perfwhiz Documentation

Release 0.3.1

Cisco Systems, Inc.

September 10, 2016

Contents

1 Overview **3**

1.1 perfwhiz Workflow . . . 3

1.2 Dependencies . . . 4

1.3 Licensing . . . 5

1.4 Links . . . 5

2 Installation **7**

2.1 perfcap.py . . . 7

2.2 perfmap.py . . . 8

3 Usage **9**

3.1 Examples of captures . . . 9

3.2 Examples of chart generation . . . 9

3.3 Task Name Annotation . . . 10

3.4 CSV Mapping file . . . 11

3.5 OpenStack Plug-In . . . 12

4 Perfwhiz FAQ **13**

4.1 Can I get charts for existing perf data files? . . . 13

5 Indices and tables **15**

Contents:

CHAPTER 1

Overview

This repository contains a set of python scripts for helping tune any Linux system for performance and scale by leveraging the Linux *perf* tool and generating from the perf traces 2 HTML dashboards that represent Linux scheduler context switches and KVM events.

The basic dashboard illustrates general scheduler and KVM events for all tasks selected at capture time:

- CPU Usage vs. context switches chart
- KVM exit types distribution stacked bar charts (exit type distribution per task)
- Summary CoreMap showing task scheduler core assignment and context switch count heat maps (run time % and context switch count on each core per task - including total time per core and per task)

The detailed dashboard illustrates detailed scheduler and KVM events at the task level:

- context switch heat maps (temporal distribution of context switch events)
- KVM exit heat maps (temporal distribution of kvm entry and exit events)
- Temporal Coremaps (on which core does any given task run over time)

The task annotation feature further allows the generation of cross-run charts (diffs) that can help detect more easily differences of behavior across multiple captures/runs.

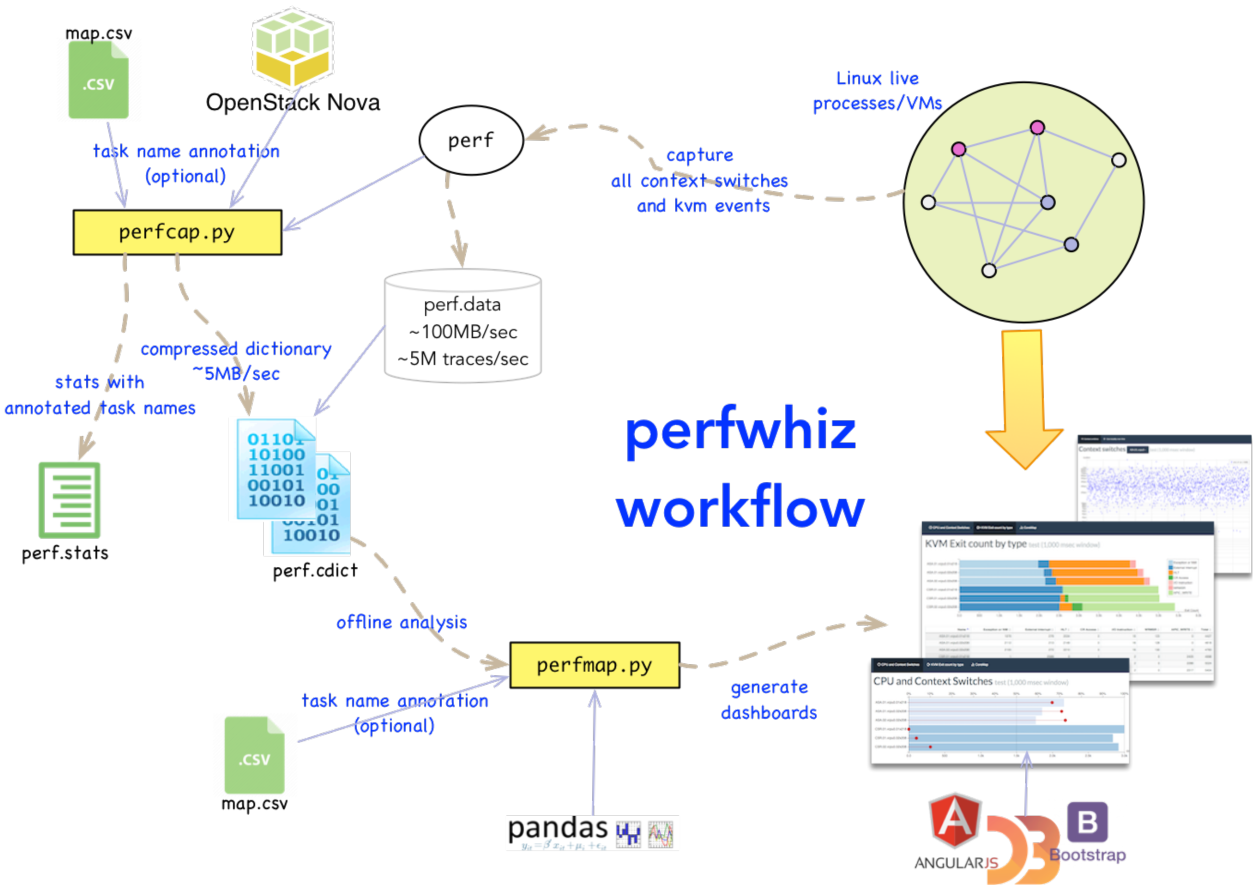
The capture script wraps around the Linux perf tool to capture events of interest (such as context switches and kvm events) and generates a much more compact binary file to be used for analysis offline.

