

Easy Site Benchmarking 1.04

2006 ff.
e-tunity

Karel Kubat

Abstract

The Easy Benchmarking Suite consists of a set of Perl programs that make website benchmarking easy and repeatable. Benchmarking data can be easily obtained, analyzed and plotted.

Contents

1 Introduction	2
1.1 Prerequisites	2
1.2 Obtaining and Installing	2
1.2.1 Installation using GNU Make	2
1.2.2 Installation by hand	3
1.3 Copyright and Warranty	3
2 sitebench - Benchmark Runner	3
2.1 Purpose and Invocation	3
2.1.1 Arguments	3
2.1.2 The Request to Send	3
2.2 Output	5
3 sitecollect - Benchmark overview generator	5
3.1 Purpose and Invocation	5
3.2 Output	5
4 siteplot - Benchmark plot generator	6
4.1 Purpose and Invocation	6
4.2 How to use siteplot	6
5 A demonstration	7

1 Introduction

The *Easy Benchmarking Suite* is aimed at testing and benchmarking sites. Basically, the suite can:

- Issue requests to a URL. A benchmark typically consists of a given number of clients that concurrently issue a number of requests. A request can be any HTTP request (GET, POST, HEAD etc.) with all necessary HTTP header information (session cookies, basic authentication information, etc.);
- Get a quick overview of the results of a benchmark: how many trials succeeded, what are the average times for connecting and processing, what is the standard deviation of these times;
- Prepare a GnuPlot command file so that the obtained results can be plotted.

This document describes the suite. Furthermore, manual pages are provided for the separate parts of the suite: `sitebench`, `sitecollect`, and `siteplot`.

1.1 Prerequisites

In order to run the Easy Benchmarking Suite, you'll need:

- A quick enough system to benchmark sites. Preferably the benchmarks shouldn't be run on the same system where the site is served, as the benchmark itself uses system resources;
- Perl 5 or better, with the following modules: `Getopt::Std`, `IO::Socket::INET`, `Time::HiRes`. Most (modern) Perl installations have these modules on-board;
- GnuPlot, if you want to make plots of collected data;
- GNU Make if you want to install this package using `make` (otherwise you can always install by hand).

1.2 Obtaining and Installing

The Easy Benchmarking Suite can be obtained at <http://public.e-tunity.com>. The suite comes as an archive `sitebenchmark-X.YY.tar.gz`, where `X.YY` is a version specifier.

1.2.1 Installation using GNU Make

To install with `make`, `untar` the archive and `change-dir` to the directory `sitebenchmark` that is created by unpacking the archive. Next, type `make install`. This installs the tools to `/usr/local/bin` and manual pages to `/usr/local/share/man`.

Optionally, you can change the installation prefix as follows:

```
export EBASE=/usr
make install
```

This installs binaries under `/usr/bin` and manpages under `/usr/share/man`.

If you're on a system that supports gzipped manpages, then optionally you can:

```
gzip /usr/local/share/man/man1/sitebench.1
gzip /usr/local/share/man/man1/sitecollect.1
gzip /usr/local/share/man/man1/siteplot.1
```

This saves some disk space by compressing the manual pages.

1.2.2 Installation by hand

To install all relevant files by hand, simply copy `src/*` to a suitable directory in your `PATH`, e.g.:

```
cp src/* /usr/local/bin
```

To install the manual pages, copy all files named `*.man` under `doc/` to a suitable directory:

```
cp doc/sitebench.man /usr/local/share/man/man1/sitebench.1
cp doc/sitecollect.man /usr/local/share/man/man1/sitebench.1
cp doc/siteplot.man /usr/local/share/man/man1/siteplot.1
```

1.3 Copyright and Warranty

This package is distributed under the Gnu Public Licence (GPL, or 'copyleft'). In short, this means that you're free to use this package as you like, and you're free to redistribute it – provided that you redistribute all your changes too. See one of the files `COPYING` on your hard disk, or check at <http://www.gnu.org>.

However, if you **do** make changes to the package, then I'd like to hear from you. If you have bugfixes, send them to me, I'll incorporate them in the next release, and state you as contributor.

As for warranty: there is none. Use at your own risk. This package is distributed as-is, without assumptions of fitness or usability.

2 sitebench - Benchmark Runner

2.1 Purpose and Invocation

The program `sitebench` collects raw data of the performance of a website. It is the workhorse of the Easy Benchmark Suite.

