NAME

httperf − HTTP performance measurement tool

SYNOPSIS

httperf [**--burst-length** *N*] [**--client** *I*/*N*] [**--close-with-reset**] [**-d**|**--debug** *N*] [**--failure-status** *N*] [**-h**|**--help**] [**--hog**] [**--http-version** *S*] [**--num-calls** *N*] [**--num-conns** *N*] [**--port** *N*] [**--rate** *X*] [**--recv-buffer** *N*] [**--send-buffer** *N*] [**--server** *S*] [**--think-timeout** *X*] [**--timeout** *X*] [**--uri** *S*] [**-v**|**--verbose**] [**-V**|**--version**] [**--wlog y**|**n,***F*] [**--wsess** *N*,*N*,*X*] [**--wset** *N*,*X*]

DESCRIPTION

httperf is a tool to measure web server performance. It speaks the HTTP protocol both in its HTTP/1.0 and HTTP/1.1 flavors and offers a variety of workload generators. While running, it keeps track of a number of performance metrics that are summarized in the form of statistics that are printed at the end of a test run. The most basic operation of **httperf** is to generate a fixed number of HTTP GET requests and to measure how many replies (responses) came back from the server and at what rate the responses arrived.

IMPORTANT: To obtain correct results, it is necessary to run at most one **httperf** process per client machine. Also, there should be as few background processes as possible both on the client and server machines.

EXAMPLES

httperf --hog --server www

This command causes **httperf** to create a connection to host www, send a request for the root document (http://www/), receive the reply, close the connection, and then print some performance statistics.

httperf --hog --ser www --num-conn 100 --rate 10 --timeout 5

Like above, except that a total of 100 connections are created and that connections are created at a fixed rate of 50 per second. Note that option ‘‘--server’’ has been abbreviated to ‘‘--ser’’.

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

Causes **httperf** to generate 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.

OPTIONS

The operation of **httperf** can be controlled through a number of options. The tool supports both short (one-character) and long (arbitrary-length) option names. Short options are prefixed with a single leading dash (-), long options with a double-dash (--). Multiple short options can be grouped together (e.g., ‘‘**-vV**’’ is equivalent to ‘‘**-v -V**’’) and long options can be abbreviated so long as they remain unique. Parameters to options can be specified either by following the long option name with an equal sign and the parameter value (e.g., **--burst=10**) or by separating the option name and value with whitespace (e.g., **--burst 10**).

--burst-length=*N*

Specifies the length of bursts. Each bursts consists of *N* calls to the server. The exact meaning of this parameter depends on the workload generator. For regular request-oriented workloads, this parameter specifies how many calls are issued concurrently on each connection. For session-oriented workloads, 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 (e.g., **--wset)** use the client identity as a bias value to ensure that not all clients generate perfectly identical workloads. When performing a test that involves several client machines, it is generally a good idea to specify this option.

--close-with-reset

Requests that **httperf** closes TCP connections by sending a RESET instead of going through the normal TCP connection shutdown handshake. Turning on this option can have ill effects such as data corruption, stuck TCP control blocks, or wrong results. For this reason, the option should not be used unless absolutely necessary and even then it should not be used unless its implications are

fully understood.

-d=*N*

--debug=*N*

Set debug level to *N*. Larger values of *N* will result in more output.

--failure-status=*N*

Specifies that an HTTP response status code of *N* should be treated as a failure (i.e., treated as if the request had timed out, for example). For example, with ''**--failure-status=504**'' responses with an HTTP status of ''504 Gateway Time-out'' would be considered failures. Caveat: this option is currently supported for session workloads only (see **--wsess** option).

-h

--help Prints a summary of available options and their parameters.

--hog This option requests to use up as many TCP ports as necessary. Without this option, **httperf** is typically limited to using ephemeral ports (in the range from 1024 to 5000). This limited port range can quickly become a bottleneck so it is generally a good idea to specify this option for serious testing. Also, this option must be specified when measuring NT servers since it avoids a TCP incompatibility between NT and UNIX machines.

--http-version=*S*

Specifies the version string that should be included in the requests sent to the server. By default, version string ''1.1'' is used. This option can be set to ''1.0'' to force the generation of HTTP/1.0 requests. Setting this option to any value other than ''1.0'' or ''1.1'' may result in undefined behavior.

--num-calls=*N*

This option is meaningful for request-oriented workloads only. It specifies number of calls to issue on each connection. If *N* is greater than 1, the server must support persistent connections. The default value for this option is 1.

--num-conns=*N*

This option is meaningful for request-oriented workloads only. 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 is considered to have failed if any activity on the connection fails to make forward progress for more than the time specified by the timeout options **--timeout** and **--think-timeout**. The default value for this option is 1.

