Optimization Guide

3rd Generation Intel® Xeon® Scalable Processors

Nginx* HTTPs with Crypto-NI Tuning Guide on 3rd Generation Intel® Xeon® Scalable Processors

Contents

Revision Record 2

1. Nginx* Introduction 3

2. BIOS Settings 5

3. Linux* Optimization 5

3.1. System Optimization 5

3.2. Network Optimization 5

4. Crypto-NI installation 7

4.1. IPP Cryptography Library and intel-ipsec-mb Installation 7

4.2. OpenSSL* and QAT_Engine Installation 7

5. Nginx installation and Optimization Settings 8

5.1. Nginx Installation 8

5.2. Optimized Settings for Nginx 9

6. Introduction to Related Tools 10

6.1. Web Server Stress Testing Tools 10

7. Conclusion 11

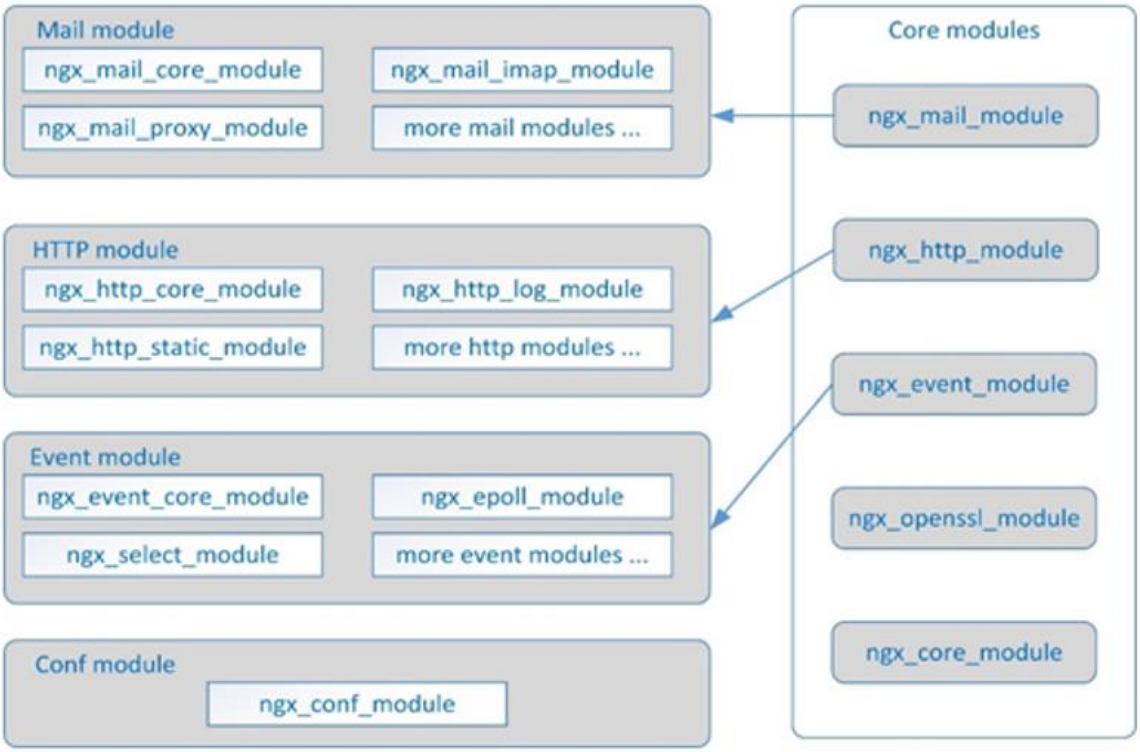
8. Feedback 11

Revision Record

Date	Rev.	Description
05/27/21	1.1	Added feedback section
04/22/2021	1.0	Initial public release.

1. Nginx* Introduction

Developed by Igor Sysoev, a Russian software engineer, Nginx* is a high-performance HTTP and reverse proxy web server based on a BSD-like license. It also provides IMAP/POP3/SMTP services in the meantime. Since the release of the first version in 2004, its market penetration rate has increased year by year, and it has been widely applied in many front-line Internet companies and IT enterprises. Nginx's architecture design is very flexible, with a very small and simple kernel containing core modules, basic modules, and tripartite modules. It collaborates with modules through file static mapping and configurable instructions, highlighting significant advantages of high performance, high concurrency, and low memory in various application scenarios such as HTTP proxy, static and dynamic separation, load balancing, virtual host, reverse proxy, cache acceleration, authorized access, and others.

