caliper-nxt

(Proof of Concept)

Introduction

Caliper is a generic benchmarking framework with a set of tools, and their respective output parsers, that generates relative score of these tests, for platforms under which tests are executed.

Currently caliper is used to measure relative performances a hardware server platform relative to some of ARM and Intel based server platforms.

Background

To remove calipers framework related functionality and focus on calipers core features of Parsing and Scoring of multiple server focused tests, many frameworks are analysed.

Comparison of frameworks

Below chart shows the initial comparison table, for features that was available at the time.

Features Comparision	Caliper	Avocado	WA	Phoronix	PerfKit
URL for source code	https://github.com/open-estuary/caliper/	https://github.com/avocado-framework	https://github.com/ARM-software/workload-automation	https://github.com/phoronix-test-suite/phoronix-test-suite/	https://github.com/GoogleCloudPlatform/PerfKitBenchmarker
Domain of test	Data Center	Generic	Android	Generic	Generic/Cloud
Targets supported	Intel, ARM64	Intel, ARM64?	Intel?, ARM64	i686, x86_64, ARM, PowerPC	
Active project?	Yes	Yes	Yes	Yes	Yes
Last release	V3.0-rc0, 07-Jan-2017	39.0, 26-Jul-2016	V2.6.0, 23-Dec-2016	V6.8.0, 6-Dec-2016	V1.10.0, 7-Jan-2017
Last commit	7-Jan-2017	4-Jan-2017	23-Dec-2016	8-Jan-2017	6-Jan-2017
Number of active contributors	7	26	23	19	60

Test trigger configuration(date, everyday time, loop etc)	Yes	No	No	No	No
Build Configurations support	No, Fixed build configs	No	No	No	No
Run Configurations support	Yes	Yes	Yes, using Agenda tunables	Yes, XML & Bash	Yes
Orginal Test Log availability	Yes	Yes	Yes?	Yes, web-based results viewer or uploaded	Yes
Framework run log availability	Yes	Yes	Yes	Yes	Yes
Report Score	Yes	No	No	No	No, Score is identified to be implemented
Plug in support	No	Yes	Yes, using Agenda	Yes, using Modules	Yes
XLS report	Yes	Yes	Yes (CSV)	Yes (CSV and more)	?
CI/Jenkins Integration	Yes	Yes	No	No	No
Parallel execution	Yes, Limited support	Yes	?	?	Yes
Multi-Node test support	No	No	No	No	Yes, multiple VM configurations supported
Framework Dev Language	Python 2.7	Python 2.7	Python 2.7	PHP-5 .3+	Python 2.7
Debugging and Monitoring Support	No	Yes, GDB	?	Yes, test-module option	Yes, progress logs
Custom instrumentation	No	Yes	Yes	Yes	Yes
Explicitly enabling or disabling tests	Yes	Yes	Yes	Yes	Yes
Categorizing tests	No, Category is fixed	Yes	No?	No	No
Environment Variables for Simple Tests	No	Yes	No	No	No
Running Remote Tests	Yes	Yes	Yes?	?	Yes, through VM's

Running VM Tests	No	Yes	No	No	Yes
Running Docker Tests	No	Yes	No	No	Yes
List workload	No	Yes	Yes	?	Yes
Show details of the work load	No	No?	Yes	?	DSTAT instrumentation
Result processing hooks	Yes, basic and fixed	Yes	Yes	?	Yes
Same test with different config	No	Yes	No	Yes	No
Resubmit specific job on failure	No	Yes	Yes	Yes	No
Compare two execution results	No	Yes	No	No	No
Expertise in the language of the framework	Yes	No	No	No, Web based	No

Advantages of Avocado

As can be seen from the table above, Avocado have slight advantage from the requirements perspective.

The features already available, features that are getting developed by avocado members, its heritage from ‘Auto-test’ test framework, opportunity to get involved in the development in open source, etc. are the advantages considered while selecting avocado.

Estimation of effort

An initial estimate of the feature required and changes needs to be done for avocado are listed below.

