SOLARFLARE
A XILINX COMPANY

Cloud Onload® NGINX Proxy Cookbook

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

A list of patents associated with this product is at http://www.solarflare.com/patent

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright

© Copyright 2019 Xilinx, Inc. Xilinx, the Xilinx logo, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

SF-122227-CD

Issue 2

Table of Contents

1 Introduction . . . 1
- 1.1 About this document . . . 1
- 1.2 Intended audience . . . 2
- 1.3 Registration and support . . . 2
- 1.4 Download access . . . 2
- 1.5 Further reading . . . 2

2 Overview . . . 3
- 2.1 NGINX overview . . . 3
- 2.2 Wrk2 overview . . . 4
- 2.3 Cloud Onload overview . . . 4

3 Summary of benchmarking . . . 6
- 3.1 Overview of NGINX benchmarking . . . 6
- 3.2 Architecture for NGINX benchmarking . . . 7
- 3.3 NGINX benchmarking process . . . 8

4 Evaluation . . . 10
- 4.1 General server setup . . . 10
- 4.2 wrk2 client (on Load server) . . . 11
- 4.3 NGINX backend webservers (on Load server) . . . 12
  - Static files for webservers . . . 13
- 4.4 NGINX proxy (on Proxy server) . . . 13
- 4.5 Graphing the benchmarking results . . . 16

5 Benchmark results . . . 17
- 5.1 Results . . . 18
  - Connections per second . . . 18
  - Requests per second . . . 19
  - Throughput . . . 20
  - Latency . . . 21
- 5.2 Analysis . . . 22
  - Connections per second . . . 22
  - Requests per second . . . 22
  - Throughput . . . 22
  - Latency . . . 22

© Copyright 2019 Xilinx, Inc

A Cloud Onload profiles 23
A.1 The wrk-profile Cloud Onload profile 23
A.2 The nginx-server Cloud Onload profile 24
A.3 The nginx-proxy Cloud Onload profiles 25
The nginx-proxy-balanced profile. 25
The nginx-proxy-performance profile 25
The nginx-proxy-config profile fragment 26
The reverse-proxy-throughput profile fragment. 28

B Installation and configuration 30
B.1 Installing NGINX. 30
Installation 30
B.2 Installing wrk2 32
Installation 32
B.3 Installing Cloud Onload. 32

© Copyright 2019 Xilinx, Inc

1 Introduction

This chapter introduces you to this document. See:

- About this document on page 1
- Intended audience on page 2
- Registration and support on page 2
- Download access on page 2
- Further reading on page 2.

1.1 About this document

This document is the *NGINX Proxy Cookbook* for Cloud Onload. It gives procedures for technical staff to configure and run tests, to benchmark NGINX as a proxy server utilizing Solarflare's Cloud Onload and Solarflare NICs.

This document contains the following chapters:

- Introduction on page 1 (this chapter) introduces you to this document.
- Overview on page 3 gives an overviews of the software distributions used for this benchmarking.
- Summary of benchmarking on page 6 summarizes how the performance of NGINX has been benchmarked, both with and without Cloud Onload, to determine what benefits might be seen.
- Evaluation on page 10 describes how the performance of the test system is evaluated.
- Benchmark results on page 17 presents the benchmark results that are achieved.

and the following appendixes:

- Cloud Onload profiles on page 23 contains the Cloud Onload profiles used for this benchmarking.
- Installation and configuration on page 30 describes how to install and configure the software distributions used for this benchmarking.

 © Copyright 2019 Xilinx, Inc

1.2 Intended audience

The intended audience for this *NGINX Proxy Cookbook* are:

- software installation and configuration engineers responsible for commissioning and evaluating this system
- system administrators responsible for subsequently deploying this system for production use.

1.3 Registration and support

Support is available from support@solarflare.com.

1.4 Download access

Cloud Onload can be downloaded from: https://support.solarflare.com/.

Solarflare drivers, utilities packages, application software packages and user documentation can be downloaded from: https://support.solarflare.com/.

The scripts and Cloud Onload profiles used for this benchmarking are available on request from support@solarflare.com.

Please contact your Solarflare sales channel to obtain download site access.

1.5 Further reading

For advice on tuning the performance of Solarflare network adapters, see the following:

- *Solarflare Server Adapter User Guide* (SF-103837-CD).
  This is available from: https://support.solarflare.com/.

For more information about Cloud Onload, see the following:

- *Onload User Guide* (SF-104474-CD).
  This is available from: https://support.solarflare.com/.

© Copyright 2019 Xilinx, Inc

2 Overview

This chapter gives an overview of the software distributions used for this benchmarking. See:

- NGINX overview on page 3
- Wrk2 overview on page 4
- Cloud Onload overview on page 4.

2.1 NGINX overview

Open source NGINX [engine x] is an HTTP and reverse proxy server, a mail proxy server, and a generic TCP/UDP proxy server.

NGINX Plus is a software load balancer, web server, and content cache built on top of open source NGINX. NGINX has exclusive enterprise-grade features beyond what's available in the open source offering, including session persistence, configuration via API, and active health checks.

Open source NGINX is used for this benchmarking.

NGINX is heavily network dependent by design, so its performance can be significantly improved through enhancements to the underlying networking layer.

 © Copyright 2019 Xilinx, Inc

2.2 Wrk2 overview

Wrk2 is a modern HTTP benchmarking tool capable of generating significant load when run on a single multi-core CPU. It combines a multithreaded design with scalable event notification systems such as epoll and kqueue.

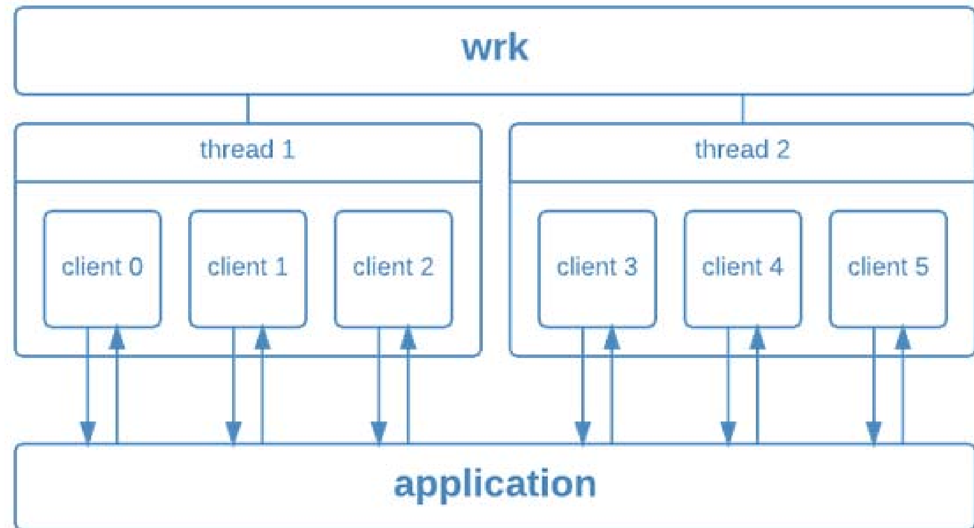

Figure 1: Wrk2 architecture

2.3 Cloud Onload overview

Cloud Onload is a high performance network stack from Solarflare (https://www.solarflare.com/) that dramatically reduces latency, improves CPU utilization, eliminates jitter, and increases both message rates and bandwidth. Cloud Onload runs on Linux and supports the TCP network protocol with a POSIX compliant sockets API and requires no application modifications to use. Cloud Onload achieves performance improvements in part by performing network processing at user-level, bypassing the OS kernel entirely on the data path.

