httperf
Web Workload Generator
Quickstart Guide

Written By: Theodore Bullock - 2007
WWW2007 Web Tools Developer Track

DESCRIPTION

This document reflects httperf version 0.9.0

httperf is a tool for UNIX-Like Operating Systems to measure web server performance. It was originally written for Hewlett Packard by David Mosberger. It can generate a number of workloads and speaks HTTP/1.0 and HTTP/1.1. Performance metrics are printed from a variety of measurements after each experiment. **NOTE:** To ensure correct results, httperf *should not* be run on the same machine as the server and should be the only significant CPU bound process on the client machine.

BASIC EXAMPLES

```
httperf --hog --server=www
```

Creates one connection to host www and requests the root document (http://www/)

```
httperf --hog --server=www --num-conns=100 --rate=10 --timeout=5
```

As above, except that 100 connections are created at a fixed rate of 10 per second

```
httperf --hog --server=www --wsess=10,5,2 --rate=1 --timeout=5
```

Generates a total of 10 **sessions** at a rate of 1 **session** per second. Each session consists of 5 calls that are spaced out by 2 seconds.

```
httperf --hog --server=www --wsess=10,5,2 --rate=1 --timeout=5
--ssl
```

As above, except that httperf contacts server www via SSL at port 443 (the default port for SSL connections).

```
httperf --hog --server=www --wsess=10,5,2 --rate=1 --timeout=5
--ssl --ssl-ciphers=EXP-RC4-MD5:EXP-RC2-CBC-MD5 --ssl-no-reuse
--http-version=1.0
```

As above, except that httperf will inform the server which cipher suites to use (EXP-RC4-MD5 or EXP-RC2-CBC-MD5); also, httperf will use HTTP version 1.0 which requires a new TCP connection for each request; additionally, SSL session ids are not reused, so the entire SSL connection establishment process occurs for each connection.

WORKLOAD GENERATORS

`Default`

- Generates fixed number of HTTP GET requests and measures the rate and number of responses

`--wlog=B,F`

- Iterates requests over a list of URI addresses (eg. replay server log)
- Parameter F is the path of a file containing the ASCII NUL **'\0'** separated list of URI addresses
- If parameter B = **n**, httperf will stop the experiment at the end of the list or after N calls (see **--num-conns=N**), whichever is shorter
- If parameter B = **y**, httperf will wrap around to the beginning of the file when reaching the end of the list (so the list of URIs is accessed repeatedly)

`--wsess=N1,N2,X`

- Measures sessions rather than individual requests. **A session is several bursts which are spaced out by the user think-time**. Each burst consists of a fixed number L calls to the server (see **--burst-length=L**)
- Bursts are organized as follows:
  - → A single call is issued
  - → After reply received, all remaining calls in the burst are issued concurrently
- Parameter N1 is the total number of sessions to generate
- Parameter N2 is the number of calls per session
- Parameter X is the user think-time (in seconds) between call bursts.
- Consider the example options **--wsess=100,50,10 --burst-length=5**:
  - → Test includes 100 sessions with a total of 50 calls each
  - → Each burst has a length of 5 calls, thus 10 call bursts per session
  - → The user think-time between call bursts would be 10 seconds
- Experiment finishes after N1 of sessions have either failed or completed
  - → A session fails if any operation in a session takes longer than the timeouts specified by options **--timeout** and **--think-timeout**

`--wsesslog=N,X,F`

- Specifies a session workload generator similar to **--wsess**
- Parameter N is the total number of sessions to generate
- Parameter X is the burst-to-burst user think time (note that this becomes a default time since the input file F can also specify user think time on a per-burst basis)
- Input file F specifies the following:
  - → Number and sequence of URIs
  - → request method
  - → think-time (per burst)
  - → burst-length
- A small example input file can most easily show the settable parameters:

Example wsesslog input file

```
# Comment lines start with a "#" as the first character. Lines
# with only whitespace delimit sessions (multiple blank lines do
# not generate "null" sessions). All other lines specify a
# uri-sequence (1 URI per line). If the first character of the
# line is whitespace (e.g. space or tab), the URI is considered
# to be part of a burst that is sent out after the previous
# non-burst uri.

# session 1 definition (this is a comment)
/foo.html think=2.0
      /pict1.gif
      /pict2.gif
/foo2.html method=POST contents='Post data'
      /pict3.gif
      /pict4.gif

# session 2 definition
/foo3.html method=POST contents="Multiline\ndata"
/foo4.html method=HEAD
```

The above description specifies 2 sessions

SESSION 1: Request /foo.html and wait for response
SESSION 1: Concurrent requests for /pict1.gif and /pict2.gif and wait for response
SESSION 1: Wait for think time of 2 seconds
SESSION 1: Request /foo2.html with POST with content 'Post data' and wait for response
SESSION 1: Concurrent requests for /pict3.gif and /pict4.gif
SESSION 1: End Session; wait for period specified with --rate or --period
SESSION 2: Request /foo3.html with POST with content "Multiline\ndata" and wait for response
SESSION 2: Wait for think time of specified by X parameter to --wsesslog
SESSION 2: Request /foo4.html with HEAD and wait for response
SESSION 2: Wait for think time of specified by X parameter to --wsesslog
SESSION 2: End Session

Note: If the N parameter of --wsesslog is greater than the number of sessions defined in input file F, then the defined sessions are used repeatedly until N sessions have been created

Note 2: Avoid using --wsesslog at the same time as other session behavior or URI workload options (eg, --burst-length, --wsess, --wlog, and --wset)

--wset=N,X

- Iterate through a list of URIs at a given rate, used to generate predictable load on the I/O server subsystem, files should be larger than server buffer cache size
- Parameter N specifies the number of distinct URIs that should be generated
- Parameter X specifies the rate at which new URIs are accessed
  - A rate of 0.25 means that the same URI is accessed four times before moving on to the next URI
  - URIs generated are of the form prefix/path.html, where prefix is the URI prefix specified by the **--uri** option and path is generated as follows:
    - for the i-th file in the working set, write down i in decimal, prefixing the number with as many zeroes as necessary to get a string that has as many digits as N-1
    - Insert a slash character between each digit. For example, the 103rd file in a working set consisting of 1024 files would result in a path of ``0/1/0/3". Thus, if the URI-prefix is /wset1024, then the URI being accessed would be /wset1024/0/1/0/3.html

COMMON OPTIONS

The following set of options are the subset that represent the most immediately useful controls for httperf. Greater tuning and reporting information is available, however these are beyond the scope of this document. Please refer to the httperf man page for extended documentation.

--hog

This option should be specified for any serious test.

This option allows httperf to request as many TCP ports as it needs. Without this option, httperf is usually limited to using ephemeral ports (in the range from 1024 to 5000). Also, this option must be specified when measuring NT servers since it avoids a TCP incompatibility between NT and UNIX machines.

--server=S

Specifies the IP hostname of the server. If this option is not specified, the hostname "localhost" is used. This option should always be specified as it is generally not a good idea to run the client and the server on the same machine.

--burst-length=N

Specifies the number N calls per burst, where N is greater or equal to one. The exact meaning of this parameter depends on the workload generator, see the description of option --wsess.

--client=I/N

Specifies that the machine httperf is running on is client I out of a total of N clients. I should be in the range from 0 to N-1. Some of the workload generators (see **--wset**) use this identity to ensure workloads differ between clients. When performing a test that involves several client machines, it is generally a good idea to specify this option.

--max-connections=N

Limits the maximum number of open connections per session to N. This option is meaningful only with options --wsess and --wsesslog.

