Caliper Developer Manual	Public
1.0	Total 10 pages

Caliper [Open source benchmarking framework] Developer Manual

Shyju PV (shyju.pv@gmail.com), Pinkesh shah (pinkesh.shh@gmail.com)

Table of Contents

1 Introduction

The user should go through the caliper user manual to understand the caliper flow and execute the integrated tools.
This document would further explain about how new tools to be added to the current framework and modifications to be done in various modules of caliper.

To add a new tool to Caliper one should define how to build, execute and parse the output of that tool. The configuration files should be updated to specify how the target, server, client to be configured and tools to be executed. One should be familiar with the directory structure of caliper before adding any tools.

2 Caliper folder organisation

Caliper folder structure is as follows.

benchmarks: This directory contains the actual source code for each tool.

client: The parser files for each tools are placed under client/parser directory.

config: This folder contains **client_config.cfg file**. This file is used to configure the host, target and client/server information.

server: This directory contains the scripts for dispatching the build, run and parse on the Host and remote login in the Target.

> **build:** This directory mainly contains the build.sh and build.py which are responsible for building of tools.
>
> **compute_model:** This directory includes the files to get the score from the parser output; the method of scoring is mainly in the scores_method.py.
>
> **parser_process:** This one contains the source code to process the yaml file for scoring and normalisations.
>
> **run:** This directory mainly includes the test_run.py, it is the main code to run the commands for execution of tools and parsing the output of each tools.
>
> **hosts:** This directory mainly contains the class of hosts and how to use hosts.

test_cases_cfg: This contains the config files to select the tools and test cases to be executed.

frontend: This folder contains all the files necessary for the report generation.

Frontend/polls: This folder contains the following files:

> **plot:** This folder contains the files for generating the graphs.
>
> **templates/polls**: This folder contains all template htmls.
>
> **static**: This folder contains all javascripts and css related files.
>
> **views.py:** This file is responsible to get the data from the yamls and jsons.

utils: contains **automation_scripts** that contains all the scripts for dependency checking and automation of caliper execution and **Documentation** folder having excel population scripts.

Note : *Caliper python django framework for report generation.*

3 Add Tool to benchmark directory

Please find the specific tools folders under the benchmark directory. You need to add your tool source code similarly in this folder.

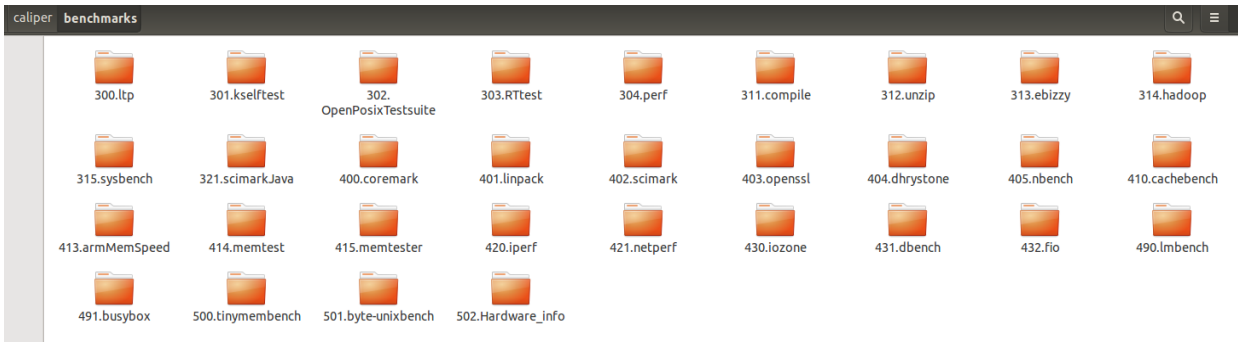


Illustration 1: Caliper benchmark directory

4 Test case configuration structure

The directory named test_cases_cfg is the key of how to build, run and parse the tool. The content of it is listed below.

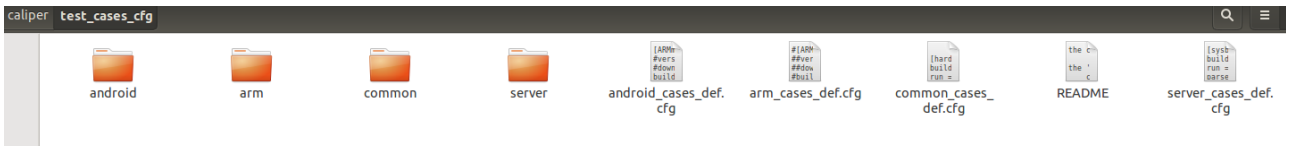


Illustration 2: Test_cases_cfg directory structure

Depending on the tool category, tool configuration files (tool_build.sh and tool_run.cfg) should be placed in **server / common / application** directory.

