The performance limit of HTTP-servers

Beat Christen[1]

Abstract

Web-servers have become an integral part of today's information technology infrastructure, as engineers have started to recognise HTTP as an excellent way to do remote procedure calls, due to the simplicity of the protocol, which allows to write and integrate an HTTP client or server in very short time. In this project, a very simple HTTP daemon has been implemented to show the maximum possible performance, by getting rid of all functionality which is not absolutely necessary or required for operation. As a comparison, the performance of general web-servers such as Apache[1] and Zeus[2] are being evaluated. What is the upper limit of performance a web-server *can* achieve compared to a standard web-server? At least for the system you have, you can find out what the maximum is. The implemented benchmarking software is available for download[3].

Introduction

The main goal of this work is to find out the key elements of a web-server which are absolutely needed for operation, as compared to functions which are nice to have such as logging, test of accuracy of the served data (i.e. checking if the in-cache copy of the file is still the same), verbose response headers and so on.

After creating such a minimal web-server, let's call it 'bummel', its number of requests/second served should be compared to the performance of standard web-servers. By gradually adding desirable functionality to 'bummel', the impact should be compared to a leanest possible implementation. For example, different logging strategies can be implemented or getting rid of the cache design and serving files directly from disk, thus adding more context-switches into the kernel of the OS.

How to implement a fast web-server

The implementation of 'bummel' mainly consists of a standard socket server implementation, which is run as n threads simultaneously.

The accept system-call which receives the incoming request is guarded by a mutex variable. This leads to the practical advantage that only threads which are preparing a response are scheduled for running by the OS, since all others are waiting on the mutex or blocked by the accept system-call.

No other unnecessary context-switches into the kernel are performed, the response data is loaded upon start of the system and headers are pre-generated for each file. Thus, the implementation invokes only the minimal number of system-calls to send a HTTP-response. They are `accept` (to receive the connection), `read` (to get the request), `write` (to send the response), `shutdown` (to initiate the shutdown of the connection) and `close` (to terminate the connection).

Benchmarked Services

The tested services involve transmission of static HTTP requests only. Such a request can be broken down to network-connection set-up, transmission of HTTP request, processing of request in client, transmission of HTTP response, network-connection tear-down. All but the processing of the request shall be accepted as is on each platform, since this is normally out of reach for change to application developers.

Applicable Metrics of a web-server

Some of the possible metrics for analysing the performance of a web-server could be:

- data throughput
- requests/second
- response time on client side
- CPU utilisation

Since CPU utilisation and concurrency of requests in the server would require profiling support in the server application to get accurate data, they have been left out in this study. Some of the web-servers are available as binary only, so in some cases, accurate profiling would only be possible from the outside, where profiling could impact on the server performance.

With high profile installations (>1million hits / day), the available hardware is used for web-serving only, so CPU utilisation would be allowed to reach a full load situation, as long as it increases the total throughput.

Response time is not really an interesting factor, since HTTP is used interactively in web-browsers. Common web-servers usually have a response time between 5-20 ms for static web-pages, so an improved server would not increase the browsing clients satisfaction.

[1] Beat Christen is a CS student at the Swiss Federal Institute of Technology (ETH Zurich)

System Parameters

The components that contribute to the performance of the system as a whole and their estimated contribution to performance on the server are

- *OS*: mostly its TCP/IP subsystem.
- *CPU*: contributes to total number of requests served.
- *Disk*: in the case where we read from disk directly - no problem with local disks).
- *Network interface bandwidth*: can have a limiting impact, if the benchmark set-up is not chosen wisely. Networks with less than 100Mbits/s bandwidth can easily be saturated by all common web-server/platform combinations.
- *Web-server*: 2 common web-servers will be benchmarked, along a mock-up implementation of a web-server ('bummel').

Varied Parameters

The factors in this analysis will be:

- the web-server in use,
- the operating system it is running on.
- the number of simultaneous threads requesting pages from the server

Web-server 1

Apache 1.3.6

Web-server 2

Bummel (custom built web-server mock-up)

Web-server 3:

Zeus v3.3.5 ('fastest' available commercial web-server.

System A

OS	Linux 2.2.5
CPU	Pentium II, 350MHz
RAM	128 Mb
Disk	IBM-DHEA-36480, IDE interface
Network Interface	100 Mbit/s on full duplex switching network

System B

OS	IRIX 6.5
CPU	MIPS R10000 167Mhz
RAM	256 Mb
Disk	unknown manufacturer, SCSI interface
Network Interface	100 Mbit/s on full duplex switching network

System C

OS	SunOS 5.7
CPU	UltraSparc 250MHz
RAM	256 Mb
Disk	unknown manufacturer, SCSI interface
Network Interface	100 Mbit/s on full duplex switching network

Evaluation Technique

The technique used to perform analysis is measuring the real system's performance. To minimise variability due to network load, the performance tests have been conducted during the night.