--num-calls=N

This has meaning for request-oriented workloads only (Not **--wsess** and **--wsesslog**). It specifies the total number of calls to issue on each connection. If N is greater than the default of 1, the server must have support persistent connections. If **--burst-length=B** is set, then the N calls are issued in bursts of B. Thus, the total number of such bursts will be N/B (per connection).

--num-conns=N

This has meaning for request-oriented workloads only (Not **--wsess** and **--wsesslog**). It specifies the total number of connections to create. On each connection, calls are issued as specified by options **--num-calls** and **--burst-length**. A test stops as soon as the N connections have either completed or failed. A connection fails if no response is received with the cumulative timeout times (sum of **--timeout** and --think-timeout). The default value for this option is 1.

--period=[D]T1[,T2]

Specifies the time interval between the creation of connections or sessions. This connection/session "interarrival time" can also be specified by the **--rate** option, although **--period** provides more flexibility.

- The optional **D** parameter specifies how the interarrival time is distributed
  - → If omitted or set to **"d"**, the period is fixed and is equivalent to --rate=$T1^{-1}$, eg) --period=d0.2 or --period=0.2 is the same as --rate=5
  - → If D is set to **"e"**, the period is distributed exponentially (eg. Poisson) around the mean interarrival time T1
  - → If D is set to **"u"**, the period is uniformly distributed over the interval [T1,T2)
- The default period of 0 results in connections or sessions generated as soon as the previous one completes

--rate=X

Specifies the discrete rate at which connections or sessions are created. The default rate of 0 results in connections or sessions being generated as soon as the previous one completes.

--timeout=X

Specifies time (in seconds) that httperf will wait for a server response (default is infinity). The timeout value is used during the following activities:

- establishing a TCP connection
- sending a request
- waiting for a reply
- receiving a reply

If the timeout time is reached during any of those activities the request is considered to have failed and the connection or session is closed. The actual timeout value used when waiting for a reply is the sum of this timeout and the think-timeout.

--think-timeout=X

Specifies the time (in seconds) server has start replying to a request. This timeout value is added to the normal timeout value (see option –timeout). This option usually only useful when accessing dynamically created web content (Especially slow CGI content)

--uri=S

Specifies the path to the requested resource on the server. For some of the workload generators (e.g., --wset), this option specifies the path prefix instead of the full path.

OUTPUT

This section describes the metrics output at the end of each experiment. Standard metrics output is organized across the following groups:

- Overall results **"Total"**
- Connection results **"Connection"**
- Request results **"Request"**
- Reply Results **"Reply"**
- miscellaneous results **"CPU"** and network **"Net I/O"** utilization
- Error Summary **"Errors"**
- Only for session based experiments, **"Session"**

Total Section

```
Total: connections 500 requests 1726 replies 1632 test-duration 11.965 s
```

Lists the total number of TCP connections initiated, the number of requests, the number of replies and the overall time spent testing.

Connection Section

This section reports on the performance metrics of the TCP connections generated during the test.

```
Connection rate: 41.8 conn/s (23.9 ms/conn, <=93 concurrent connections)
```

This line shows that new connections were initiated at a rate of 41.8 connections per second, with a period of 23.9 ms. There were at most 93 concurrent open connections to the server.

```
Connection time [ms]: min 1.7 avg 1313.1 max 3093.9 median 2003.5 stddev
1094.1
```

This line shows the lifetime statistics for successful connections which is the time between connection initiation and the time the connection is closed. A connection is considered successful if it had at least one call that completed successfully.

- The minimum ("min") connection lifetime was 1.7 milliseconds
- the average ("avg") lifetime was 1313.1 milliseconds
- the maximum ("max") was 3093.9 milliseconds

- the median ("median") lifetime was 2003.5 milliseconds
- the standard deviation of the lifetimes was 1094.1 milliseconds

The median lifetime is computed based on a histogram with one millisecond resolution and a maximum lifetime of 100 seconds. Thus, the median is accurate to within half a millisecond if at least half of the successful connections have a lifetime of no more than 100 seconds.

```
Connection time [ms]: connect 0.1
```

This metric is the average time to establish a successful connection.

```
Connection length [replies/conn]: 4.020
```

This is the average number of replies received on each connection that received at least one reply. Connections that failed before yielding the first reply are not counted. This number can be bigger than 1.0 due to persistent connections.