Complete documentation including installation and usage instructions:

http://perfwhiz.readthedocs.org/

1.1 perfwhiz Workflow

The following diagram illustrates the 2 phases for getting the dashboards: capture phase (perfcap.py) and dashboard generation phase (perfmap.py).



Capture is performed on the system under test, by invoking the perfcap.py script and specify which tasks to capture events from and for how long. The result of each capture is a binary file (with the cdict extension, cdict stands for compressed dictionary).

The binary files can then later be provided to the dashboard generation tool (perfmap.py) to generate the corresponding dashboards. This dashboard generation phase is typically done offline on a workstation where perfmap is installed (laptop, macbook...). The generated dashboards are HTML files that can be viewed using any browser.

1.2 Dependencies

Dependencies are automatically installed when perfwhiz is being installed (refer to the Installation section).

The capture tool perfcap.py depends on:

- the Linux perf tool (compiled with the python extension)
- pbr python package
- msgpack python package

The dashboard generation tool perfmap.py depends on:

- pandas/numpy python package

The generated HTML dashboards contain Javascript code that will pull some Javascript libraries from CDN servers when loaded in the browser (CDN is a public network of servers that contain libraries that are downloaded by browsers). Therefore, viewing those dashboards require access to the Internet. The following Javascript libraries are required by the dashboards:

- jquery
- datatables
- d3
- angular
- angular-ui-bootstrap
- pako (zlib inflate)

1.3 Licensing

perfwhiz is licensed under the Apache License, Version 2.0 (the "License"). You may not use this tool except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Perfwhiz includes a copy of umsgpack.py (MIT License) in its distribution (https://github.com/vsergeev/u-msgpack-python, many thanks to vsergeev)

1.4 Links

- Documentation: http://perfwhiz.readthedocs.org/en/latest/
- Source: https://github.com/cisco-oss-eng/perfwhiz
- Supports/Bugs: https://github.com/cisco-oss-eng/perfwhiz
- Gitter Room: https://gitter.im/cisco-oss-eng/perfwhiz

CHAPTER 2

Installation

perfwhiz comes with two scripts, perfcap.py and perfmap.py. Normally, perfcap.py will be installed on the target node for capturing the perf data, perfmap.py can be ran from any machine to plot different types of chart from the capture data.

Both scripts are available in PyPI, and can be installed using either “pip install” or source code based installation. In either option of installation, please have python and python-dev installed.

Ubuntu/Debian based:

```
$ sudo apt-get install python-dev python-pip python-virtualenv
```

RHEL/Fedora/CentOS based:

```
$ sudo yum install gcc python-devel python-pip python-virtualenv
```

You may also want to create a python virtual environment if you prefer to have isolation of python installations (this is recommended but optional):

```
$ virtualenv pft
$ source pft/bin/activate
```

Remember to activate your virtual environment every time before installing or using the tool.