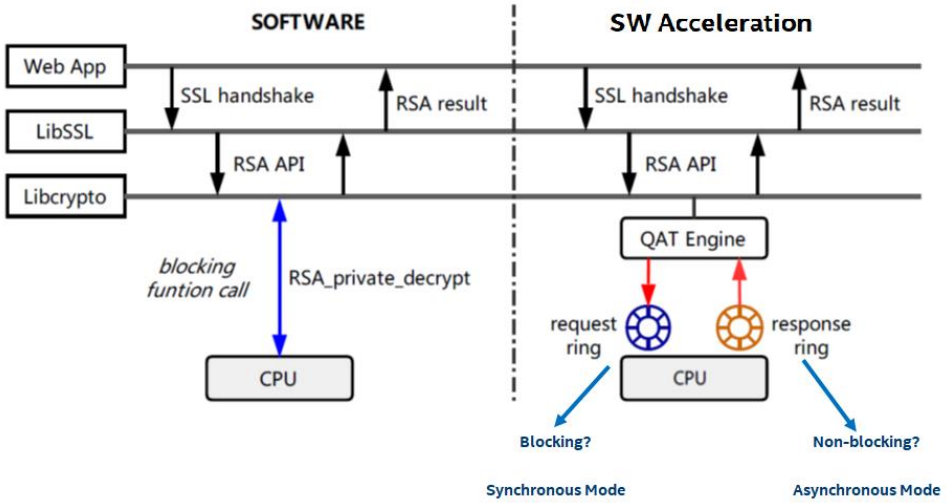


For more information about Nginx, please visit: **http://nginx.org**.

Nginx uses SSL/TLS to enhance web access security. Intel has introduced the Crypto-NI software solution which is based on 3rd generation Intel® Xeon® Scalable Processors (Codename Ice Lake/Whitley). It can effectively improve the security of web access

.

Crypto-NI (NI stands for New Instruction) is a new instruction set in the field of encryption and decryption for 3rd generation Intel® Xeon® Scalable Processors and it adds new instructions such as Vectorized AES and Integer Fused Multiply Add on the basis of the Intel® Advanced Encryption Standard New Instructions (Intel® AES-NI) that the Intel® Xeon® Scalable Processors already has. The main software used in this solution are IPP Cryptography Library, Intel Multi-Buffer Crypto for IPsec Library (**intel-ipsec-mb**) and Intel® QuickAssist Technology (Intel® QAT), which provide batch submission of multiple SSL requests and parallel asynchronous processing mechanism based on the new instruction set, greatly improving the performance.



Tested hardware and software environment used for this tuning guide:

Server Configuration	Hardware	CPU	Intel® Xeon® PLATINUM 8360Y CPU @ 2.20GHz
		Memory	16*32 GB DDR4, 3200 MT/s
		Hard Drive	Intel SSD S4610, 960G
		NIC	Intel® Ethernet Controller XXV700 25GbE SFP28
	Software	Operating System	CentOS* 7.8
		Kernel	3.10.0-1127.el7.x86_64
		Nginx	asynch_mode_nginx v0.4.4
		OpenSSL	v1.1.1j
		IPP Crypto Library	ippcp_2020u3
		Multi-Buffer Crypto for IPsec Library	v0.55
		Intel QAT Engine	v0.6.4

Notes:

- The configuration described in this article are based on the instruction set of 3rd generation Intel® Xeon® Scalable Processors. The previous generations of processors do not support all the same options. Memory, hard drives, and network interface cards can be determined according to customer usage requirements.
- For the software listed in the configuration you can refer to **https://github.com/intel/QAT_Engine.** The oldest versions of each software supported are described on the page the link is directed to. The software versions listed in this article meet the above requirements and have been verified and tested.
- **Asynch_mode_nginx** is an optimized version of Nginx, used by Intel to support Nginx hardware and software acceleration. It can be compiled directly after download. For related descriptions, please refer to the following link: **https://github.com/intel/asynch_mode_nginx.**
- **QAT_Engine for OpenSSL** link: **https://github.com/intel/QAT_Engine.**