Request Section

```
Request rate: 144.3 req/s (6.9 ms/req)
```

This shows the rate at which HTTP requests were issued (144.3 reqs/s) and the period that this rate corresponds to (6.9 ms/req). With no persistent connections, the results are similar to the connection results.

```
Request size [B]: 81.0
```

This is the average size of the HTTP requests in bytes (81.0 bytes here).

Reply Section

This section provides the most interesting information for simple measurements.

```
Reply rate [replies/s]: min 84.2 avg 148.2 max 212.2 stddev 90.5 (2
samples)
```

- The minimum "min" reply rate was 84.2 replies per second
- The average "avg" was 148.2 replies per second
- The maximum "max" rate was 212.2 replies per second
- The standard deviation "stddev" was 90.5 replies per second
- The number enclosed in parentheses shows that only 2 reply rate samples were acquired to create this data, thus it is not particularly meaningful in this example

At present, httperf collects a rate sample once every five seconds. To obtain a meaningful standard deviation, it is recommended to run tests long enough so at least thirty samples are obtained. This corresponds to a test duration of only 11.965 seconds (thus only two samples).

```
Reply time [ms]: response 74.1 transfer 2.7
```

This line shows the average time for the server to respond to a request and the average time it took to receive the reply. In the example, it took on average 74.1 milliseconds between sending the first byte of the request and receiving the first byte of the reply. The time to "transfer", or read was on average 2.7 milliseconds.

```
Reply size [B]: header 234.0 content 95506.0 footer 0.0 (total 95740.0)
```

This information contains statistics on the average length of reply headers, the content, and footers (bytes). The average total number of bytes in the replies is also given in parentheses. In the example, the average header length "header" was 234 bytes, the average content length "content" was 95506 bytes, and there were no footers "footer length" is zero. The total reply length of 95740 bytes on average.

```
Reply status: 1xx=0 2xx=1632 3xx=0 4xx=0 5xx=0
```

The final line in this section is a histogram of the major status codes received in the replies from the server. The major status code is the "hundreds" digit of the full HTTP status code. In the example, all 1632 replies had a major status code of 2. It's a good guess that all status codes were "200 OK" but the information in the histogram is not detailed enough to allow distinguishing status codes with the same major code.

Miscellaneous Section

This section summarizes the CPU and network utilization on the client machine.

```
CPU time [s]: user 1.61 system 9.27 (user 13.5% system 77.5% total 90.9%)
Net I/O: 12764.4 KB/s (104.6*10^6 bps)
```

The "CPU time" line shows that 1.61 seconds were spent executing in user mode and, 9.27 seconds were spent executing in system mode (System Calls, Interrupts, etc) and that this corresponds to 13.5% user mode execution and 77.5% system execution. The total utilization was 90.9%. Since the total CPU utilization was significantly less than 100%, this is a sign that there were competing processes that interfered with the test. Thus the results are questionable for this experiment

The line labeled "Net I/O" gives the average network throughput in **kilobytes** (not kilobits) per second. In the example, an average network usage of about 12764.4 kilobytes per second was sustained. The number in parentheses shows that this corresponds to about 104.6 megabits per second. This network bandwidth is computed based on the number of bytes sent and received on the TCP connections. In other words, it does not account for the network headers or TCP retransmissions that may have occurred.

Errors Section

```
Errors: total 94 client-timo 94 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
```

This section is a histogram of the errors that were encountered during an experiment. In the example, the two lines labeled "Errors" show that there were a total of 94 and that all errors were due to the server time-outs (client-timo). This implies that the server was close to its saturation point for the given timeout threshold (not necessarily overall saturation with real world user agent timeouts).