2.1 perfcap.py

perfcap.py is a wrapper around Linux perf tool. In order to run perfcap.py, the native perf tool must be built with Python extension. Run below command to check whether this feature is pre-built in the perf version provided by your distro:

```
$ sudo perf script -g python
```

If you see below message, congratulations! It is ready to use out-of box:

```
generated Python script: perf-script.py
```

If you see below message, unfortunately you have to do some extra works to rebuild the tool:

```
$ sudo perf script -g python
Python scripting not supported.  Install libpython and rebuild perf to enable it.
For example:
  # apt-get install python-dev (ubuntu)
  # yum install python-devel (Fedora)
  etc.
```

Normally, if you are using a RHEL/CentOS distro, the tool from official repository has been built with Python extension already. If you are using a Ubuntu distro, unfortunately you have to rebuild perf and enable the Python scripting extension. Refer to here for the details on the steps to follow to rebuild perf.

Installation from PyPI will be as easy as:

```
$ pip install perfwhiz
```

Installation from source code be as easy as:

```
$ git clone https://github.com/cisco-oss-eng/perfwhiz.git
$ cd perfwhiz
$ pip install -e.
```

Once installation is finished, you should be able to verify the installation by doing:

```
$ python perfcap.py -h
```

Note that perfcap can be used without pip install (just git clone). In that case, captures will compress the results using pure python code, which makes it relatively slower than when installing requirements using pip install (as this pulls in a faster compression library).

2.2 perfmap.py

perfmap, the analyzer tool of perfwhiz, will do the data analysis based on different scheduler events, and draw the charts to present them. It is using Pandas, Numpy to perform data processing, Bokeh and D3.js for charts.

Installation from PyPI will be as easy as:

```
$ pip install perfwhiz[analyzer]
```

Installation from source code be as easy as:

```
$ git clone https://github.com/cisco-oss-eng/perfwhiz.git
$ cd perfwhiz
$ pip install -e.[analyzer]
```

Once installation is finished, you should be able to verify the installation by doing:

```
$ python perfmap.py -h
```

CHAPTER 3

Usage

3.1 Examples of captures

Note: an installation using pip will provide wrappers that can be called directly from the shell command line (e.g. "perfcap" or "perfmap").

These wrappers are not available with an installation using a git clone and the corresponding python scripts must be called using the python executable (e.g. "python perfcap.py ...").

Note: trace captures *require root permission* (perfcap must be called by root or using sudo).

Capture context switch and kvm exit traces for 5 seconds and generate traces into test.cdict:

```
perfcap -s 5 --all test
```

Capture all traces for 1 second and use the "tmap.csv" mapping file to assign logical task names to task IDs:

```
perfcap --all --map tmap.csv test2
```

Traces will be stored in the corresponding cdict file (e.g. "test2.cdict").

Generate the cdict file for an existing perf data file and name the resulting cdict file "oldrun.cdict":

```
perfcap --use-perf-data perf.data oldrun
```

3.2 Examples of chart generation

Generate the basic dashboard containing charts for all tasks with a name ending with "vcpu0" from the "test.cdict" capture file:

```
perfmap.py -t '*vcpu0' test.cdict
```

Generate the heatmaps dashboard containing charts for all tasks with a name ending with "vcpu0" from the "test.cdict" capture file for the first 1000ms of capture (for heatmaps using smaller windows might be required in order to limit the density of the heatmaps):

```
perfmap.py -t '*vcpu0' --heatmaps -c 1000 test.cdict
```

Only show 1000 msec of capture starting from 2 seconds past the start of capture for all tasks:

```
perfmap.py -t '*' -c 1000 -f 2000 test.cdict
```

Generate the basic dashboard with diff charts for 2 capture files:

```
perfmap.py -t '*vcpu0' test.cdict test2.cdict
```

3.3 Task Name Annotation

Analyzing data with pid, tid and the raw task name may not always be optimal because numeric task/process IDs are not very meaningful and the raw task name may be a generic name (such as “/usr/bin/qemu-system-x86_64” for a VM or “/usr/bin/python” for a python process). Annotating task names allows such non descript tasks to be renamed for chart display purpose.

One additional benefit of annotating task names is that it allows easier comparative analysis across runs that may involve re-launching the tested processes (and in that case will have different task or process IDs).

For example assume each run requires launching 2 groups of 2 instances of VMs where each VM instance plays a certain role in its own group (router and firewall role, each group has 1 router and 1 firewall VM, forming what is called a service chain).