Workload

Workload A (bummel.html)
The workload consists of a small file (2692 bytes), which is being requested 10000 times from a single client machine to avoid congestion on the link to the server machine.
Workload B (index.html)
The workload consists of a file with 15096 bytes which is being requested 10000 times in a row from a single machine to avoid congestion on the link to the server machine.

Workload C (error.html)
The workload consists of 10000 requests for a file which does not exist on the server. It is the web-servers choice to chose the amount of data to be sent back.

The chosen workloads might seem too simplistic, but requesting different files from the server would very likely be affected by the file-cache used by the underlying OS, if caching is not activated. The goal is to find the critical path inside the web-server application, so naturally we are requesting very small files to increase load on the server. We don't want to simulate normal user behaviour (which usually is a magnitude of 1000 lower compared to these tests), but rather want to know where the limits for a leanest possible implementation are.

Experiment

Since the systems where the web-server is run can not be configured in respect to CPU/memory/disk set-up, the only variables will be the functionality in the web-server and the workload types.
Each of the workloads is performed with 4, 8 & 16 simultaneous threads starting requests to the server.
A full factorial design is performed with

81 experiments = (3 Platforms)(3 web-servers)(3 workloads)(3 #/threads)

ApacheBench is being used as the traffic generator on a machine on the same 100Mbit/s network segment to achieve a maximally possible load.

Note: *Not all performance tests have been conducted yet. The numbers for Irix (both Apache & Zeus) and SunOS (both Apache & Zeus) are missing, as an installation was not available at the time of writing. They will be added in a later version of this paper. Also, another web-server might be added to the data section.*

Analysis

The performance of the standard web-servers Apache and Zeus were – as expected – slower than the 'as lean as possible' implementation of 'Bummel'.
All platforms show similar numbers for peak performance, which seems to be optimal around 8 concurrent threads serving requests.
The numbers for SunOS have not been verified yet. Assuming that this platform should achieve similar results as the Linux and Irix, the reason for the mediocre performance on average should be sought in the installation. The author assumes that either a socket option in Bummel has not been set appropriately, so the OS can make some assumptions about the use of the socket or there was a problem with the link at the time of testing. This issue is subject to further investigation at this moment.

Interpretation

The author draws the following conclusions from the performed tests:

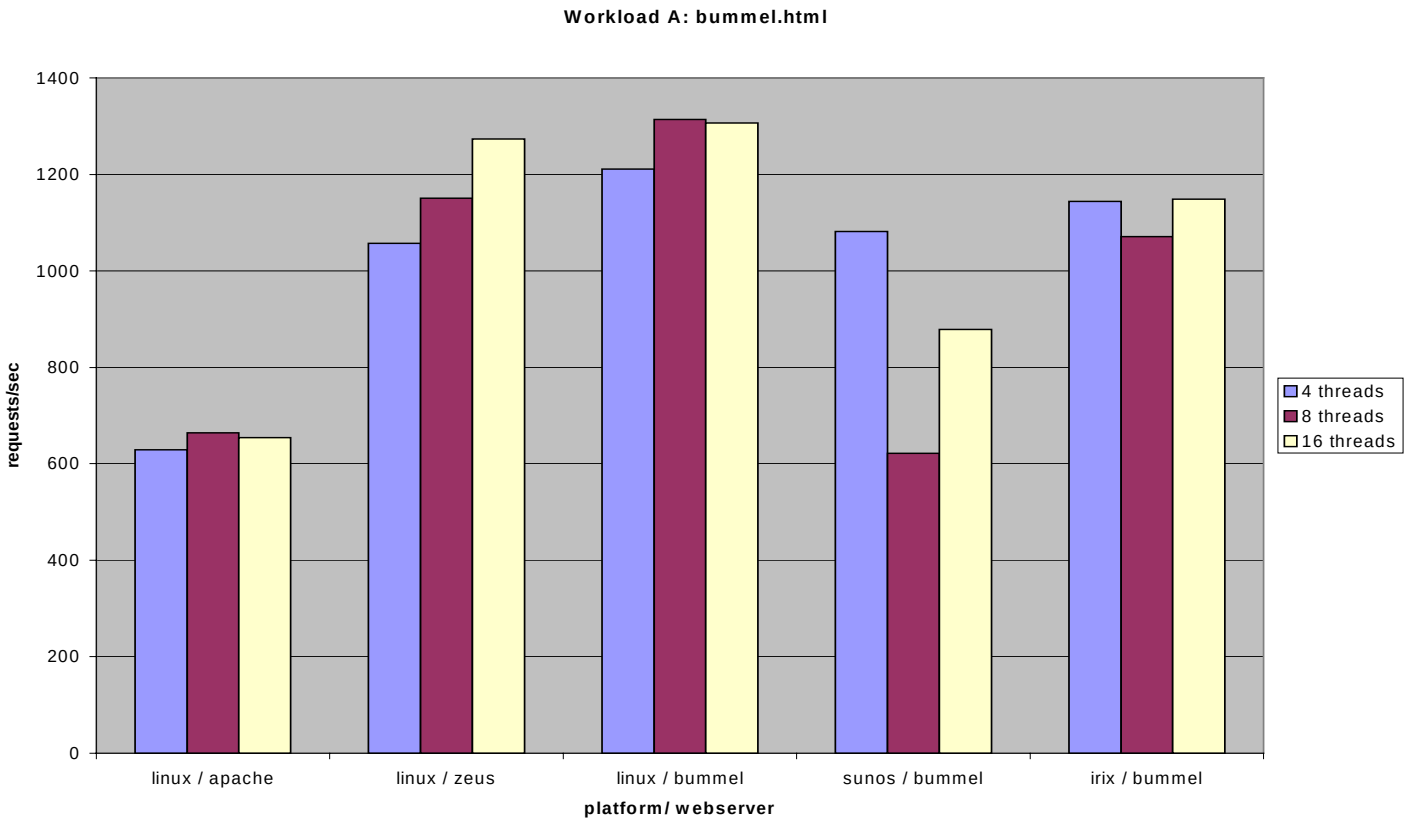
- Linux machines are in fact a very inexpensive alternative to traditional UNIX servers/ workstations; especially for web-serving.
- Today's web-servers achieve a performance of at least 50% of a leanest possible web-server (like Bummel), and thus are fast enough, given the flexibility they offer at the expense of speed.
- Zeus seems to be optimised for serving files that exist.
- good web-server performance is only needed when serving very small files (<10kb), as the need for fast response is normally cached by the Internet up-link, which becomes the bottleneck with increasing file-size. Saturation of a 100 Mbit/s link is reached at a request file size of approximately 25kb.