Features required	**Changes needed for Avocado**	**Effort**
Integration of all tools present in Caliper	Needs to write test wrapper script, identify right configurations for test parameters	Medium
Add CPU, Memory binds with numactl	Not directly supported by Avocado	Easy
Add scoring, report, observation generation	Require changes in plugin, framework code	Medium

Optional/Configurable Download, Build, Configure	Can be easily done through modifying test wrapper as an when needed. Making it configuration will require modification in avocado framework code.	Difficult, if requires framework code change
Remote execution	Requires target to be configured with Avocado tools, internet access, OS distribution (Only Ubuntu is verified, but Avocado supports CentOS/Fedora/Red hat natively)	Easy
Copying remote results	Framework supports – needs to check specific requirements are also supported.	Easy
Multiple config/param for tests	Framework support exists through multiplex with yaml files.	Easy
Instrumentation	Needs to get requirements	Difficult, if requires framework code change
CI/Jenkins support	Framework supports. Not explored.	Medium
Integration of google perfkit for server/cloud testing		Medium
Multi Node feature	Depends on resolving test report	Difficult

Installation

Steps

1. Install Avocado framework as described in the website for avocado.
2. Install caliper-nxt with command 'sudo python setup.py install'
3. Run command 'avocado plugins' and verify that 'caliper_cli' and 'caliper_post' are listed along with plugins.
4. Run caliper test and verify output file generated (details below)

Running test

avocado run <test script file> --caliper <path to config file> [--caliper_output <output folder>].

Example commands:

- Running single test

```
avocado run dhrystone.py --caliper caliper_config.txt
```

- Running multiple tests

```
avocado run hardware_info.py dhrystone.py coremark.py --caliper
/tmp/test/caliper_config.txt
```

- Running tests to output folder

```
avocado run --caliper /tmp/test/caliper_config.txt /tmp/test/hardware_info.py
--caliper-output /tmp/caliper_output /tmp/test/coremark.py
```

- Remote execution

```
avocado run
      --remote-hostname 192.168.40.56
      --remote-username test
      --remote-password test123
      --caliper /tmp/test/caliper_config.txt /tmp/test/dhrystone.py
```

- Multitarget execution
  <TBD>

Integration work details

Two plugin are added to avocado, that are,

1. Plugin (caliper_cmd) that enable caliper from avocado ‘run’ command
2. Plugin (caliper_parser) that enable caliper function that is called as post job functionality.

Caliper_cmd plugin

This plugin enable two options for the avocado run command as below.

1. '--caliper <config file>' : Specifies config file to be used for the current avocado job
2. '--caliper-output <output path>' : Specifies the location of parsed/scored outputs

Caliper_parser plugin

This plugin will be called after all the test suits are completed, as post job call. The caliper parsing and scoring feature are implemented as part of this call. Following is the steps done on this call.

1. Parse the arguments passed, config file and output folder.
2. Create output folders for the caliper parser/scorer output.
3. Copy std output for each test to the test output log folder
4. Parse test output logs using parser of each tests to generate parsed output logs.
5. Process parsed outputs and test run config files to generate final parsed yaml file.
6. Combine all yaml file and generate platform yaml file.

Implementation details

Initial approach was to use pre/post jobs feature of Avocado. Using post jobs plugins, scoring code can be run after the execution of tests.

To run Post Job, add the script to run at /etc/avocado/scripts/job/post.d

Another way to implement a pre/post job script is to as given in the examples (avocado/example/plugin/job-pre-post/sleep) we needed to define the function to be called and install this plugin with setup.py install.

But this approach has problems as these plugins are run after the completion of jobs. All pre and post job scripts will run for all test invocations. It will not get the individual jobs context. It has to wait until all jobs are completed. So this problem is not easy to handle.

Second approach was to modify the job execution to include parsing of test output. But this will make test not clean. All timings metrics includes the execution time of the scoring script too. This also make it not a proper solution.

Third approach was to using tearDown() api of the job. tearDown() was called by the frame work for each job, after SetUP() and test(). This has the benefit of running in the jobs context.

The third approach was prototyped to execute the LMbench test. Parser script from the caliper was used to get the parsed YAML file from the stdout/stderr inside tearDown() call

Modifications for plugins

Create a top level setup.py for plugins for registering to entry points as below, where entry_point1 declare the run command options, and entry_point2 defines caliper parser to be called as post job.