2.1.1 Arguments

`sitebench` requires 5 arguments and reads `stdin` to obtain the request to send to the website:

- The number of concurrent clients to simulate. The higher this number, the more concurrent requests the website will receive;
- The number of requests that each concurrent client will send. The higher this number, the longer the 'train' of hits that each client will generate;
- The server where the site to hit is located. This can be a hostname or an IP address, without preceding `http://` and without port specifier;
- The port to hit on the server. Normally this will be 80, the HTTP port;
- A string to match (regular expression). Each client will fire their request, and read back a server answer. When the string to match occurs in the answer, then the action is flagged as a 'success'.

2.1.2 The Request to Send

Furthermore `sitebench` reads `stdin` to find out what to send to the site. The request to send must be valid HTTP request. A few examples are given below.

A simple GET request is e.g. the following:

```
GET /
```

This usually requests the top-level index document (e.g. `index.html`). Given the fact that no HTTP version is specified, such requests are often called 'version 0.9' or 'ass-backwards'.

A HTTP/1.0 request with hostname is e.g. the following:

```
GET / HTTP/1.0
Host: www.mysite.org
Connection: close
```

This specifies HTTP/1.0 as the version to use. **Note that** the version specifically requests to close the TCP link after processing. **You'll always want this** when working with `sitebench`. The alternative to closing a connection is -obviously- to keep it open, but that invalidates all timings, so you won't want that.

A POST request is e.g.:

```
POST /cgi-bin/lookupname.pl HTTP/1.0
Host: www.mysite.org
Connection: close
Content-length: 11

name=Karel
```

This sends the CGI-variable `name` having value `Karel` to the stated address. Note again that the connection is not kept open.

Basic Authentication is done by adding a HTTP header `Authorization` to the request:

```
GET /protected/index.html HTTP/1.0
Host: www.mysite.org
Connection: close
Authorization: Basic abJzdfpkljkewpaiosdd==
```

The encoded string is `username:password` in base64-encoded format.

Normally you won't type such requests by hand, but will use a browser such as Firefox with a plugin such as LiveHTTPHeaders to collect a request. Such a request is then saved to a file (e.g. named `request`) and `sitebench` is run as follows:

```
sitebench 20 50 10.1.2.78 80 Welcome < request > raw
```

This starts 20 clients where each client fires 50 requests at the host 10.1.2.78, port 80. A 'hit' is considered successful when the string `Welcome` occurs in the server's answer. The output of the benchmark is collected in a file `raw`.

In order to see what's going on, `sitebench` supports two interesting flags:

- `-v` makes the processing more verbose; `sitebench` shows what it is doing, how many clients are active, and so on;
- `-s` is 'single-shot' mode. Instead of having `X` clients each firing `Y` requests, only one request by one client is issued. The server answer is shown on `stdout`. The flag `-s` is very handy for verifying that the right server page is accessed and for determining a correct matchstring.

2.2 Output

The output of `sitebench` is a stream of lines, where each line has 3 to 5 numbers. The numbers represent:

- The process ID of the client that collected the data;
- A high-resolution timestamp in seconds and microseconds since the start of epoch;
- 1 if a connection was established to the server, or 0 otherwise. When this number is 1, then the following fields are present;
- The time in seconds that was needed to connect to the server; i.e., to build up a TCP connection;
- When a request could be sent and when the server's response matched the matchstring, then the next number is 1. Otherwise this number is 0. When 1, then the next field is present;
- The processing time in seconds that was needed to send the request and to collect the response. This timing ends when the TCP connection is terminated.

Normally you won't examine these data by hand, but will use `sitecollect` or `siteplot` to get an overview.

3 sitecollect - Benchmark overview generator

3.1 Purpose and Invocation

The purpose of the program `sitecollect` is to get a quick overview of the data generated by `sitebench`. There are no arguments. The data to analyze are read from `stdin`, so that the invocations are basically:

- `sitebench sitebench-arguments |sitecollect`
- `sitecollect < raw` (where `raw` is the output of `sitebench`)

3.2 Output

The output of `sitecollect` is a quick overview of the benchmarking test. The output is quite self-explanatory; here's an example. A few remarks are made below the text.

```
Analysis Overview
Failed connects:           0
Succeeded connects:      100
Mean successful connect time: 0.04868297      (1a)
SD successful connect times: 0.0052129732341102    (1b)
Failed matches:          0
Mean failed match request time: 0
SD failed match times:   0
Succeeded matches:      100
Mean succeeded match time: 0.00671961      (2a)
SD succeeded match times: 0.00102671270735141    (2b)
Mean total processing time: 0.05540258      (3a)
SD total processing time:  0.0067818249961248    (3b)
```

Remarks:

1a and 1b: These are the timings of the building up of connections. The SD (standard deviation) is an indicator for the 'spread' of all measurements. E.g., 95% of all measurements occur within (*mean* + *sd*); or: 95% of all connections could be established within 0.054 sec. (Statistically speaking, you should not totally trust this statement – the timings don't follow a Gauss distribution, but are skewed.)