--port=*N*

This option specifies the port number *N* on which the web server is listening for HTTP requests. By default, **httperf** uses port number 80.

--rate=*X*

Specifies the fixed rate at which connections or sessions are created. Connections are created by default, sessions if option **--wsess** has been specified. In both cases a rate of 0 results in connections or sessions being generated sequentially (a new session/connection is initiated as soon as the previous one completes). The default value for this option is 0.

--recv-buffer=*N*

Specifies the maximum size of the socket receive buffers used to receive HTTP replies. By default, the limit is 16KB. A smaller value may help memory-constrained clients whereas a larger value may be necessary when communicating with a server over a high-bandwidth, high-latency connection.

--send-buffer=*N*

Specifies the maximum size of the socket send buffers used to send HTTP requests. By default, the limit is 4KB. A smaller value may help memory-constrained clients whereas a larger value may be necessary when generating large requests to a server connected via a high-bandwidth,

high-latency connection.

--server=*S*

Specifies the IP hostname of the server. By default, 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.

--think-timeout=*X*

Specifies the maximum time that the server may need to produce the reply for a given request. Note that this timeout value is added to the normal timeout value (see option **--timeout**). When accessing static web content, it is usually not necessary to specify this option. However, when performing tests with long-running CGI scripts, it may be necessary to use this option to allow for larger response-times. The default value for this option is zero seconds, meaning that the server has to be able to respond within the normal timeout value.

--timeout=*X*

Specifies the amount of time *X* that **httperf** is willing to wait for a server reaction. The timeout is specified in seconds and can be a fractional number (e.g., **--timeout 3.5**). This timeout value is used when establishing a TCP connection, when sending a request, when waiting for a reply, and when receiving a reply. If during any of those activities a request fails to make forward progress within the alloted time, **httperf** considers the request to have died, closes the associated connection or session and increases the **client-timo** error count. The actual timeout value used when waiting for a reply is the sum of this timeout and the think-timeout (see option **--think-timeout**). By default, the timeout value is infinity.

--uri=*S* Specifies that URI *S* should be accessed on the server. For some of the workload generators (e.g., **--wset**), this option specifies the prefix for the URIs being accessed.

-v

--verbose

Puts **httperf** into verbose mode. In this mode, additional output such as the individual reply rate samples and connection lifetime histogram are printed.

-V

--version

Prints the version of **httperf**.

--wlog=*B***,***F*

This option can be used to generate a specific sequence of URI accesses. This is useful to replay the accesses recorded in a server log file, for example. Parameter *F* is the name of a file containing the ASCII NUL separated list of URIs that should be accessed. If parameter *B* is set to ''**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). With *B* set to ''**n**'', the test will stop no later than when reaching the end of the URI list.

--wsess=*N1***,***N2***,***X*

Requests the generation and measurement of sessions instead of individual requests. A session consists of a sequence of bursts which are spaced out by the user think-time. Each burst consists of a fixed number *L* of calls to the server (*L* is specified by option **--burst-length**). The calls in a burst are issued as follows: at first, a single call is issued. Once the reply to this first call has been fully received, all remaining calls in the burst are issued concurrently. The concurrent calls are issued either as pipelined calls on an existing persistent connection or as individual calls on separate connections. Whether a persistent connection is used depends on whether the server responds to the first call with a reply that includes a ''Connection: close'' header line. If such a line is present, separate connections are used.

The option specifies the following parameters: *N1* is the total number of sessions to generate, *N2* is the number of calls per session, and *X* is the user think-time (in seconds) that separates consecutive

call bursts. For example, the options ''**--wsess=100,50,10 --burst-len 5**'' would result in 100 sessions with a total of 50 calls each. Since each burst has a length of 5 calls, a total of 10 call bursts would be generated per session. The user think-time between call bursts would be 10 seconds. Note that user think-time *X* denotes the time between receiving the last reply of the previous call burst and the sending of the first request of the next burst.

A test involving sessions finishes as soon as the requested number *N1* of sessions have either failed or completed. A session is considered to have failed if any operation in a session takes longer than the timeouts specified by options **--timeout** and **--think-timeout**. In addition, a session also fails if the server returns a reply with a status code matching the one specified by option **--failure-status**.

This session workload generator scans the server replies for ''Set-Cookie'' headers. If such a cookie is found, it is stored in the session and unconditionally sent in all future requests issued by this session. Only one cookie can be set per session. If multiple ''Set-Cookie'' headers appear over the lifetime of a session, only the most recently set cookie will be sent in the requests.