1. If the tool will execute on the target platform, then create the tool directory into **test_cases_def/common/** directory. Example: openblas, Dhrystone, etc.
2. If the tool execution required client-server configuration and the "**Target**" platform acts as **client** machine, then create the tool directory into **test_cases_def/server** directory. Example: iperf, netperf, etc.
    1. Write the test cases into tool_run.cfg file. These test cases will executes on "**Target**".
    2. Write the test cases into tool_server_run.cfg file. These test cases will executes on "**TestNode**".
3. If the tool execution required client-server configuration and the "**Target**" platform acts as **server** machine, then create the tool directory into **test_cases_def/application** directory. Example: nginx, redis etc.
    1. Write the test cases into tool_run.cfg file. These test cases will executes on "**Target**".
    2. Write the test cases into tool_application_run.cfg file. These test cases will executes on "**TestNode**"

Create the "tool name" directory in any of the required folders and create the **tool_run.cfg and tool_build.sh.**

Tool_run.cfg will be in following format:

```
[lmbench lat]
category = Performance
scores_way =  exp_score_compute 1 -0.5
command = "cd lmbench; ./test.sh latency "
parser = lmbench_lat_parser

#[lmbench bandwidth]
#category = Performance
#scores_way = compute_speed_score 3
#command = "cd lmbench; ./test.sh bandwidth "
#parser = lmbench_bandwidth_parser
```

Illustration 3: Tool_run.cfg format

4.1 Tool_name_run.cfg configuration

The below section explains the contents of the tool_name_run.cfg file.

[*test_case_name*]
category = category scenario sub_scenario test_case_name
scores_way = compute_speed_score 1
command = ./bin/*tool_name*
parser = *tool_name*_parser

- The **category** key defines the categorization of the tool in the resultant yaml. Category key can be **functional** or **performance**. Scenario can be network, application, memory, etc. Sub-scenario can be specific to the test case category.

- The **scores_way** defines the way to compute the score for the parsed values.
  Two scoring methods are available for computing the score.

  For **Higher The Better** values use
  scores_way = compute_speed_score index
  where the score is calculated as: score_way = test case output value / (10^index)

  For **Lower The Better** values use
  scores_way = exp_score_compute *base index*
  Where the score is calculated as: *score_way = (value/(10^base))^ index*

  Example:
  Bandwidth values can be categorized in **Higher The Better** scenario.
  Latency values can be categorized in **Lower The Better** scenario.

  Note:
  1. For redis tool, 2 score way method has been used. Because the single test case of redis contains both bandwidth and latency.
  2. For redis tool, "**scores_way**" is for latency and "**scores_way1**" is for bandwidth.
  3. If user wants to add such type of tool, which contain bandwidth and latency in single tets case, please refer redis tool configuration file and parser file.

- The **command** is the actual test case which will run on target.
- The **parser** key defines the method that will parse the output of the command executed above. Refer section 7 for more details.

The category should contain maximum 4 levels depending on the value returned by parser method.
Based on the type of value returned by the parser the length of category varies.

- If the returned value is a dictionary of depth 3 only **category** should be mentioned.
- If the returned value is a dictionary of depth 2 **category [scenario]** should be mentioned.

- If the returned value is a dictionary of depth 1 **category [scenario] [sub_scenario]** should be mentioned.
- If the returned value is of int, float or any other primitive data type then **category [scenario] [sub_scenario] [test_cases]** should be mentioned.

