
pbench Documentation

Pbench

Jun 09, 2023

PBENCH AGENT

1	Installation	3
1.1	Pbench containers	3
1.2	RPM based installation	3
1.3	Ansible based installation	3
2	User Guide	5
2.1	Getting Started	5
2.1.1	Installation	5
2.1.2	Tool Registration	5
2.1.3	Running a Benchmark	7
2.2	User Guide	10
2.2.1	What is Pbench?	11
2.2.2	TL;DR - How to set up Pbench and run a benchmark	11
2.2.3	How to install	12
2.2.4	Defaults	12
2.2.5	Available tools	12
2.2.6	Available benchmark scripts	13
2.2.6.1	pbench-fio	14
2.2.6.2	pbench-linpack	14
2.2.6.3	pbench-specjbb2005	14
2.2.6.4	pbench-uperf	14
2.2.6.5	pbench-user-benchmark	15
2.2.7	Utility Scripts	15
2.2.8	Second Steps	17
2.2.8.1	Benchmark scripts options	18
2.2.8.2	Collection tools options	18
2.2.8.3	Utility script options	18
2.2.9	Running Pbench collection tools with an arbitrary benchmark	18
2.2.10	Remote hosts	19
2.2.10.1	Multihost benchmarks	19
2.2.11	Customizing	20
2.2.12	Results handling	20
2.2.12.1	Accessing results on the web	20
2.2.12.2	Where to go to see results	20
2.2.13	Advanced topics	21
2.2.13.1	Triggers	21
2.3	Man pages	21
2.3.1	Commands by functional group	21
2.3.1.1	Performance tool management commands	21
2.3.1.2	Benchmark commands	22

2.3.1.3	Upload to Pbench Server	22
2.3.2	Commands	22
2.3.2.1	pbench-clear-results	22
2.3.2.2	pbench-clear-tools	22
2.3.2.3	pbench-copy-results	23
2.3.2.4	pbench-list-tools	24
2.3.2.5	pbench-list-triggers	24
2.3.2.6	pbench-move-results	25
2.3.2.7	pbench-register-tool	25
2.3.2.8	pbench-register-tool-set	27
2.3.2.9	pbench-register-tool-trigger	28
2.3.2.10	pbench-results-move	29
2.3.2.11	pbench-user-benchmark	29
3	FAQ	31
4	Pbench Server API documentation	33
5	Pbench Dashboard	35
6	FAQ	37
7	Guidelines for Contributing to Pbench	39
7.1	1. Forking the repository:	39
7.2	2. Cloning	39
7.3	3. Choosing an issue to work upon	39
7.4	Making changes to the codebase	39
7.5	Add, Commit and Push	40
7.6	Conventions on commits, PRs, and overall git best practices.	40
7.7	Opening a pull request	40
7.8	Creating an Issue	41
7.9	Reviewing a pull request	41

Pbench is 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.

INSTALLATION

Choose any one of the following approaches to setup *Pbench Agent*

1.1 Pbench containers

1.2 RPM based installation

1.3 Ansible based installation

2.1 Getting Started

The following is an introduction on how to use the pbench agent.

Pbench can be used to either automate tool execution and postprocessing for you, or also run any of its built-in benchmark scripts. This first test will run the fio benchmark.

2.1.1 Installation

If you have not done so, install `pbench-agent` (via RPM or other Linux distribution supported method, documented in `INSTALL` file).

After `pbench-agent` is installed, verify that your path includes:

```
/opt/pbench-agent:/opt/pbench-agent/util-scripts:/opt/pbench-agent/bench-scripts
```

If you do not have this, you may need to source your `.bashrc`, re-log in, or just run, `. /opt/pbench-agent/profile` to have the path updated.

2.1.2 Tool Registration

After you are certain the path is updated, register the default set of tools:

```
register-tool-set
```

This command will register the default tool set, which consists of `sar`, `mpstat`, `iostat`, `pidstat`, `proc-vmstat`, `proc-interrupts`, and `perf`.

When registering these tools, `pbench-agent` checks if they are installed and may install some of them if they are not present. Some of these tools are built from source, so you may see output from fetching the source and compiling. Following any installation, you should have this output:

```
sar tool is now registered in group default
[debug]tool_opts: default --interval="3"
[debug]checking to see if tool is installed...
iostat is installed
iostat tool is now registered in group default
[debug]tool_opts: default --interval="3"
[debug]checking to see if tool is installed...
mpstat is installed
```

(continues on next page)

(continued from previous page)

```
mpstat tool is now registered in group default
[debug]tool_opts: default --interval="3"
[debug]checking to see if tool is installed...
pidstat is installed
pidstat tool is now registered in group default
[debug]tool_opts: default --interval="3"
[debug]checking to see if tool is installed...
proc-vmstat tool is now registered in group default
[debug]tool_opts: default --interval="3"
[debug]checking to see if tool is installed...
proc-interrupts tool is now registered in group default
[debug]tool_opts: default --record-opts="record -a --freq=100"
[debug]checking to see if tool is installed...
perf tool is now registered in group default
```