--wset=*N***,***X*

This option can be used to walk through a list of URIs at a given rate. Parameter *N* specifies the number of distinct URIs that should be generated and *X* specifies the rate at which new URIs are accessed. A rate of **0.25** would mean that the same URI would be accessed four times in a row before moving on to the next URI. This type of access pattern is useful in generating a workload that induces a relatively predictable amount of traffic in the disk I/O subsystem of the server (assuming *N* and the accessed files are big enough to exceed the server's buffer cache). The URIs generated are of the form *prefix*/*path*.html, where *prefix* is the URI prefix specified by option **--wset** 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. Then 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**. In other words, the files on the server need to be organized as a 10ary tree.

OUTPUT

This section describes the statistics output at the end of each test run. The basic information shown below is printed independent of the selected workload generator.

Total: connections 30000 requests 29997 replies 29997 test-duration 299.992 s

Connection rate: 100.0 conn/s (10.0 ms/conn, <=14 concurrent connections)
Connection time [ms]: min 1.4 avg 3.0 max 163.4 median 1.5 stddev 7.3
Connection time [ms]: connect 0.6

Request rate: 100.0 req/s (10.0 ms/req)
Request size [B]: 75.0

Reply rate [replies/s]: min 98.8 avg 100.0 max 101.2 stddev 0.3 (60 samples)
Reply time [ms]: response 2.4 transfer 0.0
Reply size [B]: header 242.0 content 1010.0 footer 0.0 (total 1252.0)
Reply status: 1xx=0 2xx=29997 3xx=0 4xx=0 5xx=0

CPU time [s]: user 94.31 system 205.26 (user 31.4% system 68.4% total 99.9%)
Net I/O: 129.6 KB/s (1.1*10^6 bps)

Errors: total 3 client-timo 0 socket-timo 0 connrefused 3 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

There are six groups of statistics: overall results (''Total''), connection related results (''Connection''), results relating to the issuing of HTTP requests (''Request''), results relating to the replies received from the server (''Reply''), miscellaneous results relating to the CPU (''CPU'') and network (''Net I/O'') utilization and, last but not least, a summary of errors encountered (''Errors'').

Total Section

This section summarizes how many TCP connections were initiated by **httperf**, how many requests it sent out, how many replies it received, and what the total test duration was. In the example output shown above, 30,000 connections were created, 29,997 requests were sent out and 29,997 replies were received. The duration of the test was almost exactly 5 minutes (300 seconds).

Connection Section

This section conveys information related to TCP connections generated by the tool. Specifically, the ''Connection rate'' line shows that new connections were initiated at a rate of 100.0 connections per second. This rate corresponds to a period of 10.0 milliseconds per connection. The last number in this line shows that at most 14 connections were open at any given time.

The first line labeled ''Connection time'' gives lifetime statistics for successful connections. The lifetime of a connection is the time between a TCP connection is initiated and the time the connection is closed. A connection is considered successful if it had at least one call that completed successfully. In the example output, the line indicates that the minimum (''min'') connection lifetime was 1.4 milliseconds, the average (''avg'') lifetime was 3.0 milliseconds, the maximum (''max'') was 163.4 milliseconds, the median (''median'') lifetime was 1.5 milliseconds, and that the standard deviation of the lifetimes was 7.3 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.

The final statistic in this section is the average time it took to establish a TCP connection. Only successful TCP connection establishments are counted. In the example, the second line labeled ''Connection time'' shows that, on average, it took 0.6 milliseconds to establish a connection.

Request Section

The line labeled ''Request rate'' gives the rate at which HTTP requests were issued and the period that this rate corresponds to. In the example above, the request rate was 100.0 requests per second, which corresponds to 10.0 milliseconds per request. As long as no persistent connections are employed, the results in this section are very similar or identical to results in the connection section. However, when persistent connections are used, several calls can be performed on a single connection in which case the results would be different.

The line labeled ''Request size'' gives the average size of the HTTP requests in bytes. In the example above, the average request size was 75 bytes.

Reply Section

For simple measurements, this section is often the most interesting one as the line labeled ''Reply rate'' gives various statistics for the reply rate. In the example above, the minimum (''min'') reply rate was 98.8 replies per second, the average (''avg'') was 100 replies per second, and the maximum (''max'') rate was 101.2 replies per second. The standard deviation was 0.3 replies per second. The number enclosed in parentheses shows that 60 reply rate samples were acquired. 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 at least 150 seconds.

The line labeled ''Reply Time'' gives information on how long it took for the server to respond and how long it took to receive the reply. In the example, it took on average 2.4 milliseconds between sending the first byte of the request and receiving the first byte of the reply. The time to ''transfer'', or read, the reply was too short to be measured, so it shows up as zero. The is typical when the entire reply fits into a single TCP segment.

