pbench Documentation

Pbench

Dec 04, 2020

PBENCH-AGENT

A Benchmarking and Performance Analysis Framework

Pbench Agent

> The Agent is responsible for providing commands for running benchmarks across one or more systems, while properly collecting the configuration of those systems, their logs, and specified telemetry from various tools (sar, vmstat, perf, etc).

Pbench Server

> The second sub-system included here is the Server, which is responsible for archiving results and indexing them to allow the dashboard to prepare visualizations of the results.

Dashboard

> Lastly, the Dashboard is used to display visualizations in graphical and other forms of the results that were collected by the Agent and indexed by the Server.

The pbench Dashboard code lives in its own repository.

Table of Contents

CHAPTER

ONE

USER GUIDE

Contents

1.1 What is Pbench?

Pbench is a harness that allows data collection from a variety of tools while running a benchmark. Pbench has some built-in scripts that run some common benchmarks, but the data collection can be run separately as well with a benchmark that is not built-in to Pbench, or a Pbench script can be written for the benchmark. Such contributions are more than welcome!

1.2 TL;DR - How to set up Pbench and run a benchmark

Prerequisite: Somebody has already done the server setup.

The following steps assume that only a single node participates in the benchmark run. If you want a multi-node setup, you have to read up on the –remote options of various commands (in particular, pbench-register-tool-set):

- Install the agent
- Customize the agent for your server environment. This will vary from installation to installation, but it fundamentally involves copying two files that should be made available to you somehow by an admin type: an ssh private key file to allow the client(s) to send results to the server and a configuration file that should be installed in `/opt/pbench-agent/config/pbench-agent.cfg`. There is an example configuration file in that location, but you need the "real" one for your environment. Among other things, the config file specifies the IP or hostname of the server.
- Run your benchmark with a default set of tools:

```
. /etc/profile.d/pbench-agent.sh          # or log out and log back in
pbench-register-tool-set
pbench-user-benchmark -C test1 -- ./your_cmd.sh
pbench-move-results
```

- Visit the Results URL in your browser to see the results: the URL depends on the server hostname or IP"; assuming that the server is "pbench.example.com" and assuming you ran the above on a host named "myhost", the results will be found at (**N.B.:** this is a fake link serving as an example only - talk to your local administrator to find out what server to use to get to Pbench results): http://pbench.example.com/results/myhost/pbench-user-benchmark_test1_yyyy-mm-dd_HH:MM:SS.

For explanations and details, see subsequent sections.

1.3 How to install

See the install section for details.

1.4 Defaults

The benchmark scripts source the base script (/opt/pbench-agent/base) which sets a bunch of defaults:

```
pbench_run=/var/lib/pbench-agent
pbench_log=/var/lib/pbench-agent/pbench.log
date=`date "+%F_%H:%M:%S"`
hostname=`hostname -s`
results_repo=pbench@pbench.example.com
results_repo_dir=/pbench/public_html/incoming
ssh_opts='-o StrictHostKeyChecking=no'
```

These are now specified in the config file `/opt/pbench-agent/config/pbench-agent.cfg`.

1.5 Available tools

The configured default set of tools (what you would get by running pbench-register-tool-set) is:

- sar, iostat, mpstat, pidstat, proc-vmstat, proc-interrupts, perf

In addition, there are tools that can be added to the default set with pbench-register-tool:

- blktrace, cpuacct, dm-cache, docker, kvmstat, kvmtrace, lockstat, numastat, perf, porc-sched_debug, proc-vmstat, qemu-migrate, rabbit, strace, sysfs, systemtap, tcpdump, turbostat, virsh-migrate, vmstat

There is a default group of tools (that's what pbench-register-tool-set uses), but tools can be registered in other groups using the –group option of pbench-register-tool. The group can then be started and stopped using pbench-start-tools and pbench-stop-tools using their –group option.