If at any time you are unsure which tools are registered, you can run:

```
# list-tools
default: perf,proc-interrupts,proc-vmstat,pidstat,mpstat,iostat,sar
```

The output above shows which tools are in the “default” tool group. And by specifying the `--with-options` switch, you get the options used for these tools:

```
# list-tools --with-options
default: perf --record-opts="record -a --freq=100",proc-interrupts --interval="3",
proc-vmstat --interval="3",pidstat --interval="3",mpstat --interval="3",iostat --
↪interval="3",sar --interval="3"
```

In the above example, the `--interval` option is set for all tools but `perf`. Optionally, you can change these individually with the `register-tool` command:

```
# register-tool --name=pidstat -- --interval=10
[debug]tool_opts: --interval="10"
[debug]checking to see if tool is installed...
pidstat is installed
pidstat tool is now registered in group default
```

Then run `list-tools --with-options` again to confirm:

```
# list-tools --with-options
default: pidstat --interval="10",perf --record-opts="record -a --freq=100",
proc-interrupts --interval="3",proc-vmstat --interval="3",mpstat --interval="3",iostat --
↪interval="3",sar --interval="3"
```

And the interval for `pidstat` is now 10.

2.1.3 Running a Benchmark

OK, now that the tools are registered, it's time to run the benchmark. We'll use the `fio` benchmark for this example. To run, simply type `'pbench_fio'`, the wrapper script `pbench-agent` provides for the `fio` benchmark.

If this is the first time running `fio` via the `pbench-agent`, `pbench-agent` will attempt to download and compile `fio`. You may see quite a bit of output from this. Once `fio` is installed, `pbench-agent` will run several tests by default. Output for each will look something like this:

```
about to run fio read with 4 block size on /tmp/fio
-----fio will use this job file:-----
[global]
bs=4k
ioengine=libaio
iodepth=32
direct=1
time_based=1
runtime=30
[job1]
rw=read
filename=/tmp/fio
size=896M
-----
```

Right before the `pbench_fio` script starts a `fio` job, it will call `start-tools`, which will produce output like this:

```
[debug][start-tools]/opt/pbench-agent/tool-scripts/sar --start --iteration=1-read-4KiB --
↪group=default --dir=/var/lib/pbench/fio__2014-09-11_12:54:42/1-read-4KiB/tools-default-
↪default --interval="3"
[debug][start-tools]/opt/pbench-agent/tool-scripts/iostat --start --iteration=1-read-
↪4KiB --group=default --dir=/var/lib/pbench/fio__2014-09-11_12:54:42/1-read-4KiB/tools-
↪default default --interval="3"
[debug][start-tools]/opt/pbench-agent/tool-scripts/mpstat --start --iteration=1-read-
↪4KiB --group=default --dir=/var/lib/pbench/fio__2014-09-11_12:54:42/1-read-4KiB/tools-
↪default default --interval="3"
[debug][start-tools]/opt/pbench-agent/tool-scripts/pidstat --start --iteration=1-read-
↪4KiB --group=default --dir=/var/lib/pbench/fio__2014-09-11_12:54:42/1-read-4KiB/tools-
↪default default --interval="3"
[debug][start-tools]/opt/pbench-agent/tool-scripts/proc-vmstat --start --iteration=1-
↪read-4KiB --group=default --dir=/var/lib/pbench/fio__2014-09-11_12:54:42/1-read-4KiB/
↪tools-default default --interval="3"
[debug][start-tools]/opt/pbench-agent/tool-scripts/proc-interrupts --start --iteration=1-
↪read-4KiB --group=default --dir=/var/lib/pbench/fio__2014-09-11_12:54:42/1-read-4KiB/
↪tools-default default --interval="3"
[debug][start-tools]/opt/pbench-agent/tool-scripts/perf --start --iteration=1-read-4KiB -
↪group=default --dir=/var/lib/pbench/fio__2014-09-11_12:54:42/1-read-4KiB/tools-
↪default default --record-opts="record -a --freq=100"
```

That is output from `start-tools` starting all of the tools that were registered.

Next is the output from the actual `fio` job:

```
fio: Going to run [/usr/local/bin/fio /var/lib/pbench/fio__2014-09-11_12:54:42/1-read-
↪4KiB/fio.job]
job1: (g=0): rw=read, bs=4K-4K/4K-4K/4K-4K, ioengine=libaio, iodepth=32
```

(continues on next page)

