grizzly --desc "known Xen hang, don’t wait for timeout"

$ tapper-testrun newtestplan -v \ 
  --file topic/osrc/kernel/track-workload/track-workload_autoinstall \ 
  -Ddistros=rhel_6.1_64,sles_11.2_32 \ 
  -Dtests=hackbench,dbench

Plan created
id: 241
url: http://tapper/tapper/testplan/id/241
path: topic/osrc/kernel/track-workload/track-workload_autoinstall
file: /data/tapper/live/testplan/topic/osrc/kernel/track-workload/track-workload_autoinstall
### Automated test runs of last 2 days

This list shows automated test runs. Links in columns C show details of single report; in column Data show all reports of this suite; in column Machines show all reports on this machine. If you look for more filtered results not limited to the automated systems, try [Reports](#).

**Fri Oct 14, 2011**

<table>
<thead>
<tr>
<th>ID</th>
<th>Date/Time (GMT)</th>
<th>Topic</th>
<th>Machine</th>
<th>State</th>
<th>Ratio</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>12200144</td>
<td>2011-10-14 15:02</td>
<td>iso</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200144</td>
<td>2011-10-14 15:04</td>
<td>track-workload-backbench-disk1_1.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200145</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_1.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200146</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_2.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200147</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_3.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200148</td>
<td>2011-10-14 15:04</td>
<td>i686</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200149</td>
<td>2011-10-14 15:04</td>
<td>track-workload-backbench-disk1_1.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200150</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_1.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200151</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_2.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200152</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_3.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200153</td>
<td>2011-10-14 15:04</td>
<td>Debug</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200154</td>
<td>2011-10-14 15:04</td>
<td>track-workload-backbench-disk1_1.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200155</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_1.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200156</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_2.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200157</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_3.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200158</td>
<td>2011-10-14 15:04</td>
<td>i686</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200159</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_1.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200160</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_2.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200161</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_3.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200162</td>
<td>2011-10-14 15:04</td>
<td>track-workload-backbench-disk1_1.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200163</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_1.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200164</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_2.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200165</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_3.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200166</td>
<td>2011-10-14 15:04</td>
<td>Benchmark</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200167</td>
<td>2011-10-14 15:04</td>
<td>track-workload-backbench-disk1_1.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200168</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_1.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200169</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_2.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200170</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_3.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200171</td>
<td>2011-10-14 15:04</td>
<td>Track</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200172</td>
<td>2011-10-14 15:04</td>
<td>track-workload-backbench-disk1_1.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200173</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_1.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200174</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_2.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200175</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_3.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200176</td>
<td>2011-10-14 15:04</td>
<td>track-workload-backbench-disk1_1.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200177</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_1.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200178</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_2.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200179</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_3.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200180</td>
<td>2011-10-14 15:04</td>
<td>Benchmark</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200181</td>
<td>2011-10-14 15:04</td>
<td>track-workload-backbench-disk1_1.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200182</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_1.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200183</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_2.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200184</td>
<td>2011-10-14 15:04</td>
<td>track-workload-stress-disk1_3.84</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>12200185</td>
<td>2011-10-14 15:04</td>
<td>Benchmark</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>
Here be dragons.
"Preconditions"
- YAML files describing the machine setup
- Automatically producible
- Human-readable
- Verifiable
- Tweakable
  - "Do The Right Thing" internally, like always handle root image first

We auto-generate them from a database
- Test matrix of host/guest/workload/config combinations
** AUTOMATION CONTROL | Synopsis **

**precondition_type**: virt
name: automatically generated KVM test

**host**: 
  root:
    **precondition_type**: autoinstall
    grub_text: "timeout 2\n\ntitle RedHat Testing\n\nkernel […] ks=[…] $TAPPER_OPTIONS […]"
    name: autoinstall-kvm-fedora-14
    timeout: 10000

**preconditions**: 
  - **precondition_type**: package
    filename: xen-pvops-d7e0e9f3.tar.gz

**testprogram_list**: 
  - execname: /opt/tapper/bin/tapper-testsuite-ctcs
    timeout_testprogram: 300

**guests**: 
  - root:
    **precondition_type**: image
    ...
  - root:
    **precondition_type**: image
    ...
Different types
- “Copy a file”, “unpack a package”
- “Base OS from image file”, “base OS from kickstart/autoyast”
- “Virtualization environment”
- “Execute test program”
- Lazy preconditions (“producer” plugins)
  - “Use latest Xen package file by the time of scheduling”
  - Combine that with “auto-rerun”