Cloud Onload is a shared library implementation of TCP, which is dynamically linked into the address space of the application. Using Solarflare network adapters, Cloud Onload is granted direct (but safe) access to the network. The result is that the application can transmit and receive data directly to and from the network, without any involvement of the operating system. This technique is known as “kernel bypass”.

 © Copyright 2019 Xilinx, Inc

When an application is accelerated using Cloud Onload it sends or receives data without access to the operating system, and it can directly access a partition on the network adapter.

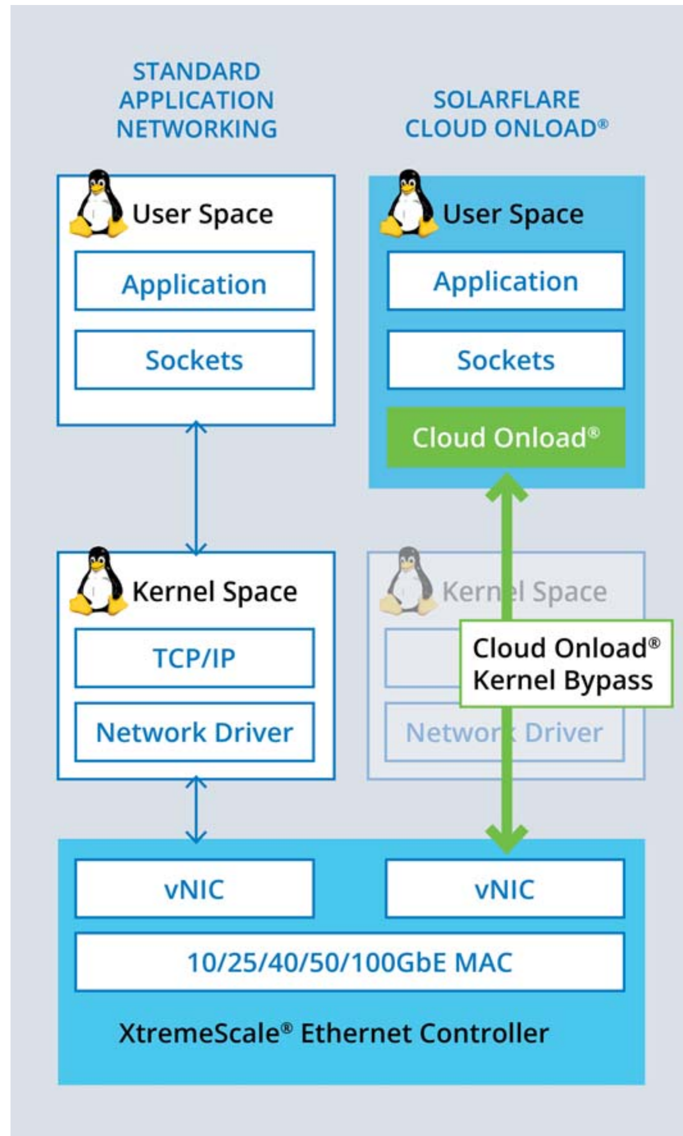

Figure 2: Cloud Onload architecture

© Copyright 2019 Xilinx, Inc

3 Summary of benchmarking

This chapter summarizes how the performance of NGINX as a proxy server has been benchmarked, both with and without Cloud Onload, to determine what benefits might be seen. See:

- Overview of NGINX benchmarking on page 6
- Architecture for NGINX benchmarking on page 7
- NGINX benchmarking process on page 8.

3.1 Overview of NGINX benchmarking

The NGINX benchmarking uses two servers:

- The *load server* runs multiple instances of wrk2 to generate requests, and multiple instances of NGINX webservers to service requests.
- The *proxy server* runs multiple instances of NGINX acting as a proxy. It receives the requests that originate from wrk2 on the load server, and proxies those requests to an NGINX webserver on the load server.

Various benchmark tests are run, with the NGINX proxy using the Linux kernel network stack.

The tests are then repeated, using Cloud Onload to accelerate the NGINX proxy. Two different Cloud Onload profiles are used, that have different priorities:

- The balanced profile gives excellent throughput, with low latency. It has reduced CPU usage at lower traffic rates.
- The performance profile is latency focused. It constantly polls for network events to achieve the lowest latency possible, and so has higher CPU usage.

The results using the kernel network stack are compared with the results using the two different Cloud Onload profiles.

 © Copyright 2019 Xilinx, Inc

3.2 Architecture for NGINX benchmarking

Benchmarking was performed with two Dell R640 servers, with the following specification:

Server	Dell R640
Memory	192GB
NICs	2 × X2541 (single port 100G): • Each NIC is affinitized to a separate NUMA node.
CPU	2 × Intel® Xeon® Gold 6148 CPU @ 2.40GHz: • Each CPU is on a separate NUMA node • There are 20 cores per CPU • Hyperthreading is enabled to give 40 hyperthreads per NUMA node
OS	Red Hat Enterprise Linux Server release 7.6 (Maipo)
Software	NGINX 1.17 wrk2 4.0.0

Each server is configured to leave as many CPUs as possible available for the application being benchmarked.

Each server has 2 NUMA nodes. 2 Solarflare NICs are fitted, each affinitized to a separate NUMA node, and connected directly to the corresponding NIC in the other server:

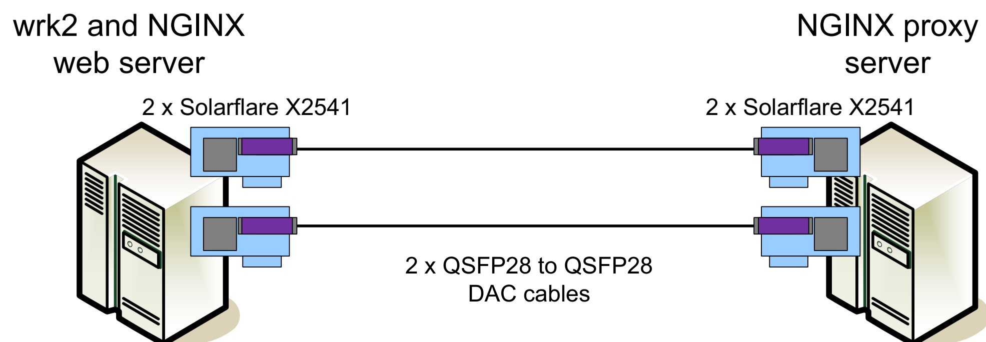

Figure 3: Architecture for NGINX benchmarking

© Copyright 2019 Xilinx, Inc

3.3 NGINX benchmarking process

These are the high-level steps we followed to complete benchmarking with NGINX:

- Install and test NGINX on both servers.
- Install wrk2 on the first server.
- Start NGINX web servers on the first server.