(continued from previous page)

```

fio-2.1.7
Starting 1 process
job1: Laying out IO file(s) (1 file(s) / 896MB)

job1: (groupid=0, jobs=1): err= 0: pid=12961: Thu Sep 11 12:55:47 2014
  read : io=1967.4MB, bw=67147KB/s, iops=16786, runt= 30003msec
    slat (usec): min=3, max=77, avg= 7.95, stdev= 2.45
    clat (msec): min=1, max=192, avg= 1.90, stdev= 1.48
      lat (msec): min=1, max=192, avg= 1.90, stdev= 1.48
    clat percentiles (usec):
      | 1.00th=[ 1736],  5.00th=[ 1736], 10.00th=[ 1752], 20.00th=[ 1752],
      | 30.00th=[ 1768], 40.00th=[ 1768], 50.00th=[ 1768], 60.00th=[ 1912],
      | 70.00th=[ 1912], 80.00th=[ 2064], 90.00th=[ 2096], 95.00th=[ 2224],
      | 99.00th=[ 2256], 99.50th=[ 2256], 99.90th=[10304], 99.95th=[10816],
      | 99.99th=[44800]
    bw (KB /s): min=34373, max=70176, per=100.00%, avg=67211.32, stdev=5212.44
    lat (msec) : 2=78.09%, 4=21.73%, 10=0.05%, 20=0.10%, 50=0.01%
    lat (msec) : 100=0.01%, 250=0.01%
  cpu          : usr=5.97%, sys=22.23%, ctx=501089, majf=0, minf=332
  IO depths    : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=0.1%, 32=100.0%, >=64=0.0%
    submit     : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
    complete   : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.1%, 64=0.0%, >=64=0.0%
    issued    : total=r=503651/w=0/d=0, short=r=0/w=0/d=0
    latency   : target=0, window=0, percentile=100.00%, depth=32

Run status group 0 (all jobs):
  READ: io=1967.4MB, aggrb=67146KB/s, minb=67146KB/s, maxb=67146KB/s, mint=30003msec,
  ↳maxt=30003msec

Disk stats (read/write):
  dm-1: ios=501328/154, merge=0/0, ticks=947625/12780, in_queue=960429, util=99.53%,
  ↳aggrios=503626/101, aggrmerge=25/55, aggrticks=949096/9541, aggrin_queue=958491,
  ↳aggrutil=99.49%
  sda: ios=503626/101, merge=25/55, ticks=949096/9541, in_queue=958491, util=99.49%

```

Now that this fio job is complete, the pbench_fio script calls stop-tools:

```

[debug][stop-tools]/opt/pbench-agent/tool-scripts/sar --stop --iteration=1-read-4KiB --
↳group=default --dir=/var/lib/pbench/fio__2014-09-11_12:54:42/1-read-4KiB/tools-default_
↳default --interval="3"
[debug][stop-tools]/opt/pbench-agent/tool-scripts/iostat --stop --iteration=1-read-4KiB -
↳group=default --dir=/var/lib/pbench/fio__2014-09-11_12:54:42/1-read-4KiB/tools-
↳default default --interval="3"
[debug]stopping sar
[debug][stop-tools]/opt/pbench-agent/tool-scripts/mpstat --stop --iteration=1-read-4KiB -
↳group=default --dir=/var/lib/pbench/fio__2014-09-11_12:54:42/1-read-4KiB/tools-
↳default default --interval="3"
[debug]stopping iostat
[debug][stop-tools]/opt/pbench-agent/tool-scripts/pidstat --stop --iteration=1-read-4KiB_
↳--group=default --dir=/var/lib/pbench/fio__2014-09-11_12:54:42/1-read-4KiB/tools-
↳default default --interval="3"
[debug]stopping mpstat

```

(continues on next page)

(continued from previous page)

```
[debug][stop-tools]/opt/pbench-agent/tool-scripts/proc-vmstat --stop --iteration=1-read-
↪4KiB --group=default --dir=/var/lib/pbench/fio__2014-09-11_12:54:42/1-read-4KiB/tools-
↪default default --interval="3"
[debug]stopping pidstat
[debug][stop-tools]/opt/pbench-agent/tool-scripts/proc-interrupts --stop --iteration=1-
↪read-4KiB --group=default --dir=/var/lib/pbench/fio__2014-09-11_12:54:42/1-read-4KiB/
↪tools-default default --interval="3"
[debug]stopping proc-vmstat
[debug][stop-tools]/opt/pbench-agent/tool-scripts/perf --stop --iteration=1-read-4KiB --
↪group=default --dir=/var/lib/pbench/fio__2014-09-11_12:54:42/1-read-4KiB/tools-default_
↪default --record-opts="record -a --freq=100"
[debug]stopping proc-interrupts
waiting for PID 12934 (perf) to finish
```