Additional tools can be registered:

```
pbench-register-tool --name blktrace
```

or unregistered (e.g. some people prefer to run without perf):

```
pbench-unregister-tool --name perf
```

Note that perf is run in a "low overhead" mode with options "record -a –freq=100", but if you want to run it differently, you can always unregister it and register it again with different options:

```
pbench-unregister-tool --name=perf
pbench-register-tool --name=perf -- --record-opts="record -a --freq=200"
```

Tools can be also be registered, started and stopped on remote hosts (see the –remote option described in What does –remote do? in FAQ section.

1.6 Available benchmark scripts

Pbench provides a set of pre-packaged scripts to run some common benchmarks using the collection tools and other facilities that pbench provides. These are found in the bench-scripts directory of the Pbench installation (`/opt/pbench-agent/bench-scripts` by default). The current set includes:

- pbench-dbench
- pbench fio
- pbench-linpack
- pbench-migrate
- pbench-tpcc
- pbench-uperf
- pbench-user-benchmark (see *Running Pbench collection tools with an arbitrary benchmark* below for more on this)

You can run any of these with the –help option to get basic information about how to run the script. Most of these scripts accept a standard set of generic options, some semi-generic ones that are common to a bunch of benchmarks, as well as some benchmark specific options that vary from benchmark to benchmark.

The generic options are:

–help	show the set of options that the benchmark accepts.
–config	the name of the testing configuration (user specified).
–tool-group	the name of the tool group specifying the tools to run during execution of the benchmark.
–install	just install the benchmark (and any other needed packages) - do not run the benchmark.

The semi-generic ones are:

–test-types	the test types for the given benchmark - the values are benchmark-specific and can be obtained using –help.
–runtime	maximum runtime in seconds.
–clients	list of hostnames (or IPs) of systems that run the client (drive the test).
–samples	the number of samples per iteration.
–max-stddev	the percent maximum standard deviation allowed in order to consider the iteration to pass.
–max-failures	the maximum number of failures to achieve the allowed standard deviation.
–postprocess-only	
–run-dir	
–start-iteration-num	
–tool-label-pattern	

Benchmark-specific options are called out in the following sections for each benchmark.

Note that in some of these scripts the default tool group is hard-wired: if you want them to run a different tool group, you need to edit the script.

1.6.1 pbench-dbench

–threads

1.6.2 pbench-fio

Iterations are the cartesian product targets X test-types X block-sizes. More information on many of the following can be obtained from the fio man page.

–direct	O_DIRECT enabled or not (1/0) - default is 1.
–sync	O_SYNC enabled or not (1/0) - default is 0.
–rate-iops	IOP rate not to be exceeded (per job, per client)
–ramptime	seconds - time to warm up test before measurement.
–block-sizes	list of block sizes - default is 4, 64, 1024.
–file-size	fio will create files of this size during the job run.
–targets	file locations (list of directory/block device).
–job-mode	serial/concurrent - default is concurrent.
–ioengine	any IO engine that fio supports (see the fio man page) - default is psync.
–iodepth	number of I/O units to keep in flight against the file.
–client-file	file containing list of clients, one per line.
–numjobs	number of clones (processes/threads performing the same workload) of this job - default is 1.
–job-file	if you need to go beyond the recognized options, you can use a fio job file.

1.6.3 pbench-linpack

Note: TBD

1.6.4 pbench-migrate

Note: TBD

1.6.5 pbench-tpcc

Note: TBD

1.6.6 pbench-uperf

–kvm-host
–message-sizes
–protocols
–instances
–servers
–server-nodes
–client-nodes
–log-response-times

1.6.7 pbench-user-benchmark

Note: TBD

1.7 Utility Scripts

This section is needed as preparation for the *Second Steps* section below.

Pbench uses a bunch of utility scripts to do common operations. There is a common set of options for some of these: –name to specify a tool, –group to specify a tool group, –with-options to list or pass options to a tool, –remote to operate on a remote host (see entries in the FAQ section for more details on these options).