  All iterations of the test use the same configuration for consistency:
  - 40 NGINX web servers are used.
  - Each web server runs a single NGINX worker process.
  - Each NGINX worker process is assigned to a dedicated CPU, distributed across the NUMA nodes.
  - Each NGINX worker process uses the NIC that is affinitized to the local NUMA node for its CPU.
  - Each NGINX worker process uses a dedicated port.
  - Each NGINX web server is accelerated by Cloud Onload, to maximize the responsiveness of the proxied server.
- Start NGINX proxy servers on the other server:
  - One NGINX proxy server is used per NUMA node on the server.
    The setup used has 2 NUMA nodes, and so 2 proxy servers are started.
  - The first iteration of the test uses a single worker process per proxy server.
- Start `wrk2` on the first server to generate load.

  All iterations of the test use the same configuration for consistency:
  - 20 `wrk2` processes are used.
  - Each `wrk2` process is assigned to a dedicated CPU, distributed across the NUMA nodes.
  - Each `wrk2` process uses the NIC that is affinitized to the local NUMA node for its CPU.
  - Each `wrk2` process is accelerated by Cloud Onload, to maximize the throughput of each connection going to the NGINX proxy server.
- Record the response rate of the proxied web server, as the number of requests per second.
- Increase the number of NGINX worker processes on each proxy server, and repeat the test.
  - Each NGINX worker process is assigned to a dedicated CPU, distributed across the NUMA nodes.
  - Each NGINX worker process uses the NIC that is affinitized to the local NUMA node for its CPU.

  Continue doing this until the number of NGINX worker processes is the same on both servers. For the setup used, this is 40 processes.

© Copyright 2019 Xilinx, Inc

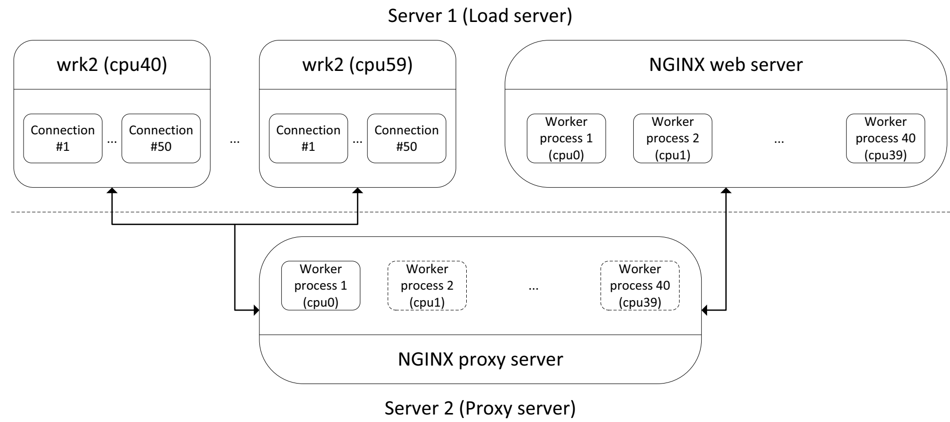

Figure 4: NGINX software usage

- Repeat all tests, accelerating the NGINX proxy with Cloud Onload.

These steps are detailed in the remaining chapters of this *Cookbook*.

The scripts and Cloud Onload profiles used for this benchmarking, that perform the above steps, are available on request from support@solarflare.com.

© Copyright 2019 Xilinx, Inc

4 Evaluation

This chapter describes how the performance of the test system is evaluated. See:

- General server setup on page 10
- wrk2 client (on Load server) on page 11
- NGINX backend webservers (on Load server) on page 12
- NGINX proxy (on Proxy server) on page 13
- Graphing the benchmarking results on page 16.

4.1 General server setup

Each server is setup using a script that does the following:

1 Create a file that makes new module settings:

```
cat > /etc/modprobe.d/proxy.conf <<EOL
options sfc \\
  performance_profile=throughput \\
  rss_cpus=20 \\
  rx_irq_mod_usec=90 \\
  irq_adapt_enable=N \\
  rx_ring=512 \\
  piobuf_size=0
options nf_conntrack_ipv4 \\
  hashsize=524288
EOL
```

NOTE: This script is required only when running the NGINX proxy server with the kernel network stack (i.e. without Cloud Onload).

2 Reload the drivers to pick up the new module settings:

```
onload_tool reload
```

3 Use the `network-throughput` tuned profile:

```
tuned-adm profile network-throughput
```

4 Stop various services:

```
systemctl stop irqbalance
systemctl stop iptables
systemctl stop firewalld
```

5 Increase the sizes of the OS receive and send buffers:

```
sysctl net.core.rmem_max=16777216 net.core.wmem_max=16777216
```

6 Configure huge pages:

```
sysctl vm.nr_hugepages=4096 > /dev/null
```

 © Copyright 2019 Xilinx, Inc

7 Ensure the connection tracking table is large enough:

```
sysctl net.netfilter.nf_conntrack_max=$(( $(sysctl --values
net.netfilter.nf_conntrack_buckets) * 4 )) > /dev/null
```

8 Increase the system-wide and per-process limits on the number of open files:

```
sysctl fs.file-max=8388608 > /dev/null
sysctl fs.nr_open=8388608 > /dev/null
```

9 Increase the range of local ports, so that the server can open lots of outgoing network connections:

```
sysctl -w net.ipv4.ip_local_port_range="2048 65535" > /dev/null
```

10 Increase the number of file descriptors that are available:

```
ulimit -n 8388608
```

11 Exclude from IRQ balancing the CPUs that are used for running the NGINX proxy servers. For example, to exclude CPUs 0 to 39:

```
IRQBALANCE_BANNED_CPUS=ff,ffffffff irqbalance --oneshot
```

4.2 wrk2 client (on Load server)

Set up 20 instances of wrk2, running on cores 40 to 59, and start them all. An example command line for the first instance (core 40) is below.

```
EF_CLUSTER_SIZE=10 \
        taskset -c 40 \
        onload -p wrk-profile.opf \
        /opt/wrk2/wrk \
        -R 500000 \
        -c 100 \
        -d 60 \
        -t 1 \
        http://192.168.0.101:1080/1024.bin
```

This example runs a *Requests per second* test using a payload size of 1024 bytes (HTTP GET with keepalive).

- The `taskset -c` parameter is changed for each instance, to use cores 40 to 59.
- Instances on the even cores (NUMA node 0) use the IP address for the NIC that is affinitized to NUMA node 0 on the proxy server.
- Instances on the odd cores (NUMA node 1) use the IP address for the NIC that is affinitized to NUMA node 1 on the proxy server.
- The port number is fixed at `1080`. This is the port listened to by the proxy server.
- `EF_CLUSTER_SIZE` is set to the number of `wrk2` instances which share the same IP address (i.e. 10 per NUMA node in this case).