Next, `pbench_fio` calls `postprocess-tools`. This is what generates the `.csv` files and renders the `.html` file containing the NVD3 graphs for the tool data.

```
collecting /proc
collecting /sys
[debug][postprocess-tools]/opt/pbench-agent/tool-scripts/sar --postprocess --iteration=1-
↪read-4KiB --group=default --dir=/var/lib/pbench/fio__2014-09-11_12:54:42/1-read-4KiB/
↪tools-default default --interval="3"
[debug][postprocess-tools]/opt/pbench-agent/tool-scripts/iostat --postprocess --
↪iteration=1-read-4KiB --group=default --dir=/var/lib/pbench/fio__2014-09-11_12:54:42/1-
↪read-4KiB/tools-default default --interval="3"
[debug][postprocess-tools]/opt/pbench-agent/tool-scripts/mpstat --postprocess --
↪iteration=1-read-4KiB --group=default --dir=/var/lib/pbench/fio__2014-09-11_12:54:42/1-
↪read-4KiB/tools-default default --interval="3"
[debug]postprocessing iostat
[debug][postprocess-tools]/opt/pbench-agent/tool-scripts/pidstat --postprocess --
↪iteration=1-read-4KiB --group=default --dir=/var/lib/pbench/fio__2014-09-11_12:54:42/1-
↪read-4KiB/tools-default default --interval="3"
[debug]postprocessing mpstat
[debug][postprocess-tools]/opt/pbench-agent/tool-scripts/proc-vmstat --postprocess --
↪iteration=1-read-4KiB --group=default --dir=/var/lib/pbench/fio__2014-09-11_12:54:42/1-
↪read-4KiB/tools-default default --interval="3"
[debug]postprocessing pidstat
[debug][postprocess-tools]/opt/pbench-agent/tool-scripts/proc-interrupts --postprocess --
↪iteration=1-read-4KiB --group=default --dir=/var/lib/pbench/fio__2014-09-11_12:54:42/1-
↪read-4KiB/tools-default default --interval="3"
[debug]postprocessing proc-vmstat
[debug][postprocess-tools]/opt/pbench-agent/tool-scripts/perf --postprocess --
↪iteration=1-read-4KiB --group=default --dir=/var/lib/pbench/fio__2014-09-11_12:54:42/1-
↪read-4KiB/tools-default default --record-opts="record -a --freq=100"
[debug]postprocessing proc-interrupts
```

This will repeat for a total of 6 different `fio` jobs, then the `fio` benchmark will be complete. Now that the job is complete, we want to move the results to the archive host. The results are currently in `/var/lib/pbench/fio-`. To move these results, simply run:

```
# move-results
```

Once that command completes, the data should be moved to the configured archive host. To view your results, use

a link like this in your browser (replacing the “resultshost.example.com” with your pbench deployed web server, and replacing the “your-HOSTNAME” with the \$(hostname -s) of the machine where you issued the “move-results” above):

`http://resultshost.example.com/results//?C=M;O=D`

Towards the top of the list, there should be a directory like “`fio__2014-09-11_12:54:42`”. That is your pbench fio job. Click on that directory to see the results.

There should be a file, `fio-summary.txt`, which will contain the results for all of the fio jobs that were run.

In this same directory, there should be more sub-directories, one for each fio job. They should have a format like “`N-[read|write]-MKiB`”. In pbench-speak, these are called an “iteration” and usually start with “1-”. Under each of these you will find the details of that job/iteration:

- `fio.cmd`: the actual fio command used
- `fio.job`: the job file `pbench_fio` created
- `result.txt`: the output from the fio job
- `tool-default`: all of the collected tool data
- `sysinfo`: data `pbench_fio` collected from `/sys` & `/proc`

Under the `tools-default` directory, there should be text output for each tool as well as `.html` files, and a `csv` sub-directory containing all of the raw tool data.

2.2 User Guide

Contents

- *User Guide*
 - *What is Pbench?*
 - *TL;DR - How to set up Pbench and run a benchmark*
 - *How to install*
 - *Defaults*
 - *Available tools*
 - *Available benchmark scripts*
 - * *pbench-fio*
 - * *pbench-linpack*
 - * *pbench-specjbb2005*
 - * *pbench-uperf*
 - * *pbench-user-benchmark*
 - *Utility Scripts*
 - *Second Steps*
 - * *Benchmark scripts options*
 - * *Collection tools options*
 - * *Utility script options*

- *Running Pbench collection tools with an arbitrary benchmark*
- *Remote hosts*
 - * *Multihost benchmarks*
- *Customizing*
- *Results handling*
 - * *Accessing results on the web*
 - * *Where to go to see results*
- *Advanced topics*
 - * *Triggers*

2.2.1 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!

2.2.2 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 “`my-host`”, 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.

2.2.3 How to install

See the [install](#) section for details.

2.2.4 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.

2.2.5 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).

2.2.6 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 fio
- pbench-linpack
- pbench-specjbb2005
- 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:

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

The semi-generic ones are:

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

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.

2.2.6.1 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.
-unique-ports	use different ports for each client (needed if e.g. multiple clients on one system)

2.2.6.2 pbench-linpack

Note: TBD

2.2.6.3 pbench-specjbb2005

Note: TBD

2.2.6.4 pbench-uperf

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

2.2.6.5 pbench-user-benchmark

Note: TBD

2.2.7 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 <code>--name</code> 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 <code>--name</code> option, it replaces pbench <code>--unregistered-tool</code> .

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 <code>--dir</code> .
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-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.

2.2.8 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
```

2.2.8.1 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.

2.2.8.2 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).

2.2.8.3 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>
```

2.2.9 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
```

(continues on next page)

(continued from previous page)

```
< 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*.