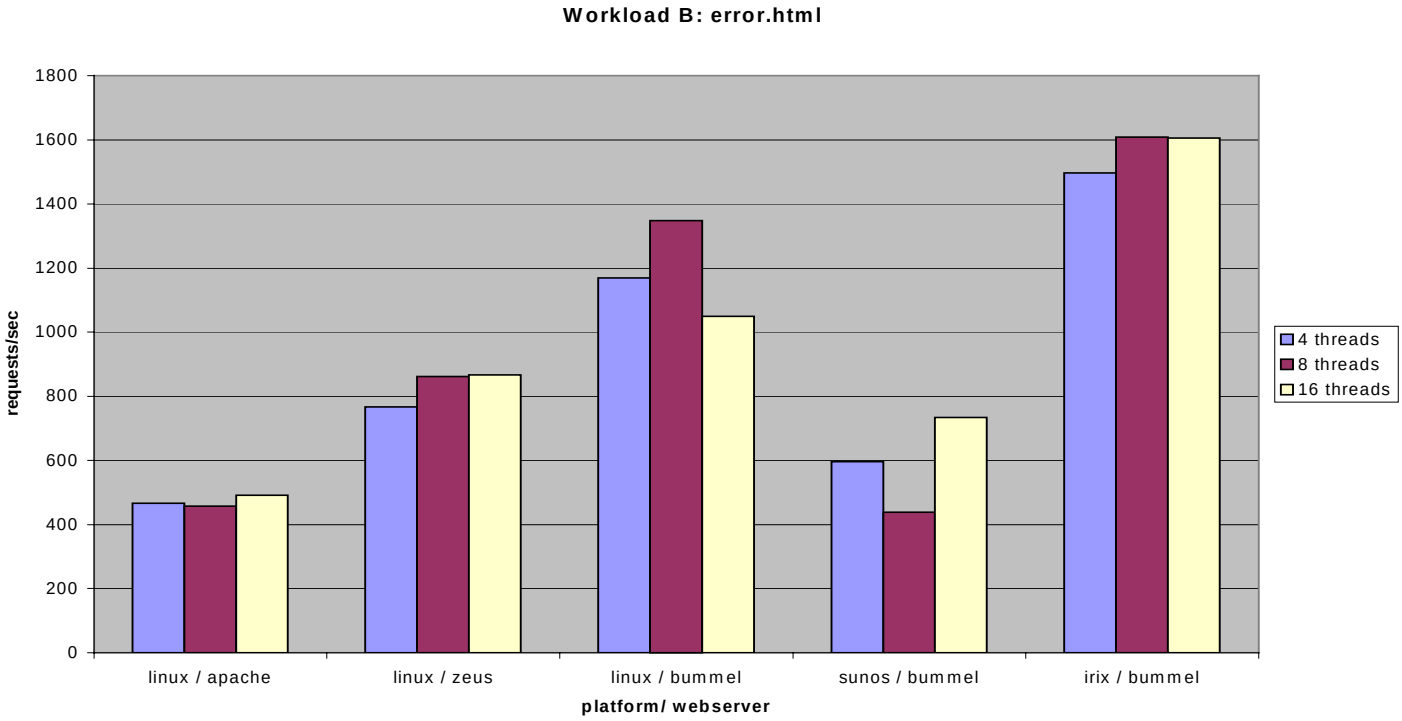
References

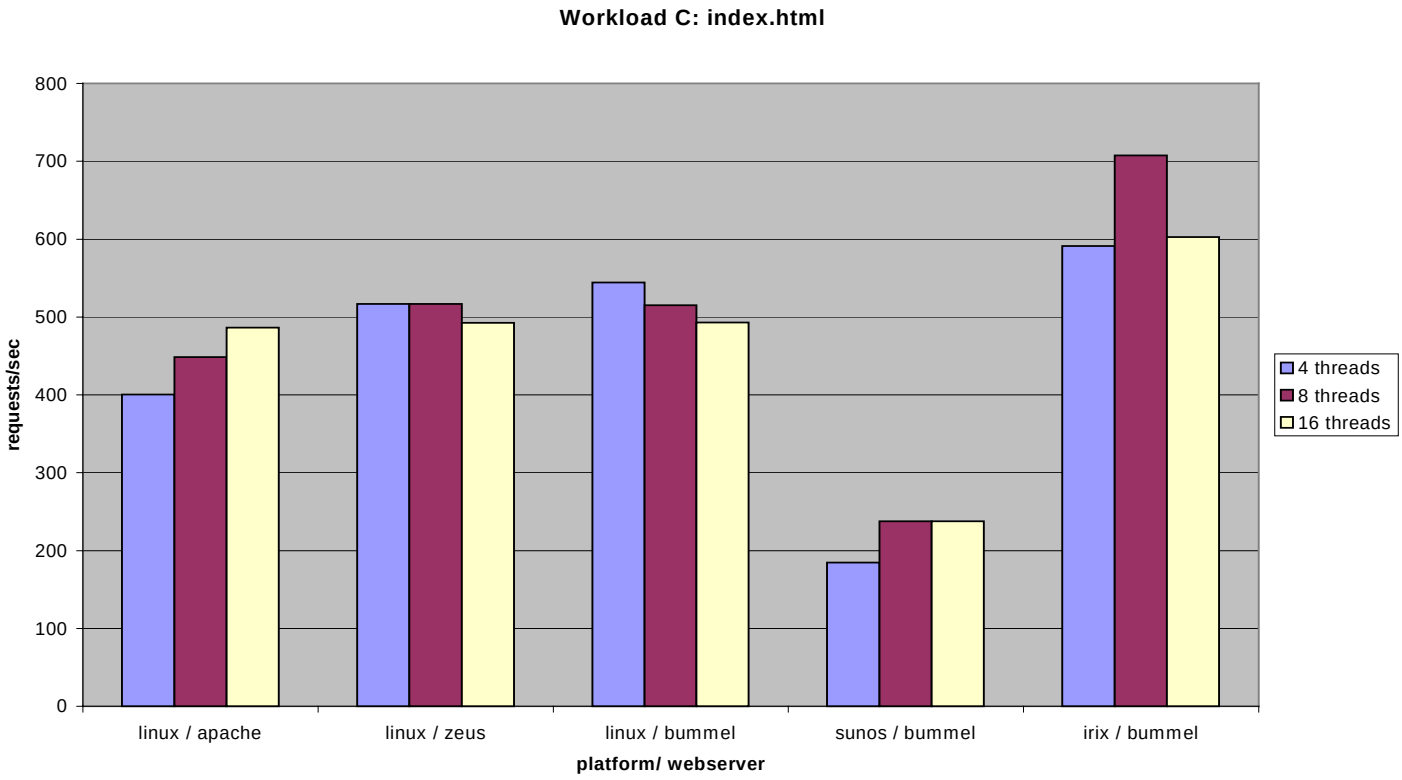
[1] http://www.apache.org/
[2] http://www.zeustech.com/
[3] http://www.longstreet.ch/bummel.tar.gz

Results

The performance results and the collected data are appended for analysis.