 © Copyright 2019 Xilinx, Inc

4.3 NGINX backend webservers (on Load server)

Create a set of 40 backend webservers, with similar configuration for each webserver, and start them all. An example command line to start the first webserver (port 1050 of the NIC that is affinitized to NUMA node 0) is below:

```
onload -p nginx-server.opf sbin/nginx -c nginx-server-node0_1050.conf
```

The corresponding `nginx-server-node0_1050.conf` configuration file is shown below.

```
cat >nginx-server-node0_1050.conf <<EOL
worker_processes       1;
worker_rlimit_nofile   8388608;
worker_cpu_affinity    auto 00000000000000000000000000000001;

pid                    /var/run/nginx-node0_1050.pid;

events {
    multi_accept            off;
    accept_mutex            off;
    use                     epoll;
    worker_connections      200000;
}

error_log logs/error-node0_1050.log debug;

http {
    default_type            application/octet-stream;

    access_log              off;
    error_log               /dev/null crit;

    keepalive_timeout  300s;
    keepalive_requests 1000000;

    server {
        listen       192.168.0.100:1050 reuseport;
        server_name  localhost;

        open_file_cache max=100000 inactive=20s;
        open_file_cache_valid 30s;
        open_file_cache_errors off;

        location = /0 {
          return 204;
        }
        location / {
            root   html-node0_1050;
            index  index.html;
        }
        location = /upload {
          return 200 'Thank you';
        }
    }
}
EOL
```


© Copyright 2019 Xilinx, Inc

- The `worker_cpu_affinity` is changed for each instance, to use cores 0 to 39.
- Instances on the even cores (NUMA node 0) have the IP address in `http` → `server` → `listen` set to use the NIC that is affinitized to NUMA node 0, and the port address incrementing from `1050` upwards.
- Instances on the odd cores (NUMA node 1) have the IP address in `http` → `server` → `listen` set to use the NIC that is affinitized to NUMA node 1, and the port address also incrementing from `1050` upwards.
- The `pid` is changed for each instance.
- The `error_log` is changed for each instance
- The `server` → `location` → `root` is changed for each instance.

Static files for webservers

Each webserver serves static files from within the install directory, in a subdirectory that is configured by the `root` directive. Each webserver instance uses its own subdirectory, to avoid filesystem contention, and to model more closely a farm of separate servers.

The static files used range from 400B to 1MB. They were generated using `dd`. The example below creates the necessary files for the server that uses the above configuration file:

```
# mkdir -p /opt/nginx/html-node0_1050
# for payload in 400 1024 10240 32768 65536 102400 131072 262144 1024000
> do
> dd if=/dev/urandom of=/opt/nginx/html-node0_1050/$payload \
>    bs=$payload count=1 > /dev/null 2>&1
> done
```

4.4 NGINX proxy (on Proxy server)

Start various numbers of NGINX proxy worker processes (2, 8, 16, 24, 32 or 40), using either the kernel network stack, or one of two different Onload-accelerated network stacks. A total of 18 iterations are required.

Example command lines to start 16 worker processes are below:

- To start the proxy server with the kernel network stack, use the following:

  ```
  sbin/nginx -c nginx_proxy-node0_16.conf
  sbin/nginx -c nginx_proxy-node1_16.conf
  ```

- To start the proxy server with an Onload-accelerated network stack, use one of the following, for the two different Onload profiles under test:

  ```
  onload -p nginx-proxy-balanced.opf sbin/nginx -c nginx_proxy-node0_16.conf
  onload -p nginx-proxy-balanced.opf sbin/nginx -c nginx_proxy-node1_16.conf
  ```

  ```
  onload -p nginx-proxy-performance.opf sbin/nginx -c nginx_proxy-node0_16.conf
  onload -p nginx-proxy-performance.opf sbin/nginx -c nginx_proxy-node1_16.conf
  ```


© Copyright 2019 Xilinx, Inc

The corresponding `nginx_proxy-node0_16.conf` configuration file is shown below.

```
cat >nginx_proxy-node0_16.conf <<EOL
worker_processes       8;
worker_rlimit_nofile   8388608;
worker_cpu_affinity auto 00000000000000000101010101010101 ;

pid                    /var/run/nginx-node0_16.pid;

events {
    multi_accept            off;
    accept_mutex            off;
    use                     epoll;
    worker_connections      200000;
}

error_log logs/error-node0_16.log debug;

http {
    default_type            application/octet-stream;

    access_log              off;
    error_log               /dev/null crit;

    # “sendfile on” is used only for the kernel network stack.
    sendfile on;

    # Proxy buffering is disabled to avoid large file tests overflowing
    # buffers which leads to temporary file use. This particularly harms
    # Onload performance due to workers taking too long between epoll_wait
    # calls which are needed for frequent Onload stack polling.
    # Issue can also be avoided by ensuring sufficient buffering, e.g.
    # proxy_buffering         on;
    # proxy_buffers           192 8k;
    proxy_buffering         off;

    keepalive_timeout  300s;
    keepalive_requests 1000000;

    server {
        listen       192.168.0.101:1080 reuseport;
        server_name  localhost;

        error_page   500 502 503 504  /50x.html;

        location = /50x.html {
            root   html;
        }
        location / {
            proxy_pass http://backend;
            proxy_http_version 1.1;
            proxy_set_header Connection "";
        }
    }

    upstream backend {
```


© Copyright 2019 Xilinx, Inc

```
    server 192.168.0.100:1050 ;
    server 192.168.0.100:1051 ;
    server 192.168.0.100:1052 ;
    server 192.168.0.100:1053 ;
    server 192.168.0.100:1054 ;
    server 192.168.0.100:1055 ;
    server 192.168.0.100:1056 ;
    server 192.168.0.100:1057 ;
    server 192.168.0.100:1058 ;
    server 192.168.0.100:1059 ;
    server 192.168.0.100:1060 ;
    server 192.168.0.100:1061 ;
    server 192.168.0.100:1062 ;
    server 192.168.0.100:1063 ;
    server 192.168.0.100:1064 ;
    server 192.168.0.100:1065 ;
    server 192.168.0.100:1066 ;
    server 192.168.0.100:1067 ;
    server 192.168.0.100:1068 ;
    server 192.168.0.100:1069 ;
      keepalive 500;
  }
}
EOL
```

- The `worker_processes` is set to the number of worker processes which share the same NUMA node (i.e. half the number of worker processes in the test).
- The `worker_cpu_affinity` is set to use one core per worker process, all on the same NUMA node (alternate bits set in this case).

  For the corresponding `nginx_proxy-node1_16.conf` configuration file, the lower 16 bits are reversed:

  `worker_cpu_affinity auto 00000000000000001010101010101010 ;`
- Instances on the even cores (NUMA node 0) have the IP addresses set as follows:
  - `http` → `server` → `listen` is set to use the NIC that is affinitized to NUMA node 0, with the port number set to `1080`.
  - `http` → `upstream backend` → `server` is set to use the NIC that is affinitized to NUMA node 0 on the load server, with all port numbers in the range `1050-1069`.
- Instances on the odd cores (NUMA node 1) have the IP addresses set as follows:
  - `http` → `server` → `listen` is set to use the NIC that is affinitized to NUMA node 1, with the port number set to `1080`.
  - `http` → `upstream backend` → `server` is set to use the NIC that is affinitized to NUMA node 1 on the load server, with all port numbers in the range `1050-1069`.
- The `pid` is changed for each instance.
- The `error_log` is changed for each instance.