2a and 2b: These are the timings for the processing of requests; i.e., from the time that a request is sent, up to the reception of the server's answer.

3a and 3b: These timings reflect the overall accessibility of the site from the client where the benchmark was run; i.e., the connection times plus the processing times. These are only the cases where `sibench`'s match string could be found in the server's response.

In general, shorter timings are of course better. However, if you find an SD that is large, then you've a clear indication of the server reaching its limit.

4 siteplot - Benchmark plot generator

4.1 Purpose and Invocation

The purpose of `siteplot` is to analyze the data obtained with `sitebench` and to plot them.

For this, `siteplot` requires two arguments, and a datastream on `stdin`. The arguments are:

- A plot title,
- A basename which is used to generate files:
 - Files named `basename.*.data` are datasets;
 - Files named `basename*.gnuplot` are GnuPlot command files;
 - Files named `basename*.png` are PNG's of the graphs (only generated when flag `-p` is given).

The input stream must be generated using `sitebench`.

4.2 How to use siteplot

Usually, the easiest way to use `siteplot` is to let it generate a GnuPlot command file, as in

```
siteplot 'Benchmark of www.mysite.org' data < raw
```

Here, `raw` is the filename of previously collected raw output of `sitebench`. After this, the following GnuPlot command files are available:

- `data.chronological.gnuplot`: A chronological overview of the benchmark, where the X axis shows the time as the benchmark proceeded, and the Y axis shows the processing times of transactions. This plot holds as many lines as concurrent clients at the time of the benchmark.
- `data.durationordered.gnuplot`: This plot shows on the X axis requests and on the Y axis their connect timings and overall processing timings. The requests are sorted by their processing timings.

Note that this plot isn't your typical statistical overview of data, but I find it very helpful. Given the fact that the fastest transactions occur on the left side of the plot and the slowest ones on the right (sorted by processing time, remember), the plot is usually a sigmoid curve that clearly shows:

- Whether the timings are scattered or close together;
- A line that separates 95% of cases. The 'few unlucky' requests have taken longer than this value;

- A line that separates 5% of the cases. The 'few lucky' requests where this fast;
- Furthermore, 25%, 50% and 75% thresholds are shown.
- `data.histogram.gnuplot`: This plot shows a histogram of on the X axis intervals of processing time, and on the Y axis the number of transactions that fall in this interval. The intervals are determined by `siteplot` so that all timings are divided into 100 pieces.

On Unix systems, a plot is easily generated using:

```
prompt> gnuplot
gnuplot> load 'data.histogram.gnuplot'
```

This interactively shows the plot. By editing the GnuPlot command file, other output devices (e.g., a PNG writer) can be selected, so that plots are written as images.

When `siteplot` is called with the `-p` flag, then furthermore `*.png` versions of the above files will be created, holding the graphs in PNG format.

5 A demonstration

The Easy Site Benchmarking Suite has a subdirectory `test/` where you may see how things work. The directory has a pre-cooked file `freshmeat.net.raw` with timings of `http://freshmeat.net`, at 40 concurrent clients. Type `make` in the `test/` subdirectory to complete tests.

This will generate an overview using `sitecollect`, but more importantly, it will generate a GnuPlot command file `freshmeat.net.gnuplot`. To view the plot, fire up GnuPlot and enter at the prompt: `load 'freshmeat.net.gnuplot'`.

The plot immediately shows that the timings are relatively smoothly distributed:

- 5% of the requests were serviced within approx. 7 seconds;
- The median of the servicing time is aprox. 16 seconds;
- 95% of the requests were serviced within approx. 27 seconds.

If one assumes that 16 seconds is acceptable for most clients, then it follows that a few lucky ones are serviced somewhat quicker, and a few unlucky ones are serviced somewhat slower – but all this is still acceptably near to oneother. Of course, the decision whether 16 seconds is acceptable, is quite another one...