LinuxOSSpecific

Linux-Specific Network Performance Tuning Hints

Linux has its own implementation of the TCP/IP Stack. With recent kernel versions, the TCP/IP implementation contains many useful performance features. Parameters can be controlled via the `/proc` interface, or using the `sysctl` mechanism. Note that although some of these parameters have `ipv4` in their names, they apply equally to TCP over IPv6.

A typical configuration for high TCP throughput over paths with high bandwidth*delay product would include the following in `/etc/sysctl.conf` :

A description of each parameter listed below can be found in section Linux IP Parameters.

Basic tuning

TCP Socket Buffer Tuning

See the EndSystemTcpBufferSizing topic for general information about sizing TCP buffers.

Since 2.6.17 kernel, buffers have sensible automatically calculated values for most uses. Unless very high RTT, loss or performance requirement (200+ Mbit/s) is present, buffer settings may not need to be tuned at all.

Nonetheless, the following values may be used:

```
net/core/rmem_max=16777216
net/core/wmem_max=16777216
net/ipv4/tcp_rmem="8192 87380 16777216"
net/ipv4/tcp_wmem="8192 65536 16777216"
```

With kernel < 2.4.27 or < 2.6.7, receive-side autotuning may not be implemented, and the default (middle value) should be increased (at the cost of higher, by-default memory consumption):

```
net/ipv4/tcp_rmem="8192 16777216 16777216"
```

NOTE: If you have a server with hundreds of connections, you might not want to use a large default value for TCP buffers, as memory may quickly run out :)

There is a subtle but important implementation detail in the socket buffer management of Linux. When setting either the send- or receive buffer sizes via the SO_SNDBUF and SO_RCVBUF socket options via `setsockopt(2)` , the value passed in the system call is doubled by the kernel to accomodate buffer management overhead. Reading the values back with `getsockopt(2)` return this modified value, but the effective buffer available to TCP payload is still the original value.

The values `net/core/rmem` and `net/core/wmem` apply to the argument to `setsockopt(2)` .

In contrast, the maximum values of `net/ipv4/tcp_rmem=/=net/ipv4/tcp_wmem` apply to the total buffer sizes **including** the factor of 2 for the buffer management overhead. As a consequence, those values must be chosen twice as large as required by a particular BandwidthDelayProduct. Also note taht the values `net/core/rmem` and `net/core/wmem` do not apply to the TCP autotuning mechanism.

Interface queue lengths

InterfaceQueueLength describes how to adjust interface transmit and receive queue lengths. This tuning is typically needed with GE or 10GE transfers.

Host/adapter architecture implications

When going for 300 Mbit/s performance, it is worth verifying that host architecture (e.g., PCI bus) is fast enough. PCI Express is usually fast enough to no longer be the bottleneck in 1Gb/s and even 10Gb/s applications.

For the older PCI/PCI-X buses, when going for 2+ Gbit/s performance, the Maximum Memory Read Byte Count (MMRBC) usually needs to be increased using `setpci` .

Many network adapters support features such as checksum offload. In some cases, however, these may even decrease performance. In particular, TCP Segment Offload may need to be disabled, with:

```
ethtool -K eth0 tso off
```

Advanced tuning

Sharing congestion information across connections/hosts

2.4 series kernels have a TCP/IP weakness in that their interface buffers' maximum window size is based on the experience of previous connections - if you have loss at any point (or a bad end host at the same route) you limit your future TCP connections. So, you may have to flush the route cache to improve performance.

```
net.ipv4.route.flush=1
```

2.6 kernels also remember some performance characteristics across connections.
In benchmarks and other tests, this might not be desirable.

```
# don't cache ssthresh from previous connection
net.ipv4.tcp_no_metrics_save=1
```

Other TCP performance variables

If there is packet reordering in the network, reordering could end up being interpreted as a packet loss too easily. Increasing `tcp_reordering` parameter might help in that case:

```
net/ipv4/tcp_reordering=20   # (default=3)
```

Several variables already have good default values, but it may make sense to check that these defaults haven't been changed:

```
net/ipv4/tcp_timestamps=1
net/ipv4/tcp_window_scaling=1
net/ipv4/tcp_sack=1
net/ipv4/tcp_moderate_rcvbuf=1
```

TCP Congestion Control algorithms

Linux 2.6.13 introduced *pluggable congestion modules*, which allows you to select one of the high-speed TCP congestion control variants,e.g.CUBIC

```
net/ipv4/tcp_congestion_control = cubic
```

Alternative values include `highspeed` (HS-TCP), `scalable` (Scalable TCP), `htcp` (Hamilton TCP), `bic` (BIC), `reno` ("Reno" TCP), and `westwood` (TCP Westwood).

Note that on Linux 2.6.19 and later, CUBIC is already used as the default algorithm.

Web100 kernel tuning

If you are using a web100 kernel, the following parameters seem to improve networking performance even further:

```
# web100 tuning
# turn off using txqueuelen as part of congestion window computation
net/ipv4/WAD_IFQ = 1
```

QoS tools

Modern Linux kernels have flexible traffic shaping built in.

See the Linux traffic shaping example for an illustration of how these mechanisms can be used to solve a real performance problem.

References

- *TCP Tuning Guide - Linux*, Lawrence Berkeley National Laboratory Web site.
  Covers the basic TCP/IP parameters (socket buffers, interface `txqueuelen` , BIC etc.) on Linux. It is also very good for describing the evolution of TCP through the Linux kernels (2.4 -> 2.6.7 -> 2.6.13)
- *Ipsysctl Tutorial*, Oscar Andreasson,

Very comprehensive guide to configuring network-related kernel settings in Linux, including detailed descriptions of many TCP parameters.

- *Boost socket performance on Linux*, M. Tim Jones, January 2006, on IBM developerWorks.
  This mostly has hints for programmers using the Socket interface on Linux, but also contains explanations of some system tunables for administrators.
- *How to achieve Gigabit speeds with Linux*, M. Rio et al., Web page on CERN's DataTAG project site. Has many recommandations about different configurable parameters.
- *A Map of the Networking Code in Linux Kernel 2.4.20*, M. Rio, M. Goutelle, T. Kelly, R. Hughes-Jones, J.P. Martin-Flatin and Y.T. Li, March 2004 - very comprehensive and readable account of the way Linux handles network traffic
- *Cluster Interconnects: The Whole Shebang*, J. Leighton, April 2006, ClusterMonkey Web site. Much general information about cluster interconnect technologies, including Gigabit Ethernet, along with Linux-specific configuration examples and performance benchmarking scripts.
- *HOWTO Packet Shaping*
- *Traffic Shaping in Shorewall*, Thomas M. Eastep, Arne Bernin
  Information on using traffic shaping on a Shoreline Firewall.
- *TCP infrastructure split out*, S. Hemminger, lwn.net, March 2005
- *Pluggable congestion avoidance modules*, J. Corbet, lwn.net, March 2005

See also the reference to Congestion Control in Linux in the FlowControl topic.

– Main.SimonLeinen - 27 Oct 2004 - 23 Jul 2009

– Main.ChrisWelti - 27 Jan 2005

-- Main.PekkaSavola - 17 Nov 2006

-- AlexGall - 28 Nov 2016