© Copyright 2019 Xilinx, Inc

4.5 Graphing the benchmarking results

The results from each pass of `wrk2` are now gathered and summed, so that they can be further analyzed. They are then transferred into an Excel spreadsheet, to create graphs from the data.

© Copyright 2019 Xilinx, Inc

5 Benchmark results

This chapter presents the benchmark results that are achieved. See:

- Results on page 18
- Analysis on page 22.

 © Copyright 2019 Xilinx, Inc

5.1 Results

Connections per second

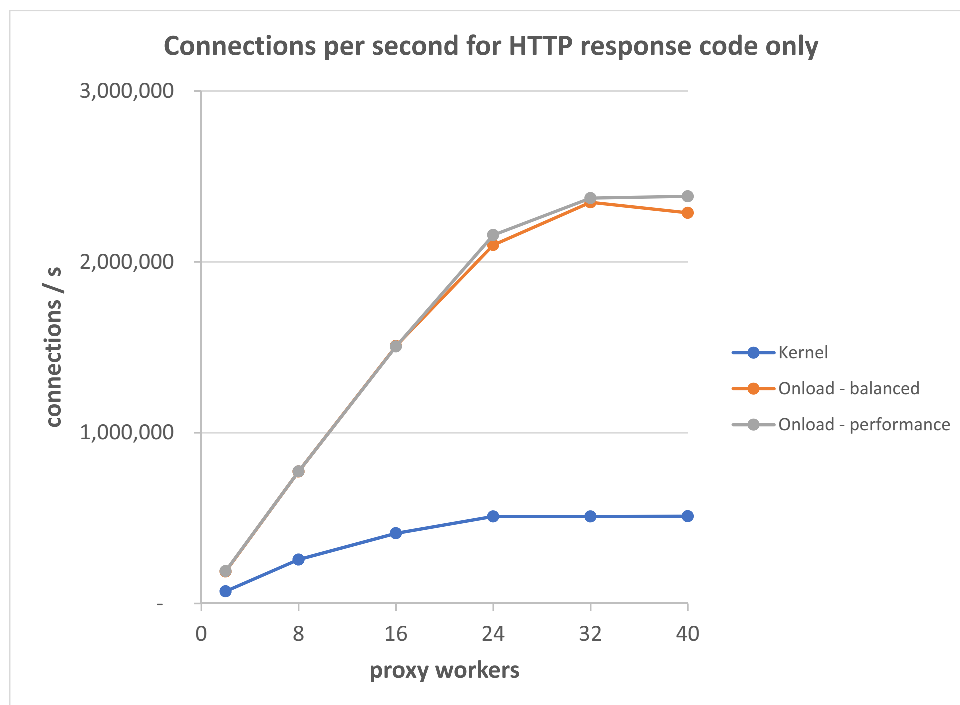

Figure 5: NGINX connections per second

Table 1 below shows the results that were used to plot the graph in Figure 5 above.

Table 1: Connections per second

Worker processes	Kernel	Onload balanced	Onload performance	Onload balanced gain	Onload performance gain
2	70957.35	187431.36	189503.89	164%	167%
8	257003.65	772494.71	772756.36	200%	200%
16	410795.52	1507558.1	1504750.5	266%	266%
24	509656.37	2098591.5	2157234.8	311%	323%
32	509396.24	2348479.4	2373220.1	361%	365%
40	511137.83	2287115.4	2383646.32	347%	366%

© Copyright 2019 Xilinx, Inc

Requests per second

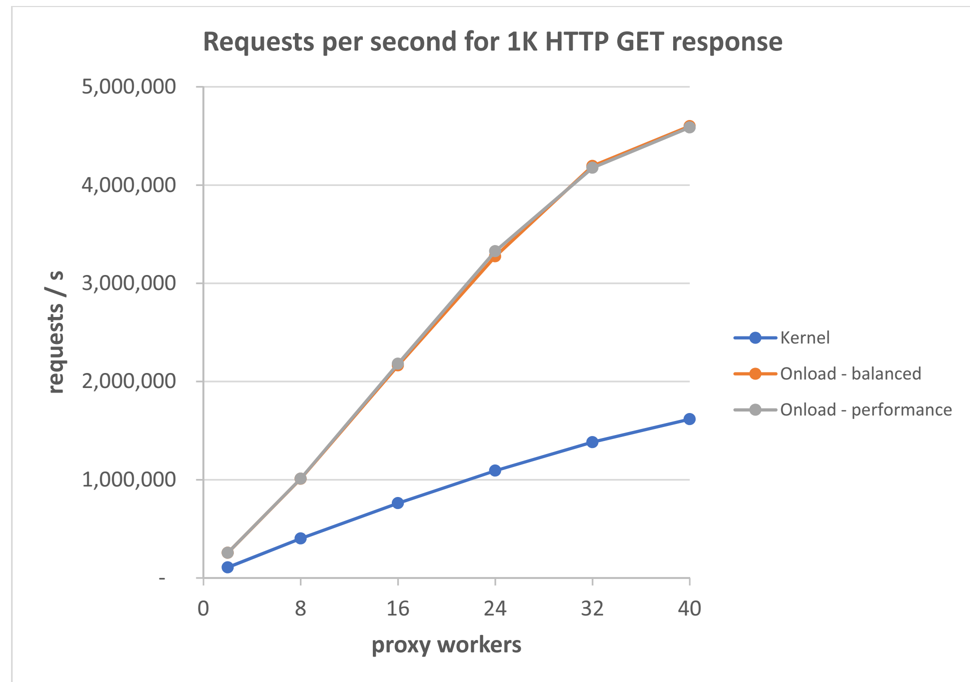

Figure 6: NGINX requests per second

Table 2 below shows the results that were used to plot the graph in Figure 6 above.