Error Descriptions

A description of each error counter follows:

- **client-timo**
  - → The number of times a session, connection, or call failed due to a client timeout (as specified by the **--timeout** and **--think-timeout**) options
  - → This is the most useful to identify server saturations points
- **socket-timo**
  - → The number of times a TCP connection failed with a socket-level timeout (ETIMEDOUT)
- **connrefused**
  - → The number of times a TCP connection attempt failed with a "connection refused by server" error (ECONNREFUSED)
- **connreset**
  - → The number of times a TCP connection failed due to a RESET from the server
  - → Typically, a RESET is received when the client attempts to send data to the server at a time the server has already closed its end of the connection. NT servers also send RESETs when attempting to establish a new connection when the listen queue is full
- **fd-unavail**
  - → The number of times the httperf process was out of file descriptors. Whenever this count is non-zero, the test results are meaningless because the client was overloaded (see section **CHOOSING TIMEOUT VALUES**)
- **addrunavail**
  - → The number of times the client was out of TCP port numbers (EADDRNOTAVAIL). This error should never occur. If it does, the results should be discarded.
- **ftab-full**
  - → The number of times the system's file descriptor table is full. Again, this error should never occur. If it does, the results should be discarded.
- **other**
  - → The number of times some other type of error occurred. Whenever this counter is non-zero, it is necessary to track down the real cause of the error. httperf prints the error code (errno) of the first unknown errors encountered

Session Section

When **--wsess** or **--wsesslog** is specified, httperf generates and measures sessions instead of individual calls and additional statistics are printed at the end of a test. An example output is shown below.

```
Session rate [sess/s]: min 18.60 avg 33.93 max 52.60 stddev 24.04
(406/500)
Session: avg 1.00 connections/session
Session lifetime [s]: 1.3
Session failtime [s]: 1.0
Session length histogram: 94 0 201 0 0 0 205
```

The line labeled "Session rate" shows the minimum, average, and maximum rate at which sessions completed and the standard deviation thereof (based on a 5 second sampling interval). The numbers in parentheses show how many sessions succeeded and how many sessions were initiated. In the example above, the minimum, average, and maximum session completion rates were 18.60, 33.93, and 52.60 sessions per second, respectively. The standard deviation was 24.04 and 406 out of 500 sessions completed successfully (94 failed due to errors such as timeouts).

The line labeled "Session:" shows the average length of a session measured in connections. In the example above, an average of 1 connection was required to complete a session (Persistent Connections).

The line labeled "Session lifetime" gives the average time it took to complete a successful session. In the example above, it took an average of 1.3 seconds.

The line labeled "Session failtime" gives the average time it took before an unsuccessful session failed. In the example above, it took on average 1.0 second for a session to fail.

Finally, the line labeled "Session length histogram" gives a histogram of the number of replies received by each session. In the example above, 94 sessions ended after receiving no reply at all, zero ended after receiving one reply, 201 ended after receiving 2 replies and so on. Note that this histogram does not distinguish between successful and failed sessions.

CHOOSING TIMEOUT VALUES

Since the machine that httperf runs on has a finite set of resources available, it can not sustain arbitrarily high HTTP loads. One limiting factor is that there are only roughly 60,000 TCP port numbers available at a time. Since on most UNIX-Like systems it takes one minute for a TCP connection to be fully closed (leave the TIME_WAIT state), the maximum rate a client can sustain is at most 1,000 requests per second.

The sustainable rate is often lower than that. Before httperf starts to run out of TCP ports, it is likely to run out of file descriptors (one file descriptor is used for each open TCP connection). By default, HP-UX and most Linux systems allow 1,024 open file descriptors per process. This means that without extra precautions, httperf could potentially very quickly use up all available file descriptors, at which point it could not induce any additional load on the server. If you are receiving **fd-unavail** errors, then this is your problem.