Workload B: error.html
1800
1600
1400
1200
1000
800
600
400
200
0
requests/sec
linux / apache
linux / zeus
linux / bummel
sunos / bummel
irix / bummel
platform/ webserver
4 threads
8 threads
16 threads


Workload C: index.html
800
700
600
500
400
300
200
100
0
requests/sec
linux / apache
linux / zeus
linux / bummel
sunos / bummel
irix / bummel
platform/ webserver
4 threads
8 threads
16 threads

Gathered performance data

platform / webserver / file / # threads	run # 1	2	3	4	5	mean	peak	deviation
irix / bummel / bummel.html / 4	1210	1182	1183	1160	987	1144	1210	63
irix / bummel / bummel.html / 8	1176	1162	937	1066	1015	1071	1176	78
irix / bummel / bummel.html / 16	1071	1256	1065	1190	1164	1149	1256	65
irix / bummel / error.html / 4	1530	1489	1505	1513	1447	1497	1530	23
irix / bummel / error.html / 8	1581	1608	1623	1617	1612	1608	1623	11
irix / bummel / error.html / 16	1594	1643	1632	1640	1521	1606	1643	39
irix / bummel / index.html / 4	618	550	605	589	596	591	618	18
irix / bummel / index.html / 8	714	708	713	693	711	708	714	6
irix / bummel / index.html / 16	646	559	556	574	677	602	677	47
linux / zeus / bummel.html / 4	1266	814	1063	1093	1050	1057	1266	100
linux / zeus / bummel.html / 8	1078	1158	1183	1282	1053	1151	1282	68
linux / zeus / bummel.html / 16	1360	1256	1249	1252	1251	1274	1360	35
linux / zeus / error.html / 4	725	621	853	863	775	767	863	76
linux / zeus / error.html / 8	843	889	849	882	849	863	889	18
linux / zeus / error.html / 16	848	889	858	886	857	868	889	16
linux / zeus / index.html / 4	505	491	522	542	524	517	542	15
linux / zeus / index.html / 8	516	521	515	520	512	517	521	3
linux / zeus / index.html / 16	493	491	492	494	495	493	495	1
linux / bummel / bummel.html / 4	1346	1297	1133	1183	1098	1211	1346	88
linux / bummel / bummel.html / 8	1394	1338	1335	1161	1341	1314	1394	61
linux / bummel / bummel.html / 16	1336	1305	1300	1310	1281	1307	1336	13
linux / bummel / error.html / 4	1283	1101	1136	1185	1137	1169	1283	52
linux / bummel / error.html / 8	1536	1324	1322	1320	1239	1348	1536	75
linux / bummel / error.html / 16	896	1216	1066	1324	745	1050	1324	183
linux / bummel / index.html / 4	542	552	540	549	539	544	552	5
linux / bummel / index.html / 8	517	516	517	511	514	515	517	2
linux / bummel / index.html / 16	501	489	489	493	493	493	501	3
linux / apache / bummel.html / 4	563	639	640	652	652	629	652	26
linux / apache / bummel.html / 8	697	675	608	670	673	665	697	22
linux / apache / bummel.html / 16	676	678	637	678	603	654	678	27
linux / apache / error.html / 4	499	475	427	432	494	465	499	29
linux / apache / error.html / 8	432	454	532	456	414	457	532	30
linux / apache / error.html / 16	501	560	479	429	489	492	560	31
linux / apache / index.html / 4	369	393	451	409	382	401	451	23
linux / apache / index.html / 8	444	471	466	429	434	449	471	16
linux / apache / index.html / 16	473	497	492	495	476	486	497	10
sunos / bummel / bummel.html / 4	1392	1344	44	1315	1312	1081	1392	415
sunos / bummel / bummel.html / 8	48	1309	392	1310	48	621	1310	550
sunos / bummel / bummel.html / 16	1485	1411	82	86	1328	878	1485	635
sunos / bummel / error.html / 4	1288	146	61	1340	147	596	1340	574
sunos / bummel / error.html / 8	46	531	1402	147	61	437	1402	423
sunos / bummel / error.html / 16	1292	135	63	777	1399	733	1399	507
sunos / bummel / index.html / 4	69	338	156	69	292	185	338	104
sunos / bummel / index.html / 8	295	59	275	291	60	196	295	109
sunos / bummel / index.html / 16	265	272	61	274	315	237	315	70