2. BIOS Settings

The BIOS configuration items that can be optimized and their recommended values are as follows:

Configuration item	Recommended value
Hyper-Threading	Enable
CPU C6 report	Auto
SpeedStep (Pstates)	Enable
Turbo Mode	Enable
PCIe Link Speed	Gen4
Energy Efficient Turbo	Disable
Boot Performance Mode	Max Performance

3. Linux* Optimization

3.1. System Optimization

1. Set the corresponding *parameters for system startup*

```
intel_iommu=off
processor.max_cstates=1 idle=poll pcie_aspm=off
```

2. Stop **cpupower** service

```
systemctl stop cpupower.service
```

3. Disable the firewall

```
systemctl disable firewalld.service
```

4. User process settings

```
ulimit -c unlimited #generate core dump
ulimit -n 1000000 #set the maximum number of file open handles
```

3.2. Network Optimization

1. Prepare the following tune_affinity.sh script to set the CPU affinity for the NIC queue interrupts, so that these interrupts can be evenly distributed to the local CPU core:

```
set_affinity()
{
    if [ $VEC -ge 32 ]
    then
        MASK_FILL=""
        MASK_ZERO="00000000"
        let "IDX = $VEC / 32"
        for ((i=1; i<=$IDX;i++))
        do
            MASK_FILL="${MASK_FILL},${MASK_ZERO}"
```

```
        done

        let "VEC -= 32 * $IDX"
        MASK_TMP=$((1<<$VEC))
        MASK=`printf "%X%s" $MASK_TMP $MASK_FILL`
    else
        MASK_TMP=$((1<<$VEC))
        MASK=`printf "%X" $MASK_TMP`
    fi

    printf "%s mask=%s for /proc/irq/%d/smp_affinity\n" $DEV $MASK $IRQ
    printf "%s" $MASK > /proc/irq/$IRQ/smp_affinity
}

if [ "$1" = "" ] ; then
    echo "Description:"
    echo "    This script attempts to bind each queue of a multi-queue NIC"
    echo "    to the same numbered core, ie tx0|rx0 --> cpu0, tx1|rx1 --> cpu1"
    echo "usage:"
    echo "    $0 eth0 [eth1 eth2 eth3]"
fi

# check for irqbalance running
#
IRQBALANCE_ON=`ps ax | grep -v grep | grep -q irqbalance; echo $?`
if [ "$IRQBALANCE_ON" == "0" ] ; then
    echo " WARNING: irqbalance is running and will"
    echo "          likely override this script\'s affinitization."
    echo "          Please stop the irqbalance service and/or execute"
    echo "          \'killall irqbalance\'"
fi

# Set up the desired devices.
#
for DEV in $*
do
  for DIR in rx tx TxRx
  do
     MAX=`grep $DEV-$DIR /proc/interrupts | wc -l`
     if [ "$MAX" == "0" ] ; then
       MAX=`egrep -i "$DEV:.*$DIR" /proc/interrupts | wc -l`
     fi
     if [ "$MAX" == "0" ] ; then
       echo no $DIR vectors found on $DEV
       continue
     fi
     for VEC in `seq 0 1 $MAX`
     do
        IRQ=`cat /proc/interrupts | grep -i $DEV-$DIR-$VEC"$" | cut -d:  -f1 | sed "s/
//g"`
        if [ -n "$IRQ" ]; then
          set_affinity
        else
           IRQ=`cat /proc/interrupts | egrep -i $DEV:v$VEC-$DIR"$" | cut -d:  -f1 | sed "s/
//g"`
           if [ -n "$IRQ" ]; then
             set_affinity
           fi
        fi
     done
```

```
  done
done
```

2. Execute the **shell** script:

```
$ sh tune_affinity.sh <network interface card name>
```

4. Crypto-NI installation

4.1. IPP Cryptography Library and intel-ipsec-mb Installation

1. Download the **IPP Cryptography Library** source code

```
$ git clone --recursive https://github.com/intel/ipp-crypto.git
```

2. Build and install **IPP Cryptography Library**

```
$ cd ipp-crypto
$ git checkout ipp-crypto_2020_update3
$ cd sources/ippcp/crypto_mb
$ cmake . -Bbuild -DCMAKE_INSTALL_PREFIX=/usr
$ cd build
$ make -j
$ make install
```