Example - lmbench_run.cfg:

[lmbench bandwidth stream v1]
category = Performance misc lmbench_bandwidth_stream_v1
scores_way = compute_speed_score 3
command = "if [-f lmbench_tar.gz]; then tar -xvf lmbench_tar.gz; rm lmbench_tar.gz; fi;cd lmbench; numactl --cpunodebind=0 --localalloc ./stream -v 1 -M 200M -P 16"
parser = lmbench_bandwidth_stream_v1

- Here, **category** contains 3 levels - **Performance, misc and lmbench_bandwidth_stream_v1**.
- The parser method **lmbench_bandwidth_stream_v1** returns the value in dictionary format like,

dic['copy'] = {}
dic['scale'] = {}
dic['add'] = {}
dic['triad'] = {}

4.2 Tool_name_build.sh configuration

- ***tool*_build.sh** file contains the procudure to build the binary image of the tool for x86_64 and arm_64 target platforms.
- Sample script file for scimark tool is as follows:

```
build_scimark() {
    set -e
    SrcPath=${BENCH_PATH}"402.scimark"
    myOBJPATH=${INSTALL_DIR}/bin
    pushd $SrcPath
    if [ $ARCH = "x86_32" -o $ARCH = "x86_64" ]; then
    make CC=$GCC CFLAGS="-msse4"
        cp scimark2 $myOBJPATH/
                    make CC=$GCC clean
                                fi
    if [ $ARCH = "arm_32" ]; then
        # -mfloat-abi=hard -mfpu=vfpv4 -mcpu=cortex-a15
        make CC=$GCC CFLAGS=" -mfloat-abi=hard -mfpu=vfpv4 -mcpu=cortex-a15 "
        cp scimark2 $myOBJPATH/
        make CC=$GCC clean
    fi
    if [ $ARCH = "arm_64" ]; then
        make CC=$GCC       #CFLAGS="-mabi=lp64"
        cp scimark2 $myOBJPATH
        make CC=$GCC clean
    fi

    if [ $ARCH = "android" ]; then
        ndk-build
        cp libs/armeabi-v7a/scimark2 $myOBJPATH/
        ndk-build clean
        rm -rf libs/ obj/
    fi
    popd
}
```

Illustration 4: Tool_build.sh format

Make changes in this file as per the requirements to build the new tool getting added. The flag GCC is alredy defined to the target architecture by the caliper framework.

5 Tool Parsing

The output of the execution needs to be parsed for the relevant information from the output log of the tool

execution. Parser file of each tool is present in **client/parser** directory. The parser method name defined in tool_name_parser.py should be same as defined in "**parser**" line in the tool_name_run.cfg file.

There are following sections in the tool_name_parser.py file:

- Import python modules.
- Parser method definition.

Below example shows the parser logic for **stressng** tool:

```
1. import re
2. def stress_ng_parser(content,outfp):
3.     result = 0
4.     if re.search(r'stress-ng: info:\s+\[[0-9]+\]\scpu\s+\d+\s+(\d+\.\d+).*',content):
5.         real_time = re.search(r'stress-ng: info:\s+\[[0-     9]+\]\scpu\s+\d+\s+(\d+\.\d+).*',content)
6.         result =  real_time.group(1)
7.     outfp.write(content)
8. return result
```

- The method must have two arguments: First argument contains the **content** of output log and the second argument is the **file pointer** which points to the parser log file.
- Here line number 4 to 6 contains the parsing logic – re.search(). Depending on the requirements to parse the output log, user can write this logic.
- In addition the parser must return a value or dictionary which is needed later for raw yaml generation and score computing.

6 Addition of tool in main test_cases_cfg directory

According to the categorization of tool under **common / server / application** category, update common_cases_def.cfg / server_cases_def.cfg / application_cases_def.cfg file with following information.

```
[tinymembench]
build = tinymembench_build.sh
run = tinymembench_run.cfg
parser = tinymembench_parser.py

[openssl]                  #this is the name of the tool
build = openssl_build.sh   #tool_name_build.sh
run = openssl_run.cfg      #tool_name_run.cfg
parser = openssl_parser.py#tool_name_parser.py
```

Illustration 5: Tool Definition

New tool can be categorized in following way:

- If the tool will execute on the target platform, then place the tool into test_cases_def/common_cases_def.cfg file. Example: openblas, Dhrystone, etc.
- If the tool execution required client-server configuration and the "**Target**" platform acts as **client** machine, then place the tool into **test_cases_def/server_cases_def.cfg** file. Example: iperf, netperf, etc.
- If the tool execution required client-server configuration and the "**Target**" platform acts as **server** machine, then place the tool into **test_cases_def/application_cases_def.cfg** file. Example: nginx, redis etc.