2.2.10 Remote hosts

2.2.10.1 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.

2.2.11 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

2.2.12 Results handling

2.2.12.1 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.

2.2.12.2 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.

2.2.13 Advanced topics

2.2.13.1 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
<code>pbench-register-tool-trigger</code>	register start and stop triggers for a tool group.
<code>pbench-list-triggers</code>	list triggers and their start/stop criteria.
<code>pbench-tool-trigger</code>	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.

2.3 Man pages

2.3.1 Commands by functional group

2.3.1.1 Performance tool management commands

- *pbench-clear-results*
- *pbench-clear-tools*
- *pbench-list-tools*
- *pbench-list-triggers*
- *pbench-register-tool*
- *pbench-register-tool-set*
- *pbench-register-tool-trigger*

2.3.1.2 Benchmark commands

- *pbench-user-benchmark*

2.3.1.3 Upload to Pbench Server

Pbench Server 0.69

- *pbench-move-results*
- *pbench-copy-results*

Pbench Server 1.0

- *pbench-results-move*

2.3.2 Commands

2.3.2.1 pbench-clear-results

NAME

`pbench-clear-results` - clears the result directory

SYNOPSIS

`pbench-clear-results` [OPTIONS]

DESCRIPTION

This command clears the results directory from `/var/lib/pbench-agent` directory.

OPTIONS

`-C, --config PATH`

Path to the Pbench Agent configuration file. This option is required if not provided by the `_PBENCH_AGENT_CONFIG` environment variable.

`--help`

Show this message and exit.

2.3.2.2 pbench-clear-tools

NAME

`pbench-clear-tools` - clear registered tools by name or group

SYNOPSIS

`pbench-clear-tools` [OPTIONS]

DESCRIPTION

Clear all tools which are registered and can filter by name of the group.

OPTIONS

-C, --config PATH

Path to the Pbench Agent configuration file. This option is required if not provided by the `_PBENCH_AGENT_CONFIG` environment variable.

-n, --name, --names <name>

Clear only the <name> tool.

-g, --group, --groups <group>

Clear the tools in the <group>. If no group is specified, the default group is assumed.

-r, --remote, --remotes STR[,STR...]

Clear the tool(s) only on the specified remote(s). Multiple remotes may be specified as a comma-separated list. If no remote is specified, all remotes are cleared.

--help

Show this message and exit.

2.3.2.3 pbench-copy-results

NAME

pbench-copy-results - copy result tarball to the 0.69 Pbench Server

SYNOPSIS

pbench-copy-results --user=<user> [OPTIONS]

DESCRIPTION

Push the benchmark result to the Pbench Server without removing it from the local host. This command requires `/opt/pbench-agent/id_rsa` file with the private SSH key, when pushing to a 0.69 Pbench Server.

OPTIONS

--user <user>

This option value is required if not provided by the `PBENCH_USER` environment variable; otherwise, a value provided on the command line will override any value provided by the environment.

--controller <controller>

This option may be used to override the value provided by the `PBENCH_CONTROLLER` environment variable; if neither value is available, the result of `hostname -f` is used. (If no value is available, the command will exit with an error.)

--prefix <prefix>

This option allows the user to specify an optional directory-path hierarchy to be used when displaying the result files on the 0.69 Pbench Server.

--show-server

This will not move any results but will resolve and then display the pbench server destination for results.

--xz-single-threaded

This will force the use of a single thread for locally compressing the result files.

--help

Show this message and exit.

2.3.2.4 pbench-list-tools

NAME

pbench-list-tools - list all the registered tools optionally filtered by name or group

SYNOPSIS

pbench-list-tools [OPTIONS]

DESCRIPTION

List tool registrations, optionally filtered by tool name or tool group.

OPTIONS

-C, --config PATH

Path to the Pbench Agent configuration file. This option is required if not provided by the `_PBENCH_AGENT_CONFIG` environment variable.

-n, --name <name>

List the tool groups in which tool <name> is registered.

-g, --group <group>

List all the tools registered in the <group>.

-o, --with-option

List the options with each tool.

--help

Show this message and exit.

2.3.2.5 pbench-list-triggers

NAME

pbench-list-triggers - list the registered triggers by group

SYNOPSIS

pbench-list-triggers [OPTIONS]

DESCRIPTION

This command will list all the registered triggers by `group-name`.

OPTIONS

-C, --config PATH

Path to the Pbench Agent configuration file. This option is required if not provided by the `_PBENCH_AGENT_CONFIG` environment variable.

-g, --group, --groups <group>

List all the triggers registered in the <group>.

--help

Show this message and exit.

2.3.2.6 pbench-move-results

NAME

pbench-move-results - move all results to 0.69 Pbench Server

SYNOPSIS

pbench-move-results [OPTIONS]

DESCRIPTION

Push the benchmark result to the 0.69 Pbench Server. Requires a `/opt/pbench-agent/id_rsa` file with the private SSH key of the server's pbench account. On a successful push, this command removes the results from the local host.