Different granularity for different needs
- Single preconditions
- Macro-preconditions (+ precompile pass with template language)
- Testplans (+ multiple machines, several testruns)
Testplan developer vs. end user

Internal power vs. external easiness

The testplan developer combines **complexity** to provide **simplicity** to the end user
  - One single front-end file for the use-case
  - Optional parameters
  - Sensible defaults
  - Self-documentation

```bash
$ tapper-testrun newtestplan --verbose \
   --file track-workload_autoinstall \
   -Ddistros=rhel_6.1_64,sles_11.2_32 \
   -Dtests=hackbench,dbench
```
The use-case:
- Schedule matrix of workloads over distros and machines
- Track perf counters to investigate the workload
- Upload perf logs
- Organize results to ease later evaluation
# tapper-description: Track performance counters over a workload
# tapper-mandatory-fields:
# tapper-optional-fields: tests, distros, machine

[%- PROCESS 'osrc/includes' -%]

[%- IF tests == '' %][%- tests = 'hackbench' %][%- END -%]

[%- IF distros == '' %][%- distros = 'sles_11.2_64' %][%- END -%]

[%- IF machine == '' %][%- machine = 'grizzly' %][%- END -%]

[%- IF title == '' %][%- title = BLOCK %]

[%- IF (tests.match(',')) %]MULTI

[%- ELSE %][%- tests %]

[%- END %]

[%- IF (distros.match(',')) %]MULTI

[%- ELSE %][%- distros %]

[%- END %][%- END %][%- END -%]

[%- AllTests = tests.split(',') %]
[%- AllDistros = distros.split(',' %]
### Track performance counters over several workloads.

### Name: track-workload-[% title %]

### Optional params:

- **Dtests=<testname>** Workload names, comma separated; default: hackbench
- **Ddistros=<distro>** Distro names, comma separated; default: sles_11.2_64
- **Dmachine=<machine>** Machine name; default: grizzly

### Available values:

- **distros**: [% FOREACH d = distro_list -%][% d %], [% END %]
- **tests**: [% FOREACH t = useful_autotest_tests -%][% t %], [% END %]
AUTOMATION CONTROL | Testplan 3/8 – open loops (distros + tests)

[%- FOREACH distro = AllDistros %]
[%- FOREACH test = AllTests %]
[% testrunsuffix = BLOCK %][% test %]-[% distro %][% END %]
[% Timeout = Timeout+10800 -%]
[\% IF distro == 'sles_11.2_64' \%]
[\% install_file = 'autoyast=http://tapper/autoinstall/sles/11.2/x86_64/bare.xml' \%]
[\% install_repo = 'install=ftp://osko/testing/sles/11.2/x86_64' \%]
[\% install_opts = 'textmode=1' \%]
[\% kernel = '/tftpboot/testing/sles/11.2/x86_64/linux' \%]
[\% initrd = '/tftpboot/testing/sles/11.2/x86_64/initrd' \%]
[\% END \%]

[\% IF distro == 'rhel_6.1_64' \%]
[\% install_file = 'ks=http://tapper/autoinstall/rhel/6.1/x86_64/bare.ks' \%]
[\% install_repo = 'repo=ftp://osko/rhel/6.1/x86_64/os' \%]
[\% install_opts = 'ksdevice=link' \%]
[\% kernel = '/tftpboot/stable/rhel/6.1/x86_64/vmlinuz' \%]
[\% initrd = '/tftpboot/stable/rhel/6.1/x86_64/initrd.img' \%]
[\% END \%]
type: multitest

description:
  topic: track-workload-% testrunsuffix %
  requested_hosts_all:
  - [% machine %]

preconditions:
precondition_type: autoinstall
name: autoinstall-%distro%
grub_text: "timeout 2\n\ntitle [%distro%] Testing\nkernel [%kernel%] \n [install_file%] \n [install_repo%] \n [install_opts%] \n console=ttyS0,115200 $TAPPER_OPTIONS\ninitrd [%initrd%]\n"

precondition_type: copyfile
protocol: local
name: /data/tapper/testprograms/track-workload/*
dest: /
precondition_type: testprogram
program: /track-workload-pmc.sh

precondition_type: testprogram
program: /opt/tapper/bin/tapper-testsuite-autotest
parameters:
  - --source_url
    - file://data/tapper/packages/autotest/osrc-autotest-snapshot.tar.gz
  - --test
    - [% test %]
timeout: [% Timeout %]

precondition_type: testprogram
program: /track-workload-upload-results.sh
AUTOMATION CONTROL | Testplan 8/8 – close loops

[% END %][%# FOREACH AllTests %]
[% END %][%# FOREACH AllDistros %]
$ tapper-testrun newtestplan --guide --file track-workload_autoinstall
$ tapper-testrun newtestplan --guide --file track-workload_autoinstall

Track performance counters over several workloads.