The first set is for registering and unregistering tools and getting some information about them:

Command	Description
pbench-list-tools	list the tools in the default group or in the specified group; with the –name option, list the groups that the named tool is in. TBD: how do you list all available tools whether in a group or not?
pbench-register-tool-set	call pbench-register-tool on each tool in the default list.
pbench-register-tool	add a tool to a tool group (possibly remotely).
pbench-unregister-tool (Obsolete)	remove a tool from a tool group (possibly remotely).
pbench-clear-tools	remove a tool or all tools from a specified tool group (including remotely). Used with a –name option, it replaces pbench -unregistered-tool.

The second set is for controlling the running of tools – pbench-start-tools and pbench-stop-tools, as well as pbench-postprocess- tools below, take –group, –dir and –iteration options: which group of tools to start/stop/postprocess, which directory to use to stash results and a label to apply to this set of results. pbench-kill-tools is used to make sure that all running tools are stopped: having a bunch of tools from earlier runs still running has been known to happen and is the cause of many problems (slowdowns in particular):

Command	Description
pbench-start-tools	start a group of tools, stashing the results in the directory specified by –dir.
pbench-stop-tools	stop a group of tools
pbench-kill-tools	make sure that no tools are running to pollute the environment.

The third set is for handling the results and doing cleanup:

Command	Description
pbench-postprocess-tools	run all the relevant postprocessing scripts on the tool output - this step also gathers up tool output from remote hosts to the local host in preparation for copying it to the results repository.
pbench-clear-results	start with a clean slate.
pbench-copy-results	copy results to the results repo.
pbench-move-results	move the results to the results repo and delete them from the local host.
pbench-edit-prefix	change the directory structure of the results (see the *Accessing results on the web* section below for details).
pbench-cleanup	clean up the pbench run directory - after this step, you will need to register any tools again.

pbench-register-tool-set, pbench-register-tool and pbench-unregister-tool can also take a –remote option (see What does –remote do?) in FAQ section in order to allow the starting/stopping of tools and the postprocessing of results on multiple remote hosts.

There is a set of miscellaneous tools for doing various and sundry things - although the name of the script indicates its purpose, if you want more information on these, you will have to read the code:

- pbench-avg-stddev
- pbench-log-timestamp

These are used by various pieces of Pbench. There is also a contrib directory that contains completely unsupported tools that various people have found useful.

1.8 Second Steps

Warning: It is highly recommended that you use one of the pbench-< benchmark> scripts for running your benchmark. If one does not exist already, you might be able to use the pbench-user-benchmark script to run your own script. The advantage is that these scripts already embody some conventions that Pbench and associated tools depend on, e.g. using a timestamp in the name of the results directory to make the name unique. If you cannot use pbench-user-benchmark and a pbench-< benchmark> script does not exist already, consider writing one or helping us write one. The more we can encapsulate all these details into generally useful tools, the easier it will be for everybody: people running it will not need to worry about all these details and people maintaining the system will not have to fix stuff because the script broke some assumptions. The easiest way to do so is to crib an existing pbench- script, e.g pbench-fio.

Once collection tools have been registered, the work flow of a benchmark script is as follows:

- Process options (see *Benchmark scripts options*).
- Check that the necessary prerequisites are installed and if not, install them.
- Iterate over some set of benchmark characteristics (e.g. pbench-fio iterates over a couple test types: read, randread and a bunch of block sizes), with each iteration doing the following:
  - create a benchmark_results directory
  - start the collection tools
  - run the benchmark
  - stop the collection tools
  - postprocess the collection tools data

The tools are started with an invocation of pbench-start-tools like this:

```
pbench-start-tools --group=$group --iteration=$iteration --dir=$benchmark_tools_dir
```

where the group is usually "default" but can be changed to taste as described above, iteration is a benchmark-specific tag that disambiguates the separate iterations in a run (e.g. for pbench-fio it is a combination of a count, the test type, the block size and a device name), and the benchmark_tools_dir specifies where the collection results are going to end up (see the section for much more detail on this).