OPTIONS

`--user <user>`

This option value is required if not provided by the `PBENCH_USER` environment variable; otherwise, a value provided on the command line will override any value provided by the environment.

`--controller <controller>`

This option may be used to override the value provided by the `PBENCH_CONTROLLER` environment variable; if neither value is available, the result of `hostname -f` is used. (If no value is available, the command will exit with an error.)

`--prefix <prefix>`

This option allows the user to specify an optional directory-path hierarchy to be used when displaying the result tar balls on the pbench server.

`--show-server`

This will not move any results but will resolve and then display the pbench server destination for results.

`--xz-single-threaded`

This will force the use of a single thread for locally compressing the result files.

`--help`

Show this message and exit.

2.3.2.7 pbench-register-tool

NAME

pbench-register-tool - registers the specified tool

SYNOPSIS

pbench-register-tool --name=<tool-name> [OPTIONS] [-- <tool-specific-options>]

DESCRIPTION

Register the specified tool.

List of available tools:

Transient

- blktrace
- bpftrace
- cpuacct
- disk

- dm-cache
- docker
- docker-info
- external-data-source
- haproxy-ocp
- iostat
- jmap
- jstack
- kvm-spinlock
- kvmstat
- kvmtrace
- lockstat
- mpstat
- numastat
- oc
- openvswitch
- pcp-transient
- perf
- pidstat
- pprof
- proc-interrupts
- proc-sched_debug
- proc-vmstat
- prometheus-metrics
- qemu-migrate
- rabbit
- sar
- strace
- sysfs
- systemtap
- tcpdump
- turbostat
- user-tool
- virsh-migrate
- vmstat

Persistent

- node-exporter
- dcgm
- pcp

For a list of tool-specific options, run:

```
/opt/pbench-agent/tool-scripts/<tool-name> --help
```

OPTIONS

`--name <tool-name>`

`<tool-name>` specifies the name of the tool to be registered.

`-g, --group, --groups <group>`

Register the tool in `<group>`. If no group is specified, the default group is assumed.

`[--persistent | --transient]`

For tools which can be run as either “transient” (where they are started and stopped on each iteration) or as “persistent” (where they are started before the first iteration and run continuously over all iterations), these options determine how the tool will be run. (Most tools are configured to run as either one the other, so these options are not necessary in those cases, and specifying the wrong one will produce an error.)

`--no-install`

[To be supplied]

`--labels=<label>[,<label>]]`

Where the list of labels must match the list of remotes.

`-remotes STR[,STR] ...` Single remote host, a list of remote hosts (comma-separated, no spaces) or an “at” sign (@) followed by a filename. In this last case, the file should contain a list of hosts and their (optional) labels. Each line of the file should contain a hostname, optionally followed by a label separated by a comma (,); empty lines are ignored, and comments are denoted by a leading hash (#), character.

`--help`

Show this message and exit.

2.3.2.8 pbench-register-tool-set

NAME

`pbench-register-tool-set` - register the specified toolset

SYNOPSIS

`pbench-register-tool-set [OPTIONS] <tool-set>`

DESCRIPTION

Register all the tools in the specified toolset.

Available `<tool-set>` from `/opt/pbench-agent/config/pbench-agent.cfg`:

- heavy
- legacy
- light
- medium

OPTIONS

-remotes STR[, STR] . . . Single remote host, a list of remote hosts (comma-separated, no spaces) or an “at” sign (@) followed by a filename. In this last case, the file should contain a list of hosts and their (optional) labels. Each line of the file should contain a hostname, optionally followed by a label separated by a comma (,); empty lines are ignored, and comments are denoted by a leading hash (#), character.

-g, --group <group>

Register the toolset in <group>. If no group is specified, the default group is assumed.

--labels=<label>[, <label>]

Where the list of labels must match the list of remotes.

--interval=<INT>

[To be supplied]

--no-install

[To be supplied]

--help

Show this message and exit.

2.3.2.9 pbench-register-tool-trigger

NAME

pbench-register-tool-trigger - register the tool trigger

SYNOPSIS

pbench-register-tool-trigger [OPTIONS]

DESCRIPTION

Register triggers which start and stop data collection for the given tool group.

OPTIONS

-C, --config PATH

Path to the Pbench Agent configuration file. This option is required if not provided by the `_PBENCH_AGENT_CONFIG` environment variable.

-g, --group, --groups <group>

Registers the trigger in the <group>. If no group is specified, the default group is assumed.

--start-trigger STR

[To be supplied]

--stop-trigger STR

[To be supplied]

--help

Show this message and exit.

2.3.2.10 pbench-results-move

NAME

pbench-results-move - move results directories to the server to a 1.0 Pbench Server

SYNOPSIS

pbench-results-move [OPTIONS]

DESCRIPTION

This command uploads one or more result directories to the configured v1.0 Pbench Server. The specified API Key is used to authenticate the user and to establish ownership of the data on the server. Once the upload is complete, the result directories are, by default, removed from the local system.