7 Graph Generation

- Once the execution of caliper is completed the resultant yamls are generated in the **caliper_output/<workspace_name>/output/results/yaml** directory.
- There are 3 files inside this directory: **<platform_hw_info>.yaml**, <p**latform_name>.yaml and** <p**latform_name>_score.yaml.**
- <p**latform_name>_score.yaml** is taken as an input for report generation.
- The values in <p**latform_name>_score.yaml** file are normalized and the graphs are generated using these values.
- There is an html page for each category as **caliper/frontend/polls/templates/polls/<category_name>.html**. (**Note:** update all code changes in caliper source code. Do not update **~/caliper_output/frontend** directory while adding a new tool.)
- Refer below example for adding **bandwidth** scenario in **nework.html** category file.

Note: please focus on highlighted text.

Refer below screen-shot is for **network.html** file.

```
1   {% if bandwidth %}
2   <script src="{% static 'polls/js/example/network/network_bandwidth.js' %}"></script>
3   {% endif %}
4
5 </head>
6 <body>
7 <div class="example">
8   <div><input type="hidden" id="network_tst" value="{{ dic_net }}"> </div>
9
10   {% if bandwidth %}
11   <h2><small>Network Score Details: Bandwidth</small></h2>
12   <div style="text-align: center;">
13   <p> <img src="{% static 'polls/pictures/network_bandwidth.png' %}" height="400" width="850" /> </p>
14     </div>
15   <div id="network-bandwidth" class="sensei-grid"></div>
16   {% endif %}
```

Illustration 6: network.html file screen-shot

- Line number 1 and 10 contains the scenario name. In this example, the scenario name is **"bandwidth"**.
- Second line contains the java script filename. Create **network_bandwidth.js** file into **caliper/frontend/polls/static/polls/js/example/<category_*name*>** directory.
    - Caliper framework follows a specific naming convention for creating the java script (js) file for each **scenario** under a particular **category**.
    - In the above case, the js file name is **network_bandwidth.js**, which is created from category_scenario.js naming convention.
- The id name are defined as **"network_tst"** and **"network-bandwidth"** at line number 8 and 15 respectively.
- These ids are used to link the test scenario of network in the **network_bandwidth.js** file.
- **network_bandwidth.png** file will be created automatically by caliper framework. This file will be used to plot graphs in HTML web page. The naming convention should be follow as described above (category_scenario.png).

Refer below screen-shot is for **network_bandwidth.js** file.

```
1   var test = document.getElementById("network_tst").value;
2   var bandwidth_dic = getJson(test, 'bandwidth')
3   var columns = getHoriColumn(bandwidth_dic);
4   var data = getHoriData(bandwidth_dic, columns);
5
6   // initialize grid
7   var options = {emptyRow: true, sortable: false};
8     var grid = $(document.getElementById("network-bandwidth")).grid(data, columns, options);
9   draw_grid(grid);
```

Illustration 7: network_bandwidth.js file screen-shot

- **"network-tst"** and **"network-bandwidth"** ids are linked to **network.html** file as described above.
- **"bandwidth**" defined at line number 2 indicates the scenario name.
- After updating all the necessary files in frontend directory, copy **caliper/frontend/** directory into **~/caliper_output/** directory.

8 Excel sheet update for new test cases

- When new tool has been added into caliper framework, following excel file needs to update inside **caliper/frontend/polls/templates/polls/** directory:
- **Performance_Template.xlsx** or **Functional_Template.xlsx**
- Based on tool category, Performance_Template.xlsx or Functional_Template.xlsx to be updated.
- After adding network test cases for iperf and netperf tools, the excel file (**Performance_Template.xlsx** file in this case) will looks like as follow:

	A	B	C	D	E	F	G	H	I
1	**Caliper Performance Tests: Network**								
2	**SL NO**	**TOOLS**	**Scenario**	**Testcase**	**Units**				
3	1	netperf	bandwidth	TCP_RR	MB/s				
4		netperf	bandwidth	TCP_stream_r	MB/s				
5		netperf	bandwidth	TCP_CRR	MB/s				
6		netperf	bandwidth	TCP_stream	MB/s				
7		netperf	bandwidth	UDP_RR	MB/s				
8	2	iperf	bandwidth	TCP_s01_RX	Mbits/sec				
9		iperf	bandwidth	TCP_s01_TX	Mbits/sec				
10		iperf	bandwidth	TCP_s03_RX	Mbits/sec				
11		iperf	bandwidth	TCP_s03_TX	Mbits/sec				
12		iperf	bandwidth	TCP_s04_RX	Mbits/sec				
13		iperf	bandwidth	TCP_s04_TX	Mbits/sec				
14		iperf	bandwidth	TCP_s05_RX	Mbits/sec				
15		iperf	bandwidth	TCP_s05_TX	Mbits/sec				
16		iperf	bandwidth	TCP_s10_RX	Mbits/sec				
17		iperf	bandwidth	TCP_s10_TX	Mbits/sec				
18		iperf	bandwidth	UDP_s01_TX	Mbits/sec				
19		iperf	bandwidth	UDP_s01_RX	Mbits/sec				
20		iperf	bandwidth	UDP_s03_RX	Mbits/sec				
21		iperf	bandwidth	UDP_s03_TX	Mbits/sec				
22		iperf	bandwidth	UDP_s05_RX	Mbits/sec				
23		iperf	bandwidth	UDP_s05_TX	Mbits/sec				
24		iperf	bandwidth	UDP_s10_RX	Mbits/sec				
25		iperf	bandwidth	UDP_s10_TX	Mbits/sec				