The stop invocation is parallel, as is the postprocessing invocation:

```
pbench-stop-tools --group=$group --iteration=$iteration --dir=$benchmark_tools_dir
pbench-postprocess-tools --group=$group --iteration=$iteration --dir=$benchmark_tools_
↪dir
```

1.8.1 Benchmark scripts options

Generally speaking, benchmark scripts do not take any pbench-specific options except –config (see What does –config do? in FAQ section). Other options tend to be benchmark-specific.

1.8.2 Collection tools options

–help can be used to trigger the usage message on all of the tools (even though it's an invalid option for many of them). Here is a list of gotcha's:

- blktrace: you need to pass –devices=/dev/sda,/dev/sdb when you register the tool:

```
pbench-register-tool --name=blktrace [--remote=foo] -- --devices=/dev/sda,/dev/sdb
```

There is no default and leaving it empty causes errors in postprocessing (this should be flagged).

1.8.3 Utility script options

Note that pbench-move-results, pbench-copy-results and pbench-clear-results always assume that the run directory is the default `/var/lib/pbench-agent`.

pbench-move-results and pbench-copy-results now (starting with Pbench version 0.31-108gf016ed6) take a –prefix option. This is explained in the *Accessing results on the web* section below.

Note also that pbench-start/stop/postprocess-tools must be called with exactly the same arguments. The built-in benchmark scripts do that already, but if you go your own way, make sure to follow this dictum.

–dir

specify the run directory for all the collections tools. This argument **must** be used by `pbench-start/stop/postprocess-tools`, so that all the results files are in known places:

```
pbench-start-tools --dir=/var/lib/pbench-agent/foo
pbench-stop-tools --dir=/var/lib/pbench-agent/foo
pbench-postprocess-tools --dir=/var/lib/pbench-agent/foo
```

–remote

specify a remote host on which a collection tool (or set of collection tools) is to be registered:

```
pbench-register-tool --name=< tool> --remote=< host>
```

1.9 Running Pbench collection tools with an arbitrary benchmark

If you want to take advantage of Pbench's data collection and other goodies, but your benchmark is not part of the set above (see *Available benchmark scripts*), or you want to run it differently so that the pre-packaged script does not work for you, that's no problem (but, if possible, heed the WARNING above). The various Pbench phases can be run separately and you can fit your benchmark into the appropriate slot:

```
group=default
benchmark_tools_dir=TBD

pbench-register-tool-set --group=$group
pbench-start-tools --group=$group --iteration=$iteration --dir=$benchmark_tools_dir
< run your benchmark>
pbench-stop-tools --group=$group --iteration=$iteration --dir=$benchmark_tools_dir
pbench-postprocess-tools --group=$group --iteration=$iteration --dir=$benchmark_tools_
↪dir
pbench-copy-results
```

Often, multiple experiments (or "iterations") are run as part of a single run. The modified flow then looks like this:

```
group=default
experiments="exp1 exp2 exp3"
benchmark_tools_dir=TBD

pbench-register-tool-set --group=$group
for exp in $experiments ;do
     pbench-start-tools --group=$group --iteration=$exp
     < run the experiment>
     pbench-stop-tools --group=$group --iteration=$exp
     pbench-postprocess-tools --group=$group --iteration=$exp
done
pbench-copy-results
```

Alternatively, you may be able to use the pbench-user-benchmark script as follows:

```
pbench-user-benchmark --config="specjbb2005-4-JVMs" -- my_benchmark.sh
```

which is going to run my_benchmark.sh in the < run your benchmark> slot above. Iterations and such are your responsibility.

pbench-user-benchmark can also be used for a somewhat more specialized scenario: sometimes you just want to run the collection tools for a short time while your benchmark is running to get an idea of how the system looks. The idea here is to use pbench- user-benchmark to run a sleep of the appropriate duration in parallel with your benchmark:

```
pbench-user-benchmark --config="specjbb2005-4-JVMs" -- sleep 10
```

will start data collection, sleep for 10 seconds, then stop data collection and gather up the results. The config argument is a tag to distinguish this data collection from any other: you will probably want to make sure it's unique.