In setup.py

```
...
        entry_points={
                'avocado.plugins.cli': [entry_point1],
                'avocado.plugins.job.prepost': [entry_point2],
        }
```

```
    ...
```

In the parser file, caliper_parser.py add a class

```
from avocado.core.plugin_interfaces import JobPre, JobPost

class Parser(JobPre, JobPost):
    name = 'Parser'
    description = ' Description ...'

    def __init__(self):

    def pre_parse(self, job):
        #Add code to be executed as pre job

    def post_parser(self, job):
        #Add code to be executed as post job

    pre = pre_parse
    post = post_parser
```

Multiple configuration/Multiplexer

To support executions for Single core, Four core (single cluster), 16 core (super cluster) tests, multiple configurations may be used.

Example
A simple prototype was made to experiment this features.
The command “numactl --physcpubind=%s --membind=0 ls” (where %s can be “0” or “0,1,2,3” or “0-15”) as below may be used experiment.

Test.py

```
    def test(self):
    cpubind = self.params.get('cpubind', default=0)
    cmd = 'numactl --physcpubind=%s --membind=0 ls' % (cpubind)
    self.log.debug("Command executing ...[%s]", cmd)
    process.system(cmd)
```

Test.yaml

```
!mux
single:
```

```
      cpubind: 0
cluster:
      cpubind: 0,1,2,3
super:
      cpubind: 0-15
```

This test creates three output folders with individual test run status, stdout/stderr and logs.

Parsers for benchmarks

Test python (dhrystone.py) code needs to be updated with regular expression markers for caliper parsers to work correctly. This involves inserting code as shown below.

Hardware info script updated with markers

```
      #===Pre Markers ===
      cmd = 'hardwareinfo'
      start_log = "%%%%%%        %s test start        %%%%%% \n" %
cmd
      echo_cmd = "echo '%s' " % start_log
      process.run(echo_cmd)
      echo_cmd = "echo '<<BEGIN TEST>>>'"
      process.run(echo_cmd)
      #===Test Command===
      process.run('./hw_info_run.sh')
      # ===Post Markers===
      echo_cmd = "echo '[status]: PASS'"
      process.run(echo_cmd)
      echo_cmd = "echo '<<<END>>>'"
      process.run(echo_cmd)
      echo_cmd = "echo '%%%%%% test_end %%%%%%'"
      process.run(echo_cmd)
      # ===End ===
```

Build/Make test cases

Presently, Download, Configure and Build/Make are done as part of ‘SetUP()’ function call of post job interfaces of the tests. (Example in file dhrystone.py)

```
class Dhrystone(Test):
      def setUp(self):
      '''
```

```
Download, Configure, Build done here
'''
```

Avocado installation on ARM64 and Ubuntu 16.04

Commands

```
sudo apt-get install -y git gcc python-dev python-pip libvirt-dev libyaml-dev
git clone git://github.com/avocado-framework/avocado.git
```

Make requirements failed while compiling cryptography.
So needs to install dependency as per https://cryptography.io/en/latest/installation/

```
sudo apt-get install build-essential libssl-dev libffi-dev
```

The installation fail for lzma library, fixed as below.

```
sudo apt-get install liblzma-dev
```

Reports

For Report generation we need to follow the same steps as in Caliper, as it is not yet included in this Proof of Concept implementation of Caliper-nxt.

Restrictions

There are feature present in Caliper, but not available in Avocado or Caliper-next frameworks as of now (February 2017). For example multi-node tests, for which there is development/discussions are going on in Avocado forum for the implementation details. Once it is available it will not be difficult to use it through Caliper-nxt.

But there are workarounds available for most of the restrictions present in the Avocado framework. For example, multi-node tests may be implemented by invoking multiple Avocado test instances. Details are discussed in the Avocado forum.

Future work

The support for feature like multi-node, multi-instance tests are need to be added. Tests for Virtual Machines, Dockers, Cloud and other data centre specific tests etc. needs to be explored to be included.

References

https://avocado-framework.github.io/
http://open-estuary.org/caliper-benchmarking/
https://github.com/open-estuary/caliper-nxt