Table 2: Requests per second for 1KB

Worker processes	Kernel	Onload balanced	Onload performance	Onload balanced gain	Onload performance gain
2	107704.26	256210.07	256858.82	137%	138%
8	401871.84	1009312.55	1011250.65	151%	151%
16	761825.71	2163700.25	2181539.11	184%	186%
24	1092638.24	3273655.22	3326961.9	199%	204%
32	1382586.62	4194785.8	4175574.83	203%	202%
40	1616827.43	4599771.22	4586751.76	184%	183%

© Copyright 2019 Xilinx, Inc

Throughput

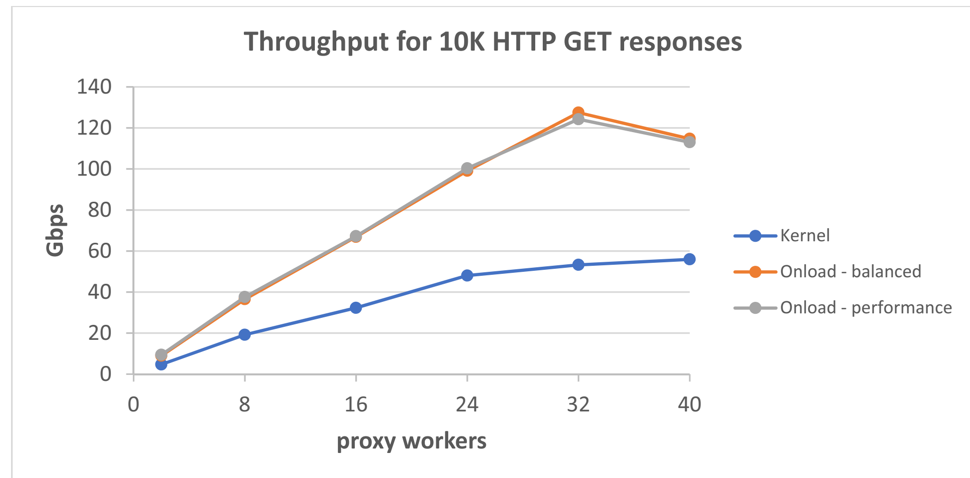

Figure 7: NGINX throughput

Table 3 below shows the results that were used to plot the graph in Figure 7 above.

Table 3: Throughput for 10K in Gbps

Worker processes	Kernel	Onload balanced	Onload performance	Onload balanced gain	Onload performance gain
2	4.71113728	8.96106496	9.4035968	90%	99%
8	19.1726387	36.51264512	37.5843226	90%	96%
16	32.3158016	66.90340864	67.2515686	107%	108%
24	48.0385434	99.10771712	100.310794	106%	108%
32	53.2519322	127.4857062	124.255437	139%	133%
40	55.9471002	114.7304346	113.028792	105%	102%

© Copyright 2019 Xilinx, Inc

Latency

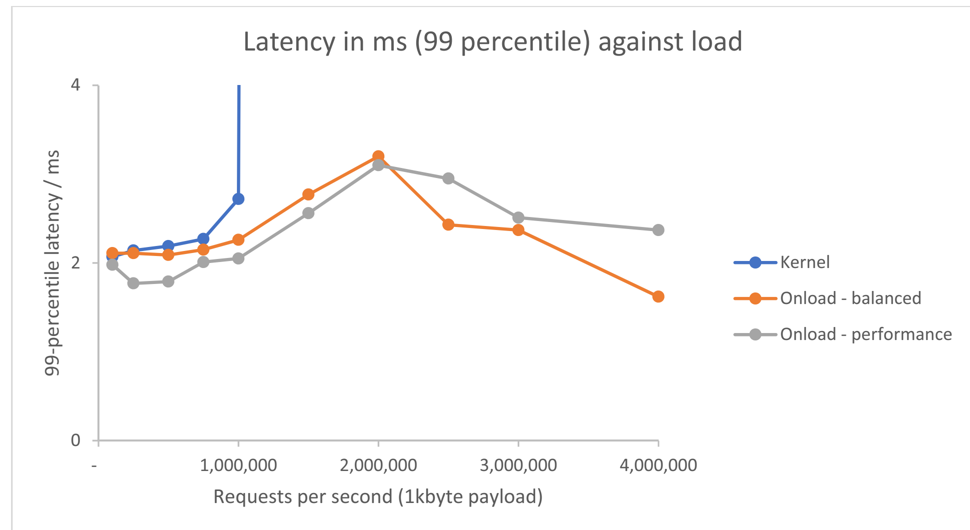

Figure 8: NGINX latency

Table 4 below shows the results that were used to plot the graph in Figure 8 above.

Table 4: Latency for 1KB

Requests per second	Kernel	Onload balanced	Onload performance	Onload balanced gain	Onload performance gain
100000	3100	2070	1980	49%	56%
250000	2000	2110	1750	-6%	14%
500000	2240	2080	1810	7%	23%
750000	2540	2150	1990	18%	27%
1000000	2680	2280	2040	17%	31%
1100000	2650	2210	2040	19%	29%
1200000	70010	2380	2150	Kernel cannot maintain requested packet rate. Gain is meaningless.	
1300000	2310000	2440	2220		
1400000	11190000	2690	2300		
1500000	16440000	2760	2410		
2000000	23450000	3220	2910		
2500000	29920000	2420	2580		
3000000	34870000	2360	2490		
4000000	40530000	1620	2370		

© Copyright 2019 Xilinx, Inc

5.2 Analysis

When compared with the kernel stack, Cloud Onload delivers significant improvements to all metrics.

Connections per second

The connections per second shows great improvement with Cloud Onload, peaking at an improvement of 366% over the kernel stack. With large numbers of proxy workers (32 to 40) the Cloud Onload performance levels out. This is most likely because the load server is unable to generate and measure any more traffic, but might be because the proxy server itself is saturated.

Requests per second

The requests per second also shows great improvement with Cloud Onload, peaking at an improvement of 204% over the kernel stack. With 40 worker processes, results continue to improve, indicating that further performance is available from Cloud Onload.

Throughput

The throughput shows significant improvement with Cloud Onload, peaking at an improvement of 139% over the kernel stack. With large numbers of proxy workers (32 to 40) the Cloud Onload performance again levels out, either because the load server cannot generate any more traffic, or because the proxy server itself is saturated.

Latency

The latency figures are as output by `wrk2`, and show the time from when the *should* have been sent (according to the configured packet rate), until when the packet was actually received. The 99 percentile figure is reported.

When the kernel stack packet rate is raised above 1.1 million requests per second, it can no longer maintain this rate. Jitter increases, the number of outliers exceeds 1%, and so the reported latency suddenly and dramatically increases. Any further small increase in load would make the server appear completely unresponsive to an end user.

In contrast, Cloud Onload continues to deliver low latency with 4 million requests per second, and is actually trending towards even lower latency. The stable and low value for the 99th percentile of latency indicates low jitter and predictable performance.

© Copyright 2019 Xilinx, Inc

A Cloud Onload profiles

This appendix contains the Cloud Onload profiles used for this benchmarking. See:

- The wrk-profile Cloud Onload profile on page 23
- The nginx-server Cloud Onload profile on page 24
- The nginx-proxy Cloud Onload profiles on page 25.

These profiles, along with the scripts used for this benchmarking, are available on request from support@solarflare.com.

