TCP Tuning with Quattor

Jérôme Pansanel <jerome.pansanel@iphc.cnrs.fr>

22 Mar 2010

This document presents the usage of Quattor for modifying kernel parameters in order to improve TCP performances. It is divided in four sections:

- Quattor usage for modifying kernel parameters
- Relevant kernel parameters for TCP tuning
- Benchmark tools
- Further reading

Quattor usage for modifying kernel parameters

This section describes how to modify kernel parameters with Quattor.

Sysctl

The **sysctl** software is a command line tool for examining and dynamically modifying the kernel parameters. The available parameters are those listed under `/proc/sys`. The **sysctl** command can be used in two ways for parameter modifications:

- By directly passing the parameter to the **sysctl** command. This method is useful for testing a parameter, as the modification will not persist after the next reboot.

  ```
  [root@localhost ~]$ sysctl -w net.core.rmem_max=8388608
  ```

- By adding the parameter to the `/etc/sysctl.conf` file and loading it with the **sysctl** command. This way is interesting when you want to preserve the modification over a reboot.

  ```
  [root@localhost ~]$ echo 'net.core.rmem_max=8388608' >> /etc/sysctl.conf
  [root@localhost ~]$ sysctl -p
  ```

It can also be used in the following way to get the current value of a parameter. This is useful for verifying that your modification has been successfully applied.

```
[root@localhost ~]$ sysctl net.core.rmem_max
net.core.rmem_max = 8388608
```

The *sysctl* Quattor component

Quattor provides a dedicated component to handle the `/etc/sysctl.conf` configuration file. To modify the content of this file, include the *sysctl* component in your template and set the `/software/components/sysctl/variables` path to a nlist:

```
include { 'components/sysctl/config' };

'/software/components/sysctl/variables' = nlist(
  'net.core.rmem_max','8388608',
);
```

Note that when you remove a parameter from the template, it is not deleted from the `/etc/systctl.conf` file on the host. You have to do it manually.

The current TCP tuning in the templates

Several QWG templates are currently modifying the kernel parameters to enhance the TCP performance:

- `grid/glite-3.1/glite/se_dpm/disk/service.tpl`
- `grid/glite-3.2/glite/se_dpm/disk/service.tpl`
- `grid/glite-3.2/glite/vobox/config.tpl`

Relevant kernel parameters for TCP tuning

The performance of TCP/IP can be significantly increased by carefully tuning several parameters:

- TCP window
- SACK and DSACK
- Timestamps
- Backlog

TCP window

A TCP window is the amount of outstanding (unacknowledged by the recipient) data a sender can send on a particular connection before it gets an acknowledgment back from the receiver that it got some of it. In theory, the TCP window size should be set to be the product of the available network's bandwidth and the round-trip time of data going over the network:

buffer size = bandwidth * RTT

The round-trip time is the delay between the emission of a signal at one side of a line and the acknowledgement of its reception. In our case the signal is generally a data packet, and the time is also known as ping time. Look at the ping statistics to get the average RTT to a remote host (You should try to make an oversea ping).

```
--- host.oversee.gov ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10011ms
rtt min/avg/max/mdev = 109.101/109.409/110.155/0.476 ms, pipe 2
```

For example, if a network has a bandwidth of 1 Gbits/s and the round-trip time (RTT) was 110ms, the theoretical maximum buffer size should be $10^9 * 110 * 10^{-3}$ bits (13750000 bytes) to use 100% of your bandwidth.

There are two TCP settings that modify the buffer size and affect the TCP window:

- The maximum TCP send and receive buffer size; they can be set with the following lines:

  ```
  'net.core.rmem_max','8388608',
  'net.core.wmem_max','8388608',
  ```

- The default send and receive buffer size. These parameters are composed of a vector of 3 integers (min, default, max). The first integer is the minimal size of buffer used by TCP socket. It is guaranteed to each TCP socket, even under moderate memory pressure. The second integer is the initial size of buffer used by TCP sockets, and the third integer is the maximal size of buffer allowed for automatically selected buffers for TCP socket. This last parameter does not override the `net.core.(r|w)mem_max` parameter. The size of the buffer is dynamically tuned by the kernel and is limited by the *min* and *max* integers. These parameters can be modified with the following lines:

  ```
  'net.ipv4.tcp_rmem','4096 87380 8388608',
  ```

```
'net.ipv4.tcp_wmem','4096 65536 8388608',
```

The above settings will set a maximum of 8 MBytes buffers. For oversea data transfer, it is recommended to use at least 4MBytes maximum buffer size and a maximum of 16 MBytes should be enough. It is not recommended to set a very high values for those parameters, and especially do not set the same value for all the fields in `net.ipv4.tcp_(r|w)mem`. It is also a good practice to have the same value for `net.core.r(w)mem_max` with the last value in the `net.ipv4.tcp_r(w)mem`.

SACK and DSACK

Selective acknowledgments (SACK) and duplicate SACK (SACK) are a way to optimize TCP traffic. When this option is turned on, the host will validate SACK in the TCP option field in the TCP header when it sends out a SYN packet. This option selectively acknowledges each segment in a TCP window. This is especially good on very lossy connections. However, SACKs and DSACKs can adversely affect performance on gigabit networks with no traffic competition. While enabled by default, SACK and DSACK options should be disabled on high-speed network for optimal TCP/IP performance.