OPTIONS

-C, --config PATH

Path to the Pbench Agent configuration file. This option is required if not provided by the `_PBENCH_AGENT_CONFIG` environment variable.

--controller <controller>

Override the default controller name.

--token <token>

Pbench Server API key [required].

--delete | --no-delete

Remove local data after successful copy [default: delete]

--xz-single-threaded

Use single-threaded compression with xz.

--help

Show this message and exit.

2.3.2.11 pbench-user-benchmark

NAME

pbench-user-benchmark - run a workload and collect performance data

SYNOPSIS

pbench-user-benchmark [OPTIONS] -- <command-to-run>

DESCRIPTION

Collects data from the registered tools while running a user-specified action. This can be a specific synthetic benchmark workload, a real workload, or simply a delay to measure system activity.

Invoking `pbench-user-benchmark` with a workload generator as an argument will perform the following steps:

- Start the collection tools on all the hosts.
- Execute the workload generator.
- Stop the collection tools on all the hosts.
- Gather the data from all the remote hosts and generate a `result.txt` file by running the tools' post-processing on the collected data.

<command-to-run> A script, executable, or shell command to run while gathering tool data. Use -- to stop processing of pbench-user-benchmark options if your command includes options, like

```
pbench-user-benchmark --config string -- fio --bs 16k
```

OPTIONS

-C, --config PATH

Path to the Pbench Agent configuration file. This option is required if not provided by the `_PBENCH_AGENT_CONFIG` environment variable.

--tool-group STR

The tool group to use for data collection.

--iteration-list STR

A file containing a list of iterations to run for the provided script; the file should contain one iteration per line. With a leading hash (#) character used for comments and blank lines are ignored. Each iteration line should use alpha-numeric characters before the first space to name the iteration, with the rest of the line provided as arguments to the script.

NOTE: --iteration-list is not compatible with --use-tool-triggers

--sysinfo STR[,STR...]

Comma-separated values of system information to be collected; available: default, none, all, ara, block, insights, kernel_config, libvirt, security_mitigations, sos, stockpile, topology

--pbench-pre STR

Path to the script which will be executed before tools are started.

NOTE: --pbench-pre is not compatible with --use-tool-triggers

--pbench-post STR

Path to the script which will be executed after tools are stopped and postprocessing is complete.

NOTE: --pbench-post is not compatible with --use-tool-triggers

--use-tool-triggers

Use tool triggers instead of normal start/stop around script.

NOTE: --use-tool-triggers is not compatible with --iteration-list, --pbench-pre, or --pbench-post

--no-stderr-capture

Do not capture the standard error output of the script in the `result.txt` file

--help

Show this message and exit.

CHAPTER
THREE

FAQ

PBENCH SERVER API DOCUMENTATION

The Pbench Server API provides the interface to Pbench data for use by the UI dashboard as well as any other web clients.

The Pbench Server provides a set of HTTP endpoints to manage user authentication and curated performance information, called “dataset resources” or just “datasets”.

The V1 API provides a REST-like functional interface.

The Pbench Server APIs accept parameters from a variety of sources. See the individual API documentation for details.

1. Some parameters, especially “resource ids”, are embedded in the URI, such as `/api/v1/datasets/<resource_id>`;
2. Some parameters are passed as query parameters, such as `/api/v1/datasets?name:fio`;
3. For PUT and POST APIs, parameters may also be passed as a JSON (`application/json` content type) request payload, such as `{"metadata": {"dataset.name": "new name"}}`

PBENCH DASHBOARD

Pbench Dashboard is the web-based platform for consuming indexed performance benchmark data. It provides data curation capabilities for the performance datasets.

The landing page is the browsing page where the user can view the list of public datasets. Those datasets can be filtered based on name and/or uploaded time.

The screenshot shows the Pbench Dashboard interface. At the top is a dark header with the Pbench logo, a menu icon, and a 'Login' button. Below the header is a light blue banner with the text 'Want to see your own data? Login or Create an account'. The main content area is titled 'Results' and includes a search bar, a date filter (YYYY-MM-DD to YYYY-MM-DD), and an 'Update' button. Below the filter is a table with two tabs: 'All Results(5)' and 'Favorites(0)'. The table lists five benchmark datasets with columns for Name, Uploaded On, and a star icon. The datasets are: fio_rw_2018.02.01T22.40.57, linpack_mock_2020.02.28T19.10.55, pbench-user-benchmark_example-vmstat_2018.10.24T14.38.18, uperf_rhel8.1.4.18.0-107e18_snap4_25gb_virt_2019.06.21T01.28.57, and trafficgen_basic-forwarding-example_tg:trex-profile_pf:forwarding_test:json_ml5_tt:bs_2019-08-27T14.58.38. At the bottom right of the table is a pagination control showing '1 - 5 of 5' and '1 of 1'.