To avoid this problem, httperf provides the option **--timeout** to set a timeout for all communication with the server. If the server does not respond before the timeout expires, the client considers the corresponding session, connection, or call to be "dead", closes the associated TCP connection, and increases the **client-timo** error count. The only exception to this rule is that after sending an entire request to the server, httperf allows the server to take some additional time before it starts sending the reply. This is to accommodate HTTP requests that take a long time to complete on the server. This additional time is called the "server think time" and can be specified by option **--think-timeout**. By default, this additional think time is zero seconds, so the server would always have to respond within the time alloted by option **--timeout**.

Timeouts allow httperf to sustain high offered loads even when the server is overloaded. For example, with a timeout of 2 seconds and assuming that 1,000 file-descriptors are available, the offered load could be up to 500 requests per second (in practice, the sustainable load is often somewhat smaller than the theoretical value). On the downside, timeouts artificially truncate the connection lifetime distribution. Thus, it is recommended to pick a timeout value that is as large as possible yet small enough to allow sustaining the desired offered rate. A timeout as short as one second may be acceptable, but larger timeouts (5-10 seconds) are preferable.

It is important to keep in mind that timeouts do not guarantee that a client can sustain a particular offered load. There are many other potential resource bottlenecks. For example, in some cases the client machine may simply run out of CPU time. To ensure that a given test really measured the server's capabilities and not the client's, it is a good idea to vary the type and number of machines participating in a test. If the observed performance remains the same when the types and number of client machines are varied, the test results are likely to be valid.

Analysis of CPU and I/O load on the server during, and following experiments are excellent mechanisms for identifying if a server has been saturated and if the timeout values should be adjusted, or additional client machines should be added to the experiment. Again, be sure that httperf is not run on the same machine as the server.

ABOUT THIS DOCUMENT

This document was written in preparation for the WWW 2007 conference in Banff (May 8-12, 2007). It is roughly based on the README and man page associated with the tool, but with additional focus expended on introducing the tool. Information on the full extended option set is available in the httperf man page.

GETTING HTTPERF

Hewlett Packard Labs maintains the official httperf website, the source code can be downloaded from there under the terms of GNU Public License v2. The most recent version as of this writing is httperf-0.9.0.

http://www.hpl.hp.com/research/linux/httperf/

BUILDING HTTPERF

The following commands can be issued from the httperf source directory to build the tool

```
        $ mkdir build
        $ cd build
        $ SRCDIR/configure
        $ make
        $ make install
```

In this example, SRCDIR refers to the httperf source directory. The last step may have to be executed as "root".

SUPPORTED PLATFORMS

httperf 0.9.0 has been tested successfully on the following platforms:

- HP-UX 11i (64-bit PA-RISC and IA-64)
- Red Hat Enterprise Linux AS (AMD64 and IA-64)
- SUSE Linux 10.1 / openSUSE 10.2 (i386)
- OpenBSD 4.0 (i386)
- FreeBSD 6.0 (AMD64)
- Solaris 8 (UltraSparc 64-bit)

GETTING SUPPORT

httperf is maintained and developed through an email mailing list maintained by Hewlett Packard Labs. Please send commentary, questions and corrections regarding this document and the tool to the httperf mailing list. To subscribe to the mailing list send "subscribe" (without the quotation marks) in the body of an email to:

httperf-request@linux.hpl.hp.com

To post to the mailing list send an email to:

httperf@linux.hpl.hp.com

ABOUT THE AUTHOR

Theodore Bullock is a Software Engineering graduate of the University of Calgary Schulich School of Engineering. He actively participated in the development of httperf 0.8.1 and 0.9.0 during an engineering design project in his final undergraduate year.

He can be reached at tbullock@canada.com.

Thanks to Martin Arlitt, Dr. Diwakar Krishnamurthy and Mark Nottingham