Name : track-workload-hackbench-sles_11.2_64

Optional params:

-Dtests=<testname> Workload names, comma separated; default: hackbench
-Ddistros=<distro> Distro names, comma separated; default: sles_11.2_64
-Dmachine=<machine> Machine name; default: grizzly

Available values:

distros: rhel_6.1_64, sles_11.2_64, ...
tests: hackbench, dbench, tiobench, ...
AUTOMATION CONTROL | Testplan execution

$ tapper-testrun newtestplan --verbose \ 
   --file track-workload_autoinstall \ 
   -Ddistros=rhel_6.1_64,sles_11.2_32 \ 
   -Dtests=hackbench,dbench
$ tapper-testrun newtestplan --verbose \
   --file track-workload_autoinstall \
   -Ddistros=rhel_6.1_64,sles_11.2_32 \
   -Dtests=hackbench,dbench

Plan created
   id:   241
   url:  http://tapper/tapper/testplan/id/241
   path: topic/osrc/kernel/track-workload/track-workload_autoinstall
   file: /data/tapper/[…]topic/osrc/kernel/[…]track-workload_autoinstall
## AUTOMATION CONTROL | Screenshots (5)

![Screenshot of Tapper interface](image-url)

### Report Details

**Report ID:** 309790  
**Title:** Topic-track-workload-stress-rhel_6.1_64  
**Date:** October 28, 2011  
**Owner:** [Name]  
**Suite:** Tapper  
**Environment:** Linux

### Test Execution Context

- **System:** 128 GB RAM, 8-core AMD Opteron 6388 processor  
- **CPU:** 8 cores [AMD Engineering Sample]  
- **OS:** Linux argon 2.6.32-13.1.0-13.6.16-86.94 #1 SMP Mon May 10 13:54:10 EST 2011 x86_64 GNU/Linux

### Test Plan

- **Name:** Test Plan  
- **Description:** Test plan details  
- **Start:** October 28, 2011
AUTOMATION CONTROL | Screenshots (7)
EVALUATE RESULTS
RESULTS

Automation Framework → Test Suites → Reports Framework

TAP
EVALUATE RESULTS | What do we have so far?

- Remember: we dropped TAP into Tapper with “fire & forget” (*netcat*)
  - Hide internal complexity
    - Actual success status
    - Aggregated results
    - Report groups
    - Meta-information
    - Embedded YAML data
    - Any sufficiently advanced technology
      - TAP::Parser
      - TAP::DOM
      - TAP::Formatter::HTML
      - Databases
      - Etc.

- How to trivially access results?
EVALUATE RESULTS | The no-problem

- Web application for “end users”
  - RED / YELLOW / GREEN
  - Cautious but useful Javascript
  - Overviews, details, attachments
  - List, filters, RSS feeds
EVALUATION | Query API
QUERY API | The query gap

- Scriptable querying
  - The same ease as reporting
  - Again: shell level, `netcat`

- Use-cases
  - Generally access our own reports
    - Data + attachments
  - Track test success over time
  - Track benchmark results
  - Custom-visualize the data

- Challenges
  - Test suites change over time → fuzzy find
  - Hide the toolchain
QUERY API | The solution

- Provide template mechanism
- With embedded query language “DPath”
- Dialog-oriented protocol
  - HERE-doc style
  - \( \rightarrow \) Send template with “netcat”
  - \( \leftarrow \) Receive processed content
QUERY API | Example 1 – Get simple values

- Command

```
$ cat report.mas | netcat tapper 7358 > result.txt
```

- Template

```
#! tt <<EOTEMPLATE
Planned tests:
[% FOREACH x IN reportdata('{ "suite.name" => "power_msr" } :: //tap/tests_planned') -%]
    [% x %]
[% END %]
EOTEMPLATE
```

- Result

```
Planned tests:
  3
  4
  17
```
QUERY API | Example 2 – Fill a GNUPLOT file