Without annotation the analysis will have to work with generic names such as:

Run #1:

```
- vm_router, pid1 (group 1)
- vm_router, pid2 (group 2)
- vm_firewall, pid3 (group 1)
- vm_firewall, pid4 (group 2)
```

Run #2:

```
- vm_router, pid5 (group 1)
- vm_router, pid6 (group 2)
- vm_firewall, pid7 (group 1)
- vm_firewall, pid8 (group 2)
```

The group membership for each process is completely lost during capture, making comparative analysis extremely difficult as you’d need to make a mental association of pid1 to pid5, pid2 to pid6 etc...

Worst, with the use of default non decript names you’d have to juggle with tasks such as:

- /usr/bin/qemu-system-x86_64, pid1
- /usr/bin/qemu-system-x86_64, pid2
- etc...

With annotation, the task name could reflect the role and group membership:

Run #1:

```
- vm_router.1, pid9
- vm_router.2, pid10
- vm_firewall.1, pid11
- vm_firewall.2, pid12
```

Run #2:

```
- vm_router.1, pid13
- vm_router.2, pid14
- vm_firewall.1, pid15
- vm_firewall.2, pid16
```

It is much easier to analyze for example how heat map tasks relate to group membership or how the vm.router in each group compare across the 2 runs.

Task name annotation is supported by both perfcap.py and perfmap.py.

The perfcap.py script supports annotating task names at capture time using either a CSV mapping file or the OpenStack plug-in. Annotating will mean that the generic task name will be replaced by the annotated name right after the perf capture is done and while creating the cdict file.

The perfmap.py script supports annotating task names using the CSV mapping file method only. In this case, the task name replacement will happen while loading the data from the cdict file.

In general it is better to annotate earlier (at capture time) as it results in annotated cdict files and will avoid having to tow along the mapping file corresponding to each cdict file.

3.4 CSV Mapping file

A mapping file is a valid comma separated value (CSV) text file that has the following fields in each line:

CSV format:

```
<tid>,<libvirt-instance-name>,<task-system-type>,<uuid>,<group-type>,<group-id>,<task-name>
```

Table 3.1: CSV field description

name	description
<tid>	linux task ID (also called thread ID)
<libvirt-instance-name>	libvirt instance name (VM) - ignored
<task-system-type>	a task type (VM: emulator or vcpu task)
<uuid>	instance uuid (OpenStack instance) - ignored
<group-type>	type of grouping (e.g. service chain type name) - ignored
<group-id>	indentifier of the group to distinguish between multiple groups (e.g. service chain number)
<task-name>	name of the task - describes what the task does (e.g. firewall or router...)

Example of mapping file:

```
19236,instance-000019f4,vcpu0,8f81e3a1-3ebd-4015-bbee-e291f0672d02,FULL,5,Firewall
453,instance-00001892,emulator,4a81e3cc-4de0-5030-cbfd-f3c43213c34b,FULL,2,Router
```

Equivalent simplified version:

```
19236,,vcpu0,,,5,Firewall
453,,emulator,,,2,Router
```

In the current version, the annotated name is calculated as:

```
<task-name>.<group-id>.<task-system-type>
```

The <tid> is used as a key for matching perf records to annotated names (i.e. all perf records that have a tid matching any entry in the mapping file will have their task name renamed using the above annotated name). All other fields are therefore ignored.

Resulting annotated name from the above example:

```
Firewall.05.vcpu0
Router.02.emulator
```

The helper script create-vm-csv.sh that is included in the git repository illustrates how such csv file can be created before capturing the traces.

3.5 OpenStack Plug-In

Task name mapping can be performed automatically when VMs are being launched by OpenStack. In that case, the perfcap.py script will query OpenStack to retrieve the list of VM instances and deduct the task name mapping by associating OpenStack instance information to the corresponding task ID. This feature is still experimental and may be moved out of perfwhiz completely into a separate tool that generates the CSV mapping file from OpenStack queries.

CHAPTER 4

Perfwhiz FAQ

4.1 Can I get charts for existing perf data files?

Yes it is possible to generate cdict files from existing perf binary data files. You just need to make sure that the perf capture includes context switching events (and kvm evemts if needed). Use the perfcap tool with the –use-perf-data argument to pass the perf data file (see usage)

CHAPTER 5

Indices and tables

- genindex
- modindex
- search