A.1 The wrk-profile Cloud Onload profile

The `wrk-profile.opf` Cloud Onload profile is as follows:

```
onload_set EF_SOCKET_CACHE_MAX 40000
onload_set EF_TCP_TCONST_MSL 1
onload_set EF_TCP_FIN_TIMEOUT 15
onload_set EF_HIGH_THROUGHPUT_MODE 1
onload_set EF_LOG_VIA_IOCTL 1
onload_set EF_NO_FAIL 1
onload_set EF_UDP 0

#ensure sufficient resources
onload_set EF_MAX_PACKETS 205000
onload_set EF_MAX_ENDPOINTS 400000
onload_set EF_FDTABLE_SIZE 8388608
onload_set EF_USE_HUGE_PAGES 2
onload_set EF_MIN_FREE_PACKETS 50000

#environment variable can overwrite
onload_set EF_LOAD_ENV 1

#spinning configuration
onload_set EF_POLL_USEC 100000
onload_set EF_SLEEP_SPIN_USEC 50
onload_set EF_EPOLL_SPIN 1

#scalable filters with clustering for outgoing connections
onload_set EF_SCALABLE_FILTERS 'any=rss:active'
onload_set EF_SCALABLE_FILTERS_ENABLE 1
onload_set EF_CLUSTER_NAME 'load'
onload_set EF_CLUSTER_SIZE 12 #needs to overwritten by environment

#shared local ports to improve rate of socket recycling
onload_set EF_TCP_SHARED_LOCAL_PORTS_MAX 28000
onload_set EF_TCP_SHARED_LOCAL_PORTS 28000
onload_set EF_TCP_SHARED_LOCAL_PORTS_PER_IP 1
```


© Copyright 2019 Xilinx, Inc

```
onload_set EF_TCP_SHARED_LOCAL_PORTS_REUSE_FAST 1
onload_set EF_TCP_SHARED_LOCAL_PORTS_NO_FALLBACK 1

#epoll configuration
onload_set EF_UL_EPOLL 3
onload_set EF_EPOLL_MT_SAFE 1

#reduce transmit CPU load
onload_set EF_TX_PUSH 0
onload_set EF_PIO 0
onload_set EF_CTPIO 0

# Adjustments for potentially-lossy network environment
onload_set EF_TCP_INITIAL_CWND 14600
onload_set EF_DYNAMIC_ACK_THRESH 4
onload_set EF_TAIL_DROP_PROBE 1
onload_set EF_TCP_RCVBUF_MODE 1
```

A.2 The nginx-server Cloud Onload profile

The `nginx-server.opf` Cloud Onload profile is as follows:

```
onload_set EF_ACCEPTQ_MIN_BACKLOG 400
onload_set EF_SOCKET_CACHE_MAX 40000
onload_set EF_TCP_TCONST_MSL 1
onload_set EF_TCP_FIN_TIMEOUT 15
onload_set EF_TCP_SYNRECV_MAX 90000
onload_set EF_TCP_BACKLOG_MAX 400
onload_set EF_HIGH_THROUGHPUT_MODE 1
onload_set EF_LOG_VIA_IOCTL 1
onload_set EF_NO_FAIL 1
onload_set EF_UDP 0

#ensure sufficient resources
onload_set EF_MAX_PACKETS 205000
onload_set EF_MAX_ENDPOINTS 400000
onload_set EF_USE_HUGE_PAGES 2
onload_set EF_MIN_FREE_PACKETS 50000

#epoll configuration
onload_set EF_UL_EPOLL 3
onload_set EF_EPOLL_MT_SAFE 1

#don't use clustering when SO_REUSEPORT is set
onload_set EF_CLUSTER_IGNORE 1

#environment variable can overwrite
onload_set EF_LOAD_ENV 1

#spinning configuration
onload_set EF_POLL_USEC 100000
onload_set EF_SLEEP_SPIN_USEC 50
onload_set EF_EPOLL_SPIN 1

#reduce transmit CPU load
onload_set EF_TX_PUSH 0
```

 © Copyright 2019 Xilinx, Inc

```
onload_set EF_PIO 0
onload_set EF_CTPIO 0

# Adjustments for potentially-lossy network environment
onload_set EF_TCP_INITIAL_CWND 14600
onload_set EF_DYNAMIC_ACK_THRESH 4
onload_set EF_TAIL_DROP_PROBE 1
onload_set EF_TCP_RCVBUF_MODE 1
```

A.3 The nginx-proxy Cloud Onload profiles

There are two nginx-proxy Cloud Onload profiles.

- The balanced profile gives excellent throughput, with low latency. It has reduced CPU usage at lower traffic rates.
- The performance profile is latency focused. It constantly polls for network events to achieve the lowest latency possible, and so has higher CPU usage.

The differences between these profiles are minor, and are in the profile files. See:

- The nginx-proxy-balanced profile on page 25
- The nginx-proxy-performance profile on page 25.

The majority of the settings are common to both profiles, and are in separate shared files that each profile sources or includes. See:

- The nginx-proxy-config profile fragment on page 26.
- The reverse-proxy-throughput profile fragment on page 28.

The nginx-proxy-balanced profile

The `nginx-proxy-balanced.opf` Cloud Onload profile is as follows:

```
. ${PROXY_CONFIG_DIR}/nginx-proxy-config.opf-fragment
onload_import ${PROXY_CONFIG_DIR}/reverse-proxy-throughput.opf-fragment
```

The nginx-proxy-performance profile

The `nginx-proxy-performance.opf` Cloud Onload profile is as follows:

```
. ${PROXY_CONFIG_DIR}/nginx-proxy-config.opf-fragment
onload_set EF_TX_PUSH 1
onload_set EF_SLEEP_SPIN_USEC 0
onload_import ${PROXY_CONFIG_DIR}/reverse-proxy-throughput.opf-fragment
```

 © Copyright 2019 Xilinx, Inc

The nginx-proxy-config profile fragment

The nginx-proxy-config`.opf-fragment` file, sourced by both the above profiles, is as follows:

```
## User may supply the following environment variables:
#
#   NGINX_NUM_WORKERS        - the number of workers that nginx is
#                              configured to use. Overrides value
#                              automatically detected from nginx
#                              configuration
#
set -o pipefail

# For diagnostic output
module="nginx profile"

# Regular expressions to match nginx config directives
worker_processes_pattern="/(^|;)\s*worker_processes\s+(\w+)\s*;/"
include_pattern="/(^|;)\s*include\s+(\S+)\s*;/"

# Identify the config file that nginx would use
identify_config_file() {
  local file

  # Look for a -c option
  local state="IDLE"
  for option in "$@"
  do
    if [ "$state" = "MINUS_C" ]
    then
      file=$option
      state="FOUND"
    elif [ "$option" = "-c" ]
    then
      state="MINUS_C"
    fi
  done

  # Extract the compile-time default if config not specified on command line
  if [ "$state" != "FOUND" ]
  then
    file=$($1 -h 2>&1 | perl -ne 'print $1 if '"$worker_processes_pattern")
  fi

  [ -f "$file" ] && echo $file
}

# Recursively look in included config files for a setting of worker_processes.
# NB If this quantity is set in more than one place then the wrong setting might
# be found, but this would be invalid anyway and is rejected by Nginx.
read_config_file() {
  local setting
  local worker_values=$(perl -ne 'print "$2 " if '"$worker_processes_pattern" $1)
  local include_values=$(perl -ne 'print "$2 " if '"$include_pattern" $1)

  # First look in included files
```