- Command
  
  ```
  $ cat CTCS_ratio.gnuplot | netcat tapper 7358 | gnuplot
  ```

- Template
  
  ```
  #! tt <<EOTEMPLATE
  TITLE = "success ratio: CTCS"
  set output "CTCS_ratio.png"
  plot '-' using 0:2 with linespoints
  [% time = reportdata('...') %]
  [% ratio = reportdata('...') %]
  [% FOREACH i IN ... %]
  [  [% time.$i %] [% ratio.$i %]
  [% END %]
  EOTEMPLATE
  ```

- Result
  
  - Generated file "CTCS_ratio.png"
QUERY API | Example 2 – Fill a GNUPlot file

- Command
  ```
  $ cat CTCS_ratio.gnuplot | netcat tapper 7358 | gnuplot
  ```

- Template
  ```
  #! tt <<EOTEMPLATE
  TITLE = "success ratio: CTCS"
  set output "CTCS_ratio.png"
  plot '-' using 0:2 with linespoints
  [% time = reportdata('...') %]
  [% ratio = reportdata('...') %]
  [% FOREACH i IN ... %]
  [% time.$i %] [% ratio.$i %]
  [% END %]
  EOTEMPLATE
  ```

- Result
  - Generated file "CTCS_ratio.png"
Did you notice? No client-side toolchain dependencies!
QUERY API | How does it work?

- TAP::DOM
  - A data structure (DOM) out of TAP
 QUERY API | How does it work?

- **TAP::DOM**
  - A data structure (DOM) out of TAP

```ruby
{ 'tests_planned' => 6,
  'tests_run' => 8,
  # [...]
  'summary' => {
    'status' => 'FAIL',
    'total' => 8,
    'passed' => 6,
    'failed' => 2,
    'skipped' => 1,
    'todo' => 4,
    'todo_passed' => 2,
    # [...]
  },
  'lines' => [{
    'number' => '1',
    'is_ok' => 1,
    'description' => '- connection established',
    '_children' => [
      { 'is_yaml' => 1,
        'data' => [{'pass1' => '1234.56',
                     'pass2' => '999.99'}] } ]
  }]
}
```
QUERY API | How does it work?

- **TAP::DOM**
  - A data structure (DOM) out of TAP
- **DPath to fuzzy navigate data**
  - XPath-like

```javascript
{
   'tests_planned' => 6,
   'tests_run'    => 8,
   # [...]
   'summary' => {
      'status' => 'FAIL',
      'total'  => 8,
      'passed' => 6,
      'failed' => 2,
      'skipped' => 1,
      'todo'   => 4,
      'todo_passed' => 2,
      # [...]
   },
   'lines' => [
      {
         'number' => '1',
         'is_ok'  => 1,
         'description' => '- connection established',
         '_children' => [
            {
               'is_yaml' => 1,
               'data' => [
                  {
                     'pass1' => '1234.56',
                     'pass2' => '999.99'
                  }
               ]
            }
         ]
      }
   ]
}
```
QUERY API | How does it work?

- **TAP::DOM**
  - A data structure (DOM) out of TAP
- **DPath to fuzzy navigate data**
  - XPath-like

```
{ 'tests_planned' => 6,
  'tests_run' => 8,
  # [...]
  'summary' => {
    'status' => 'FAIL',
    'total' => 8,
    'passed' => 6,
    'failed' => 2,
    'skipped' => 1,
    'todo' => 4,
    'todo_passed' => 2,
    # [...]
  },
  'lines' => [
    { 'number' => '1',
      'is_ok' => 1,
      'description' => '- connection established',
      '_children' => [
        { 'is_yaml' => 1,
          'data' => [ { 'pass1' => '1234.56', 'pass2' => '999.99' } ] }
      ]
  }
}
```
QUERY API | How does it work?

- TAP::DOM
  - A data structure (DOM) out of TAP
- DPath to fuzzy navigate data
  - XPath-like

```
''/tests_planned''
''//tests_planned''
```

```json
{
  'tests_planned' => 6,
  'tests_run' => 8,
  # [...]  
  'summary' => {
    'status' => 'FAIL',
    'total' => 8,
    'passed' => 6,
    'failed' => 2,
    'skipped' => 1,
    'todo' => 4,
    'todo_passed' => 2,
    # [...]  
  },
  'lines' => [
    {
      'number' => '1',
      'is_ok' => 1,
      'description' => '- connection established',
      '_children' => [
        {'is_yaml' => 1,
        'data' => ['pass1' => '1234.56',
                   'pass2' => '999.99']}
      # [... lines ...]
    } ] }
```
QUERY API | How does it work?