This works well for one-off scenarios, but for repeated usage on well defined phase changes you might want to investigate *Triggers*.

1.10 Remote hosts

1.10.1 Multihost benchmarks

Usually, a multihost benchmark is run using a host that acts as the "controller" of the run. There is a set of hosts on which data collection is to be performed while the benchmark is running. The controller may or may not be itself part of that set. In what follows, we assume that the controller has password-less ssh access to the relevant hosts.

The recommended way to run your workload is to use the generic pbench-user-benchmark script. The workflow in that case is:

- Register the collection tools on each host in the set:

```
for host in $hosts ;do
    pbench-register-tool-set --remote=$host
done
```

- Invoke pbench-user-benchmark with your workload generator as argument: that will start the collection tools on all the hosts and then run your workload generator; when that finishes, it will stop the collection tools on all the hosts and then run the postprocessing phase which will gather the data from all the remote hosts and run the postprocessing tools on everything.
- Run pbench-copy-results or pbench-move-results to upload the data to the results server.

If you cannot use the pbench-user-benchmark script, then the process becomes more manual. The workflow is:

- Register the collection tools on **each** host as above.
- Invoke pbench-start-tools on the controller: that will start data collection on all of the remote hosts.
- Run the workload generator.
- Invoke pbench-stop-tools on the controller: that will stop data collection on all of the remote hosts.
- Invoke pbench-postprocess-tools on the controller: that will gather all the data from the remotes and run the postprocessing tools on all the data.
- Run pbench-copy-results or pbench-move-results to upload the data to the results server.

1.11 Customizing

Some characteristics of Pbench are specified in config files and can be customized by adding your own config file to override the default settings. TBD

1.12 Results handling

1.12.1 Accessing results on the web

This section describes how to get to your results using a web browser. It describes how pbench-move-results moves the results from your local controller to a centralized location and what happens there. It also describes the –prefix option to pbench-move -results (and pbench-copy-results) and a utility script, pbench-edit-prefix, that allows you to change how the results are viewed.

1.12.2 Where to go to see results

Where pbench-move/copy-results copies the results is site-dependent. Check with the admin who set up the Pbench server and provided you with the configuration file for the pbench-agent installation.

1.13 Advanced topics

1.13.1 Triggers

Triggers are groups of tools that are started and stopped on specific events. They are registered with pbench-register-tool-trigger using the –start-trigger and –stop-trigger options. The output of the benchmark is piped into the pbench-tool-trigger tool which detects the conditions for starting and stopping the specified group of tools.

There are some commands specifically for triggers:

Command	Description
pbench-register-tool-trigger	register start and stop triggers for a tool group.
pbench-list-triggers	list triggers and their start/stop criteria.
pbench-tool-trigger	this is a Perl script that looks for the start-trigger and end-trigger markers in the benchmark's output, starting and stopping the appropriate group of tools when it finds the corresponding marker.

As an example, pbench-dbench uses three groups of tools: warmup, measurement and cleanup. It registers these groups as triggers using

```
pbench-register-tool-trigger --group=warmup --start-trigger="warmup" --stop-trigger=
↪"execute"
pbench-register-tool-trigger --group=measurement --start-trigger="execute" --stop-
↪trigger="cleanup"
pbench-register-tool-trigger --group=cleanup --start-trigger="cleanup" --stop-trigger=
↪"Operation"
```

It then pipes the output of the benchmark into pbench-tool-trigger:

```
$benchmark_bin --machine-readable --directory=$dir --timelimit=$runtime
       --warmup=$warmup --loadfile $loadfile $client |
        tee $benchmark_results_dir/result.txt |
        pbench-tool-trigger "$iteration" "$benchmark_results_dir" no
```

pbench-tool-trigger will then start the warmup group when it encounters the string "warmup" in the benchmark's output and stop it when it encounters "execute". It will also start the measurement group when it encounters "execute" and stop it when it encounters "cleanup" - and so on.

Obviously, the start/stop conditions will have to be chosen with some care to ensure correct actions.