The `net.ipv4.tcp_sack` and `net.ipv4.tcp_dsack` options takes a boolean value. This is per default set to 1, or turned on. Disable these options with the following lines:

```
'net.ipv4.tcp_sack','0',
'net.ipv4.tcp_dsack','0',
```

Timestamps

Each time an Ethernet frame is forwarded to the network stack of the Linux kernel, it receives a timestamps. Timestamps are defined in RFC 1323. This behavior is useful and necessary for edge systems such as firewalls, but backend systems might benefit from disabling the TCP timestamps by reducing some overhead. In addition, on some kernel versions timestamps disables automatic window scaling.

The `net.ipv4.tcp_timestamps` variable tells the kernel to use timestamps as defined in RFC 1323. It takes a boolean value and is per default set to 1 (enabled). Add the following line to disable timestamps:

```
'net.ipv4.tcp_timestamps','0',
```

Backlog

The backlog variable tells the kernel the maximum number of remembered connection requests, which currently did not receive an acknowledgment from connecting client. It is recommended to increase this number if the server suffers of overload.

The `net.core.netdev_max_backlog` parameter should be set at least to 2500 for a 1 GBits Ethernet connection and 30000 for 10 Gbits Ethernet connection.

```
# For 1 Gbits network
'net.core.netdev_max_backlog','2500',
# For 10 Gbits network
'net.core.netdev_max_backlog','30000',
```

Benchmark tools

In this section, we discuss major benchmark tools. A benchmark is nothing more than a model for a specific workload that might or might not be close to the actual workload that will run on a system. Generally, the following rules should be observed when performing a benchmark on any system:

- Simulate the expected workload: All benchmarks have different configuration options that should be used to tailor the benchmark towards the workload that the system should be running in the future.
- Isolate the benchmark systems: If a system is to be tested with a benchmark it is paramount to isolate it from any other load as much as possible.
- Average results: Even if you try to isolate the benchmark system there might be unknown factors that could impact systems performance just at time of your benchmark. It is good practice to run any benchmark at least three times and look at the dispersion. If the dispersion is to high, it is advisable to investigate uncontroled parameters in the benchmark.

Iperf

Iperf is a performance benchmark tool that focuses on TCP/IP networking performances. It allows the user to set various parameters that can be used for testing, or alternately for optimizing or tuning a network. Iperf has client and server functionality, and can measure the throughput between the two ends.

Further documentation can be obtained on the Iperf website [http://iperf.sourceforge.net/].

For testing the TCP tuning, the following command can be used:

- On the server

  ```
  [root@localhost ~]$ iperf -m -sD -p 5001
  ```

  This command start the iperf server as a daemon. The service is listening on port 5001.

- On the client

  ```
  [root@localhost ~]$ iperf -c iperfserver.my_domain -m -i 5 -p 5001 \
  > -w 2048k -l 40M -t 40 -r -L 24001
  ```

  In this example, the command will launch an iperf client that connect to the iperfserver.my_domain server with the following options:

  - `-m`: print TCP maximum segment size;
  - `-i 5`: pause 5 seconds between periodic bandwidth reports;
  - `-p 5001`: connect to port 5001;
  - `-w 2048k`: use a size of 2048 KB for the TCP window;
  - `-l 40M`: set the length of the read/write buffer to 40 MB;
  - `-t 40`: transmit for 40s;
  - `-r`: Do a bidirectional test individually;
  - `-L 24001`: receive bidirectional tests back on port 24001.

Netperf

Netperf is a benchmark software that can be used to measure the performance of many different types of network usage. It provides tests for both unidirectional throughput, and end-to-end latency. It is based on a client-server model. The netserver application run on a target system and netperf runs on the client, netperf controls the netserver and passes configuration data to netserver generates network traffic, and gets the result from netserver through a control connection that is separated from the actual benchmark traffic connection. During the benchmarking, no communication occurs on the control connection so it does not have any effect on the result. The netperf benchmark tool also has a reporting capability including a CPU utilization report.

For further informations, visit the official website [http://www.netperf.org/netperf/]

Further reading

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topic covered by this documentation.

Books

- TCP/IP Architecture, Design and Implementation in Linux, by Sameer Seth and M. Ajaykumar Venkatesulu (ISBN 978-0-470-14773-3)
- Linux Performance and Tuning Guidelines, by Eduardo Ciliendo and Takechika Kunimasa (http://www.redbooks.ibm.com/redpapers/abstracts/redp4285.html).

Web Articles

- http://fasterdata.es.net/
- http://monalisa.cern.ch/FDT/documentation_syssettings.html
- http://indico.cern.ch/contributionDisplay.py?sessionId=31&contribId=55&confId=61917
- http://www.psc.edu/networking/projects/tcptune/
- http://onlamp.com/pub/a/onlamp/2005/11/17/tcp_tuning.html
- http://en.wikipedia.org/wiki/TCP_tuning

RFCs

- RFC 1323 [http://www.ietf.org/rfc/rfc1323.txt?number=1323]: TCP Extensions for High Performance
- RFC 2883 [http://www.ietf.org/rfc/rfc2883.txt?number=2883]: An Extension to the Selective Acknowledgement (SACK) Option for TCP