The next line, labeled ''Reply size'' contains statistics on the average size of the replies---all

numbers are in reported bytes. Specifically, the line lists the average length of reply headers, the content, and footers (HTTP/1.1 uses footers to realize the ‘‘chunked’’ transfer encoding). For convenience, the average total number of bytes in the replies is also given in parentheses. In the example, the average header length (‘‘header’’) was 242 bytes, the average content length (‘‘content’’) was 1010 bytes, and there were no footers (‘‘footer’’ length is zero). The total reply length of 1252 bytes on average.

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 29,997 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 starts with a summary of the CPU utilization on the client machine. In the example, the line labeled ‘‘CPU time’’ shows that 94.31 seconds were spent executing in user mode (‘‘user’’), 205.26 seconds were spent executing in system mode (‘‘system’’) and that this corresponds to 31.4% user mode execution and 68.4% system execution. The total utilization was 99.9%, which is expected given that httperf is a CPU hog. A total CPU utilization of significantly less than 100% is a sign that there were competing processes that interfered with the test.

The line labeled ‘‘Net I/O’’ gives the average network throughput in kilobytes per second (where a kilobyte is 1024 bytes) and in megabits per second (where a megabit is 10^6 bits). In the example, an average network usage of about 129.6 kilobytes per second was sustained. The number in parentheses shows that this corresponds to about 1.1 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

The last section contains statistics on the errors that were encountered during a test. In the example, the two lines labeled ‘‘Errors’’ show that there were a total of three errors and that all three errors were due to the server refusing to accept a connection (‘‘connrefused’’). 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.

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. To assist in doing this, **httperf** prints the error code (errno) of the first unknown errors that occurs during a test run.

When option **--wsess** is specified, **httperf** generates and measures sessions instead of individual calls and also causes additional statistics to be printed at the end of a test. An example output is shown below.

Session rate [sess/s]: total 11.18 p 11.18 (4439/4800) non-p 0.00 (0/0)
Session lifetime [s]: p 96.1 non-p 0.0
Session failtime [s]: p 21.5 non-p 0.0
Session failures: non-p 0 0 0 0 0 0 ...
Session failures: p 185 4 2 4 20 9 ...

Call response time [ms]: p 279.5 non-p 0.0

The session statistics distinguish between sessions that completed on a single persistent connection (label ''p'') and sessions that used multiple connections (label ''non-p''). In the example above, the line labeled ''Session rate'' shows that 11.18 sessions completed on persistent connections, zero completed on non-persistent connections and that the total session throughput was 11.18 sessions per second. The numbers in parentheses show that 4439 out of 4800 attempted sessions completed successfully on persistent connections and that zero out of zero attempted sessions completed on non-persistent connections.

The line labeled ''Session lifetime'' shows that, on average, it took 96.1 seconds to complete a successful session on a persistent connection. Since no session completed on non-persistent connections, the session lifetime for ''non-p'' is reported as zero seconds.

The line labeled ''Session failtime'' shows that, on average, a session on a persistent connection that eventually failed lived for 21.5 seconds. Again, the value for non-persistent connections is zero as no such sessions were used.

The two lines labeled ''Session failures'' give a histogram of how many sessions failed after successfully completing *n* calls. In the example above, only the first six entries of the histogram are shown. The line for persistent connections (label ''p'') shows that 185 sessions failed even before completing a single call. Four sessions failed after completing one call, 2 failed after completing 2 calls, and so on.

The last line, labeled ''Call response time'' shows the average time it took to complete one call. In the example output, it took on average 279.5 milliseconds to complete one call on a persistent connection. The number for non-persistent connections is again zero as no non-persistent connections were used. On a persistent connection, the call response time is the time between sending the first byte of the request and receiving the last byte of the reply. On a non-persistent connection, the call response time is the time between initiating a connection and the closing of the connection.

CHOOSING TIMEOUT VALUES

Since the machine that **httperf** runs on has only a finite set of resource available, it can not sustain arbitrarily high HTTP loads. For example, one limiting factor is that there are only roughly 60,000 TCP port numbers that can be in use at any given time. Since on most UNIX 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 actual sustainable rate is often much lower than that because before running out of TCP ports, the machine is likely to run out of file descriptors (one file descriptor is used up for each open TCP connection). By default, HP-UX 10.20 allows 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. To avoid this problem, **httperf** provides 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 number of machines participating in a test. If observed performance remains the same as the number of client machines is varied, the test results are likely to be valid.

AUTHOR

httperf was developed by David Mosberger and was heavily influenced by a tool written by Tai Jin. Stephane Eranian contributed the log-file based URI generator. All three authors are with Hewlett-Packard Research Laboratories.

BUGS

Probably many. Always be sure to double-check results and don't fall prey to measuring client-performance instead of server performance!

The user-interface definitely could be improved. A simple workload description language might be more suitable than the dozens of little command-line options the tool has right now.