© Copyright 2019 Xilinx, Inc

```
  for file in $include_values
  do
    local possible=$(read_config_file $file)
    if [ -n "$possible" ]
    then
      setting=$possible
    fi
  done

  # Then look in explicit settings at this level
  for workers in $worker_values
  do
    setting=$workers
  done
  echo $setting
}

# Method to parse configuration files directly
try_config_files() {
  local config_file=$(identify_config_file "$@")
  [ -n "$config_file" ] && read_config_file $config_file
}

# Method to parse configuration via nginx, if supported
try_nginx_minus_t() {
  "$@" -T | perl -ne 'print "$2" if '"$worker_processes_pattern"
}

# Method to parse configuration via tengine, if supported
try_tengine_minus_d() {
  "$@" -d | perl -ne 'print "$2" if '"$worker_processes_pattern"
}

# Determine the number of workers nginx will use
determine_worker_processes() {
  # Prefer nginx's own parser, if available, for robustness
  local workers=$(try_nginx_minus_t "$@" || try_tengine_minus_d "$@" || try_config_files
"$@")
  if [ "$workers" = "auto" ]
  then
    # Default to the number of process cores
    workers=$(nproc)
  fi
  echo $workers
}

# Define the number of workers
num_workers=${NGINX_NUM_WORKERS:-$(determine_worker_processes "$@")}
if ! [ -n "$num_workers" ]; then
  fail "ERROR: Environment variable NGINX_NUM_WORKERS is not set and worker count cannot
be determined from nginx configuration"
fi
log "$module: configuring for $num_workers workers"
```

The reverse-proxy-throughput profile fragment

The `reverse-proxy-throughput.opf-fragment` file, included by both the above profiles, is as follows:

```
## User may supply the following environment variables:
#
#   NGINX_NUM_WORKERS       - the number of workers that nginx is
#                             configured to use. Overrides value
#                             automatically detected from nginx
#                             configuration
#
set -o pipefail

# For diagnostic output
module="nginx profile"

# Regular expressions to match nginx config directives
worker_processes_pattern="/(^|;)\s*worker_processes\s+(\w+)\s*;/"
include_pattern="/(^|;)\s*include\s+(\S+)\s*;/"

# Identify the config file that nginx would use
identify_config_file() {
  local file

  # Look for a -c option
  local state="IDLE"
  for option in "$@"
  do
    if [ "$state" = "MINUS_C" ]
    then
      file=$option
      state="FOUND"
    elif [ "$option" = "-c" ]
    then
      state="MINUS_C"
    fi
  done

  # Extract the compile-time default if config not specified on command line
  if [ "$state" != "FOUND" ]
  then
    file=$($1 -h 2>&1 | perl -ne 'print $1 if '"$worker_processes_pattern")
  fi

  [ -f "$file" ] && echo $file
}

# Recursively look in included config files for a setting of worker_processes.
# NB If this quantity is set in more than one place then the wrong setting might
# be found, but this would be invalid anyway and is rejected by Nginx.
read_config_file() {
  local setting
  local worker_values=$(perl -ne 'print "$2 " if '"$worker_processes_pattern" $1)
  local include_values=$(perl -ne 'print "$2 " if '"$include_pattern" $1)

  # First look in included files
```


© Copyright 2019 Xilinx, Inc

```
  for file in $include_values
  do
    local possible=$(read_config_file $file)
    if [ -n "$possible" ]
    then
      setting=$possible
    fi
  done

  # Then look in explicit settings at this level
  for workers in $worker_values
  do
    setting=$workers
  done
  echo $setting
}

# Method to parse configuration files directly
try_config_files() {
  local config_file=$(identify_config_file "$@")
  [ -n "$config_file" ] && read_config_file $config_file
}

# Method to parse configuration via nginx, if supported
try_nginx_minus_t() {
  "$@" -T | perl -ne 'print "$2" if '"$worker_processes_pattern"
}

# Method to parse configuration via tengine, if supported
try_tengine_minus_d() {
  "$@" -d | perl -ne 'print "$2" if '"$worker_processes_pattern"
}

# Determine the number of workers nginx will use
determine_worker_processes() {
  # Prefer nginx's own parser, if available, for robustness
  local workers=$(try_nginx_minus_t "$@" || try_tengine_minus_d "$@" || try_config_files
"$@")
  if [ "$workers" = "auto" ]
  then
    # Default to the number of process cores
    workers=$(nproc)
  fi
  echo $workers
}

# Define the number of workers
num_workers=${NGINX_NUM_WORKERS:-$(determine_worker_processes "$@")}
if ! [ -n "$num_workers" ]; then
  fail "ERROR: Environment variable NGINX_NUM_WORKERS is not set and worker count cannot
be determined from nginx configuration"
fi
log "$module: configuring for $num_workers workers"
```

 © Copyright 2019 Xilinx, Inc

B Installation and configuration

This appendix describes how to install and configure the software distributions used for this benchmarking. See:

- Installing NGINX on page 30
- Installing wrk2 on page 32

B.1 Installing NGINX

This section describes how to install and configure NGINX.

Installation

NOTE: For a reference description of how to install NGINX, see https://docs.nginx.com/nginx/admin-guide/installing-nginx/installing-nginx-open-source/.

In summary:

1 If you already have an old NGINX installation on your system:

a) Back up your configs and logs:

```
# cp -a /etc/nginx /etc/nginx-plus-backup
# cp -a /var/log/nginx /var/log/nginx-plus-backup
```

b) Remove the old installation:

```
# rm -rf /opt/nginx
```

2 Create a new NGINX directory:

```
mkdir -p /opt/nginx
```

3 Change to a temporary directory:

```
cd $( mktemp -d )
```

4 Clone NGINX from its git repository:

```
git clone https://github.com/nginx/nginx .
```

5 Configure NGINX:

```
./auto/configure --prefix=/opt/nginx
```

6 Make and install NGINX:

```
make install
```

 © Copyright 2019 Xilinx, Inc

7 Check the NGINX binary version to ensure that you have NGINX installed correctly:

```
# nginx -v
nginx version: nginx/1.17
```

8 Start NGINX:

```
# systemctl start nginx
```

or just:

```
# nginx
```

9 Verify access to Web Server

10.20.128.35

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

© Copyright 2019 Xilinx, Inc

B.2 Installing wrk2

This section describes how to install and configure `wrk2`.

Installation

NOTE: For a reference description of how to install `wrk2`, see: https://github.com/giltene/wrk2/wiki/Installing-wrk2-on-Linux.

In summary:

1. If the build tools are not already installed, install them:

```
# yum groupinstall 'Development Tools'
```

2. If the OpenSSL dev libs are not already installed, install them:

```
# yum install -y openssl-devel
```

3. If git is not already installed, install it:

```
# yum install -y git
```

4. Create a directory to hold wrk2:

```
# mkdir -p Onload_Testing/WRK2
# cd Onload_Testing/WRK2
```

5. Use git to download wrk2:

```
# git clone https://github.com/giltene/wrk2.git
```

6. Build wrk2:

```
# cd wrk2
# make
```

7. Copy the wrk2 executable to a location on your PATH. For example:

```
# cp wrk2 /usr/local/bin
```

B.3 Installing Cloud Onload

For instructions on how to install and configure Cloud Onload, refer to the *Onload User Guide* (SF-104474-CD). This is available from https://support.solarflare.com/.

© Copyright 2019 Xilinx, Inc