Name	Uploaded On	Star
fio_rw_2018.02.01T22.40.57	2023-05-23T11:01:39.711761+00:00	★
linpack_mock_2020.02.28T19.10.55	2023-05-23T11:01:39.845130+00:00	★
pbench-user-benchmark_example-vmstat_2018.10.24T14.38.18	2023-05-23T11:01:39.909414+00:00	★
uperf_rhel8.1.4.18.0-107e18_snap4_25gb_virt_2019.06.21T01.28.57	2023-05-23T11:01:40.214264+00:00	★
trafficgen_basic-forwarding-example_tg:trex-profile_pf:forwarding_test:json_ml5_tt:bs_2019-08-27T14.58.38	2023-05-23T11:01:39.991406+00:00	★

Login button can be found on the right side of the Header. Clicking on it will redirect the browser to the login page.

On logging in, the user can view the Overview Page which is the data curation page. It has three components.

- *New and Unmanaged Runs* shows the newly created runs which can be saved
- *Saved Runs* lists the saved runs which can be published to share with others
- *Expiring Runs* lists the saved runs which will be deleted from the server within the next 20 days

GUIDELINES FOR CONTRIBUTING TO PBENCH

7.1 1. Forking the repository:

Forking a repository allows you to freely experiment with changes without affecting the original project. Most commonly, forks are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea.

7.2 2. Cloning

Cloning is used to create a local copy of the repository. It takes only one command in the terminal to clone the repository.

```
git clone https://github.com/distributed-system-analysis/pbench.git
```

7.3 3. Choosing an issue to work upon

1. Go to the issues section, to find a list of open issues.
1. Select the issues you are interested to work upon based upon the labels and descriptions.
1. It is a good practice to assign the issue to yourself to let others know you're working upon it.

7.4 Making changes to the codebase

- Follow the instructions in the README.md to setup and install pbench.
- Save your changes by creating your own local branches on git

7.5 Add, Commit and Push

- Follow these commands to push the changes to your branch.

```
git add .  
git commit -m "Issue solved"  
git push origin branch_name
```

7.6 Conventions on commits, PRs, and overall git best practices.

- Commit messages should have a short description (50 - 70 characters) followed by a longer format description of the changes below if needed. You'll also notice each line is formatted for a specific length in the longer format description. For example:

```
Extend auditing to incoming, results, and users
```

```
The server audit is now applied to the incoming, results, and users  
directory hierarchies. Any unpacked tar ball should now be compre-  
hensively checked to see that all is in the correct place.
```

```
The test-20 unit test gold file holds an example of an audit report  
covering all the possible outputs it can emit. Each unit test runs  
the report as well, and they have been updated accordingly.
```

- For more on best practices, check out this article for reference from time to time: <https://www.git-tower.com/learn/git/ebook/en/command-line/appendix/best-practices>

7.7 Opening a pull request

1. **If there are multiple commits, squash down the commits to one**
2. For more complicated commits it is appropriate to have more than one
3. Commit the changes
4. Click on New Pull Request
5. Write appropriate Pull Request Title stating the fix
6. Use present tense (ex. Fixes, Changes, Fixing, Changing..)
7. Reference the issue that the PR is fixing with **"Fixes #issue_number"** in the description
8. Provide a detailed description of the changes; if UI related, add screenshots
9. Make sure that the branches can be automatically merged (otherwise rebase the PR with **master**) and then click the drop down next to, Create pull request, and select **Create draft pull request**
10. Assign the PR to yourself and add appropriate labels
11. Add **"DO NOT MERGE"** label if the work does not need to be merged or there is no agreement on the work yet
12. Make sure to add Milestone to the PR to mention specific release
13. Request for review once the work is ready for getting reviewed

14. Select the Ready for Review button to move the PR out of Draft mode indicating it is ready for review and merging

7.8 Creating an Issue

1. Make sure to add proper details to the Issue raised
2. Upload screenshot(if possible) in dashboard issues
3. Apply proper labels to the Issue
4. Try to actively respond to the communication in case of comments in the same issue.

7.9 Reviewing a pull request

1. Go to Files changed and check for the fixes proposed by the Pull Request
2. Check for certain general criteria:
3. The PR has no merge conflicts with the base branch
4. The commits are squashed into one
5. There is proper indentation and alignment
6. No additional lines are added unnecessarily
7. The code is clearly understandable, with comments if necessary to clarify
8. Do not merge the PR with DO NOT MERGE or WIP label.
9. In case of the requirement of running the changes in the PR on the local system, follow the mentioned process:
 - To fetch a remote PR into your local repo,

```
git fetch origin pull/ID/head:BRANCHNAME
where ID is the pull request id and BRANCHNAME is the name of the new branch that you
want to create. Once you have created the branch, then simply
git checkout BRANCHNAME
```

- If modification is required, then either “Request changes” or add “General comments” for your feedback
- For more information about reviewing PR in github go through: <https://help.github.com/en/articles/about-pull-request-reviews> <https://help.github.com/en/articles/reviewing-proposed-changes-in-a-pull-request>