3. Download the **intel-ipsec-mb** source code

```
$ git clone https://github.com/intel/intel-ipsec-mb.git
```

4. Build and install **intel-ipsec-mb**

```
$ cd intel-ipsec-mb
$ git checkout v0.55
$ make -j SAFE_DATA=y SAFE_PARAM=y SAFE_LOOKUP=y
$ make install NOLDCONFIG=y
```

4.2. OpenSSL* and QAT_Engine Installation

1. Download **OpenSSL** source code

```
$ cd /usr/local/src
$ wget https://www.openssl.org/source/openssl-1.1.1j.tar.gz
```

2. Build and install **OpenSSL**

```
$ tar xvzf openssl-1_1_1j.tar.gz
$ cd ./openssl-1_1_1j
$./config --prefix=/usr/local/ssl -Wl,-rpath, /usr/local/ssl/lib
$ ./make
$ ./make install
```

3. Download QAT **Engine** source code

```
$ export OPENSSL_ENGINES=/usr/local/ssl/lib/engines-1.1
$ cd /usr/local/src
```

```
$ git clone https://github.com/intel/QAT_Engine.git
```

4. Build and install **QAT Engine**

```
$cd /QAT_Engine
$./autogen.sh
$./configure \
            --with-openssl_install_dir=/usr/local/ssl \
            --enable-ipsec_offload \
            --enable-multibuff_offload \
            --with-multibuff_install_dir=/root/ipp-crypto/sources/ippcp/crypto_mb
$ make
$ make install
```

The above build and installation instructions may change in the new version. Please refer to the following URL to execute the latest build and installation instructions:
https://github.com/intel/QAT_Engine.git

```
$ ls /usr/local/ssl//lib/engines-1.1/*
```

List the following files: capi.so padlock.so qatengine.la qatengine.so

*Run the following command to check if **qat engine** is loaded properly.*

```
$./openssl engine -t -c -vvvv qatengine
```

5. Using the **openssl speed** command, user can view and verify information before or after using the **QAT Engine**.

```
$/usr/local/ssl/bin/openssl speed rsa2048
$/usr/local/ssl/bin openssl speed -engine qatengine -async_jobs 8 rsa2048
```

5. Nginx installation and Optimization Settings

5.1. Nginx Installation

1. Complete the installation and settings of **Crypto-NI** related software/hardware.

2. Download **Nginx** source code

```
$ git clone https://github.com/intel/asynch_mode_nginx.git
```

3. Build and install **Nginx**

```
$ cd asynch_mode_nginx
$ export NGINX_INSTALL_DIR=<Nginx installation directory>
$ export OPENSSL_LIB=<SSL installation directory>
$./configure \
        --prefix=$NGINX_INSTALL_DIR \
        --with-http_ssl_module \
        --add-dynamic-module=modules/nginx_qat_module \
        --with-cc-opt="-DNGX_SECURE_MEM -I$OPENSSL_LIB/include -Wno-error=deprecated-
declarations" \
        --with-ld-opt="-Wl,-rpath=$OPENSSL_LIB/lib -L$OPENSSL_LIB/lib"
$ make
$ make install
```

4. Start Nginx services

   After completing the Nginx settings, execute the **$ ./sbin/nginx** command in the Nginx installation directory to start the **Nginx** service.

5.2. Optimized Settings for Nginx

1. Generate RSA self-signed certificate and key file.

```
$ openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 -keyout
server.crt -subj "/C=CN/ST=Beijing/L=Beijing/O=Example Inc./OU=Web
Security/CN=example1.com"
```

   Create a new **cert** directory under the **Nginx** installation directory and copy **server.crt** and **server.key** to that directory.