- If user wants to merge cell names in **TOOLS** or **Scenario** column, as shown in above figure, then please make sure that each of cells should contains the proper contents. Otherwise values will be not populating for that test case.

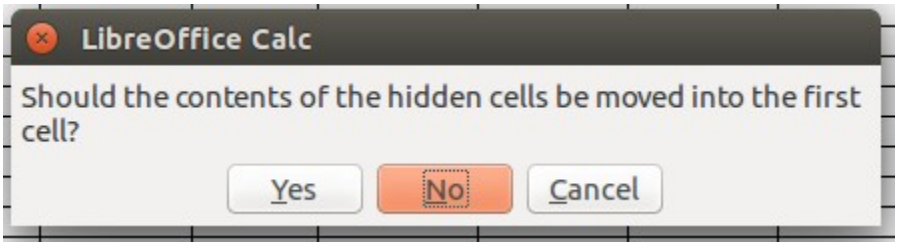
Follow below steps to merge the cells:

Step 1: Select the cells which user wants to merge.

	A	B	C	D	E	F	G	H	I
1	**Caliper Performance Tests: Network**								
2	**SL NO**	**TOOLS**	**Scenario**	**Testcase**	**Units**				
3	1	netperf	bandwidth	TCP_RR	MB/s				
4		netperf	bandwidth	TCP_stream_r	MB/s				
5		netperf	bandwidth	TCP_CRR	MB/s				
6		netperf	bandwidth	TCP_stream	MB/s				
7		netperf	bandwidth	UDP_RR	MB/s				
8	2	iperf	bandwidth	TCP_s01_RX	Mbits/sec				
9		iperf	bandwidth	TCP_s01_TX	Mbits/sec				
10		iperf	bandwidth	TCP_s03_RX	Mbits/sec				
11		iperf	bandwidth	TCP_s03_TX	Mbits/sec				
12		iperf	bandwidth	TCP_s04_RX	Mbits/sec				
13		iperf	bandwidth	TCP_s04_TX	Mbits/sec				
14		iperf	bandwidth	TCP_s05_RX	Mbits/sec				
15		iperf	bandwidth	TCP_s05_TX	Mbits/sec				
16		iperf	bandwidth	TCP_s10_RX	Mbits/sec				
17		iperf	bandwidth	TCP_s10_TX	Mbits/sec				
18		iperf	bandwidth	UDP_s01_TX	Mbits/sec				
19		iperf	bandwidth	UDP_s01_RX	Mbits/sec				
20		iperf	bandwidth	UDP_s03_RX	Mbits/sec				
21		iperf	bandwidth	UDP_s03_TX	Mbits/sec				
22		iperf	bandwidth	UDP_s05_RX	Mbits/sec				
23		iperf	bandwidth	UDP_s05_TX	Mbits/sec				
24		iperf	bandwidth	UDP_s10_RX	Mbits/sec				
25		iperf	bandwidth	UDP_s10_TX	Mbits/sec				
26				TCP_bw_000001B	Mbytes/sec				

Step 2: Press "**Merge and Center Cells**" button in meubar in LibreOffice.

Step 3: Refer below screen-shot and select "**No**".



After updating the xlsx file, copy it into **~/caliper_output/frontend/polls/templates/polls/** directory.

Note:

1. Update **~/caliper_output/frontend/polls** directory for javascript, html and Performance_Template.xlsx changes manually.
2. After updating all the necessary files, install the caliper with ***sudo python setup.py install*** command.