- **TAP::DOM**
  - A data structure (DOM) out of TAP

- **DPath to fuzzy navigate data**
  - XPath-like

```json
{
    'tests_planned' => 6
    'tests_run'    => 8,
    # [...]
    'summary' => {
        'status' => 'FAIL',
        'total'  => 8,
        'passed' => 6,
        'failed' => 2,
        'skipped' => 1,
        'todo'   => 4,
        'todo_passed' => 2,
        # [...]}
},
'lines' => [{
    'number' => '1',
    'is_ok'  => 1,
    'description' => '- connection established',
    '_children' => [{
        'is_yaml' => 1,
        'data' => [{'pass1' => '1234.56',
                    'pass2' => '999.99'}]
    },
    # [... lines ...]
}]
```
QUERY API | How does it work?

- **TAP::DOM**
  - A data structure (DOM) out of TAP

- **DPath to fuzzy navigate data**
  - XPath-like

```json
{
    'tests_planned' => 6,
    'tests_run'    => 8,
    # [...]  
    'summary' => {
        'status'  => 'FAIL',
        'total'   => 8,
        'passed'  => 6,
        'failed'  => 2,
        'skipped' => 1,
        'todo'    => 4,
        'todo_passed' => 2,
        # [...]  
    },
    'lines' => [
        {
            'number'  => '1',
            'is_ok'   => 1,
            'description'  => '- connection established',
            '_children' => [
                {
                    'is_yaml' => 1,
                    'data'    => ['pass1' => '1234.56', 'pass2' => '999.99']
                }
            ]
        }
    ]
}
```
QUERY API | How does it work?

- **TAP::DOM**
  - A data structure (DOM) out of TAP
- **DPath to fuzzy navigate data**
  - XPath-like

```json
{ 'tests_planned' => 6,
  'tests_run' => 8,
  # [...] 'summary' => {
    'status' => 'FAIL',
    'total' => 8,
    'passed' => 6,
    'failed' => 2,
    'skipped' => 1,
    'todo' => 4,
    'todo_passed' => 2,
    # [...] } },
'lines' => {
  number => '1',
  'is_ok' => 1,
  'description' => '- connection established',
  'children' => [ # subsequent comments/yaml
    { 'is_yaml' => 1,
      'data' => [ { 'pass1' => '1234.56',
                    'pass2' => '999.99' } ] } ]
  # [... lines ...]
} }
```
QUERY API | Anatomy of a Tapper::DPath

{ suite_name => "CTCS" } :: //tests_planned[value > 10]/../summary/passed

- Virtual DOM of the TAP database
- Two orthogonal concepts
  - Database axis: provide but hide relational access
    - SQL::Abstract
    - The “history of reports”
  - Report axis: inside single reports data structure
    - TAP::DOM
    - Data::DPath
    - Inside “one point in history”
MORE | Topics

- Benchmark sub-infrastructure
  - Subscribe to incoming data with DPaths (//data//codespeed/*)
  - Pass-through to benchmark rendering application → Codespeed

- Integration with TaskJuggler
  - Map task IDs to filesystem hierarchy of testplan files
    - osrc.productfoo.xen.power_msr.xen4_3 → osrc/productfoo/xen/power_msr/xen4_3
  - Schedule testruns by taskjuggler task dates
  - Report back results per E-Mail as TaskJuggler timesheets
  - TaskJuggler renders project status from that

- Deployment
  - Bootstrap your own infrastructure
  - Create utility images, client packages
SUMMARY
SUMMARY / Framework

- Complete testing environment fitting several parties’ needs
  - Test team
    - Automation to run machine pool
  - Developer / Tester
    - Support on developing and running tests
    - Locally and/or automated
  - Manager / Tester
    - Visual presentation of test results
    - QA lifecycle, driven by planning software
- Built on top of open source standards
- Provide complexity – allow simplicity

- The End – Thank You!
Trademark Attribution

AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names used in this presentation are for identification purposes only and may be trademarks of their respective owners.

©2011 Advanced Micro Devices, Inc. All rights reserved.
http://amd64.org/mailman/listinfo/tapper
irc.freenode.net / #tapper
github.com/amd