2. Key **nginx** configuration. Set the following parameters in the **nginx.conf** file:

   - Set the number of worker processes and event mode

```
worker_processes <number of worker processes>; # it is recommended to have the same
number of worker processes as the number of CPU cores on server
events {
    Use epoll; #epoll is an enhanced version of select/poll that improves the handling of
large number of file descriptors
    …
}
```

   - Load and set up the **SSL Engine** module

```
load_module modules/ngx_ssl_engine_qat_module.so;
ssl_engine {
    use_engine qatengine;
          default_algorithms ALL;
          qat_engine {
                qat_offload_mode async;
                qat_notify_mode poll;
                qat_poll_mode heuristic;
                qat_shutting_down_release off;
          }
}
```

   - Set **HTTPS Server**

```
server {
          Listen 443 ssl reuseport so_keepalive=on... ; #if you want to verify the SSL
handshake performance, then set it to off.
          sendfile on;
          Server_name example1.com; #consistent with certificate CN
          ssl_asynch on;
          ssl_session_tickets on; #set it to off, if you want to verify the SSL
handshake performance
          ssl_certificate <Nginx installation directory>/cert/server.crt;
          ssl_certificate_key <Nginx installation directory>/cert/server.key;
          ssl_ciphers         AES256-SHA:AES128-SHA;
    #You can set other cipher suites according to customer needs, to verify the
performance of different authentication, encryption and decryption protocols
          ssl_prefer_server_ciphers on;
```

```
            …
}
```

3. A complete sample configuration of **nginx.conf**, which is suitable for **RSA2K** handshake performance test:

```
user root;
worker_processes 8;
load_module modules/ngx_ssl_engine_qat_module.so;
events {
    use epoll;
    worker_connections 8192;
    multi_accept on;
    accept_mutex on;
}

ssl_engine {
    use_engine qatengine;
    default_algorithms ALL;
    qat_engine {
        qat_offload_mode async;
        qat_notify_mode poll;
        qat_poll_mode heuristic;
        qat_shutting_down_release off;
    }
}
http {
    server {
        listen example1:443 ssl reuseport backlog=131072 so_keepalive=off rcvbuf=65536
sndbuf=65536;
        keepalive_timeout 0s;
        ssl_verify_client off;
        ssl_session_tickets off;
        access_log  off;
        ssl_asynch on;
        ssl_session_timeout 300s;
        ssl_protocols TLSv1.2;
        ssl_ciphers  AES128-SHA;
        ssl_prefer_server_ciphers on;
        ssl_certificate      server2048.crt;
        ssl_certificate_key server2048.key;
        location / {
            root html;
            index  index.html index.htm;
        }
    }
}
```

6. Introduction to Related Tools

6.1. Web Server Stress Testing Tools

In this section, we will introduce two web server stress testing tools: **wrk** and **ab**.

- **Wrk** is a benchmark testing tool for **HTTP/HTTPS** protocols. On a single machine powered by multi-core CPUs, it can generate a huge workload on the target machine in multi-threading and event I/O modes, using the system's high-performance I/O mechanisms, such as **epoll** and **kqueue**.

Usage Example: *(where users would replace the URL with their own Nginx URL)*

```
$ wrk -t 10 -c 1000 -d 30S https://example1.com:443/index.html
```

Description: the command uses 10 threads and 1,000 concurrent connections to fetch files and stress test the server for 30 seconds. For more information, please visit **https://github.com/wg/wrk**.

- The **ab** tool is a stress testing tool that comes with Apache*. It can be used not only for website access stress testing of Apache servers, but also for stress testing of other types of servers. For example, **Nginx**, ***Apache* Tomcat***, and **IIS**. If the user has installed Apache, the **ab** tool has already been installed with Apache. If the user has not installed Apache, then it can be installed easily with the following command (**CentOS***):

```
$ yum -y install httpd-tools
```

Usage Example: *(where users would replace the URL with their own Nginx URL)*

```
$ ab -n 100000 -c 100 -Z AES128-SHA -f TLS1.2 https://example1.com:443/index.html
```

Description: The command uses 100 concurrent connections to send 100,000 requests for "get index" files to perform stress testing on the server. For more information, please visit **https://httpd.apache.org/docs/2.4/programs/ab.html.**

7. Conclusion

We understand every application is unique. We shared many of our experiences with NGINX HTTPs with Crypto-NI hoping that some of our learnings could be applied to your specific application. NGINX HPPTS with Crypto-NI has been well tested on Intel platforms. With 3rd Generation Intel® Xeon® Scalable processor, Intel takes it even further by optimizing the platform as a whole -- CPU, memory, storage, and networking working together for the best user experience.

8. Feedback

We value your feedback. If you have comments (positive or negative) on this guide or are seeking something that is not part of this guide, please reach out to us here:
https://community.intel.com/t5/Software-Tuning-Performance/bd-p/software-tuning-perf-optimization