D1.3 End-to-End Performance Monitoring

Freek Dijkstra, Ronald van der Pol, and Peter Tavenier

SARA Computing & Networking Services, Science Park 121, 1098 XG Amsterdam, The Netherlands
March 2010
freek.dijkstra@sara.nl,ronald.vanderpol@sara.nl,peter.tavenier@sara.nl
http://nrg.sara.nl/

1 Introduction

This document describes the performance tuning of a server. SARA has successfully used this approach in 2010 to get over 40 Gb/s from disk to network from a single server. This document is intended as a how to for others who like to repeat this work. After reading this document you should have a firm grasp of how to optimize your hardware for a given data flow in your machine, and will be able to find potential bottlenecks by applying the right tools.

This document is by no means a complete overview of performance tuning. Readers are encouraged to check out the related work in section 13.

This document, including fixes and recent additions, is also available online at `https://noc.sara.nl/wiki/Server_Performance_Tuning`

2 Hardware Design

A server is typically tuned to a specific usage. For example memory to network, CPU intensive, disk to CPU to disk, network to GPU to display, etc. It helps to understand the data flows inside the server.

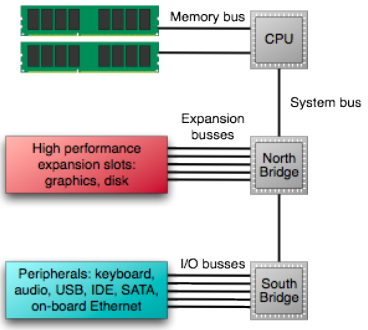

Fig. 1. Motherboard block diagram with memory controller in CPU.

Figures 1 and 2 give a typical design of a modern motherboard, both for a single CPU and a dual CPU/dual controller system. Previously, the memory was connected to the North Bridge, but since the Intel Nehalem processors and AMD64 processors, the memory controller is integrated in the CPU.

The North Bridge (the data interconnect chipset or system controller) is designed for high performance, while the South Bridge (the I/O controller hub) is designed for versatility. For that reason, all high components requiring high bandwidth should be connected to the North Bridge.

The technologies in table 1 are used for the various busses in a server in 2010. This list will change over time. While the interconnects are still referred to as a (computer)bus, it is no longer a shared medium and the data transport occurs in parallel.

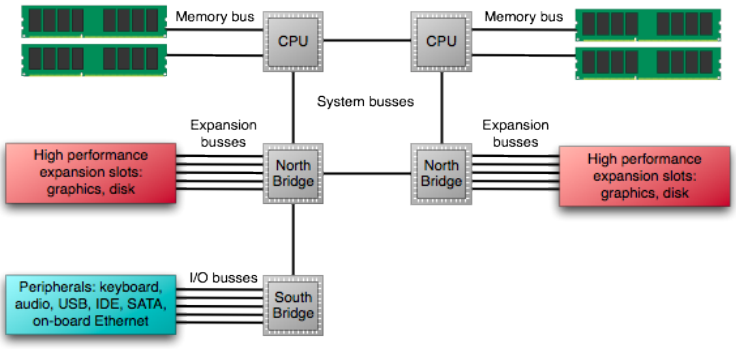

Fig. 2. Motherboard block diagram with dual CPU and dual North Bridge.

Memory bus	DDR3
System bus or Frontside bus	Quick Path Interconnect (QPI) or HyperTransport3
Expansion bus	PCI express 2.0

Table 1. Common computer busses.

3 Memory

DDR3 SDRAM is commonly used in servers (with DDR4 SDRAM expected to hit the market in 2011).

Roughly three performance metrics matter for memory: capacity (the total amount of memory), bandwidth (the number of memory operations per second) and latency (the wait time before an operation is finished).

The DIMMS in a bus operate at a speed which depends on the clock speed of the memory bus (see figure 2). (The acronym DDR stands for Dual Data Rate, since the bus is read both at the rising and falling clock signal).

Chipset	Module	I/O bus clock (data rate)	Lane size	Peak data transfer
DDR3-800	PC3-6400	400 MHz (800 MT/s)	64 bit	51.2 Gbit/s
DDR3-1066	PC3-8500	533 MHz (1066 MT/s)	64 bit	68.3 Gbit/s
DDR3-1333	PC3-10600	667 MHz (1333 MT/s)	64 bit	85.3 Gbit/s
DDR3-1600	PC3-12800	800 MHz (1600 MT/s)	64 bit	102.4 Gbit/s

Table 2. Maximum data transfer rates for DDR3 SDRAM.

A memory bus can contain multiple DIMMs. These DIMM share the bandwidth on the bus, and to make it worse, some memory controllers (including the one in the Intel Nehalem processors) reduce the clock rate if there are multiple DIMMs on the same channel (see table 3).

Number of DIMMs per channel	Clock speed
1	667 MHz (1333 MT/s)
2	533 MHz (1066 MT/s)
3	400 MHz (800 MT/s)

Table 3. DIMM speeds.

So more memory allows larger calculations, but makes it slower. Most memory controllers have multiple channels, so the performance may be increased by carefully distributing the DIMMs over the channels.

4 System Bus

QPI, HyperTransport and PCI express are the three high-bandwidth interconnect technologies in current use (see figure 4). Of these, QPI and HyperTranport are used for the front side bus or system bus.

System Bus	version	I/O bus clock (data rate)	Lane size	Encoding	Peak transfer
Intel QPI		2.4 GHz (4800 MT/s)	20 bit	72/80	86.4 Gbit/s
Intel QPI		2.93 GHz (5860 MT/s)	20 bit	72/80	105.5 Gbit/s
Intel QPI		3.2 GHz (6400 MT/s)	20 bit	72/80	115.2 Gbit/s
HyperTransport	1.0	800 MHz (1600 MT/s)	2-32 bit	none	51.2 Gbit/s
HyperTransport	2.0	1.4 GHz (2800 MT/s)	2-32 bit	none	89.6 Gbit/s
HyperTransport	3.0	2.6 GHz (5200 MT/s)	2-32 bit	none	166.4 Gbit/s
HyperTransport	3.1	3.2 GHz (6400 MT/s)	2-32 bit	none	204.8 Gbit/s

Table 4. Maximum data transfer rates for System Busses per direction.

The QPI peak data rate excludes the link layer overhead, which is 8 bit CRC per 80 bit *flit* (a flit is a kind of package) (72/80). The peak data rate does not take the routing layer overhead into account. This overhead is roughly equal for both technologies. For QPI, each 72-bit flit contains 64-bits payload. For HyperTransport, each 4-64 byte data packet is preceded by a 4 or 8 byte control packet. So the effective data rate is roughly 64/72-th of the above reported rates.

QPI and HyperTranport are full duplex. Most peak data transfer rate quote the combined bidirectional speed. The above table lists the unidirectional speed.

5 PCI Express

PCI Express is very similar in technology to HyperTransport. The different versions have different clock speeds, and the lane size is flexible as well. Table 5 shows the speeds per amount of lanes for the three versions of PCI Express:

Version	Data rate	Encoding	x1 lane	x4 lane	x8 lane	x16 lane
1.0	2.5 GT/s	8/10	2.0 Gb/s	8.0 Gb/s	16.0 Gb/s	32.0 Gb/s
2.0	5.0 GT/s	8/10	4.0 Gb/s	16.0 Gb/s	32.0 Gb/s	64.0 Gb/s
3.0	8.0 GT/s	128/130	7.9 Gb/s	31.5 Gb/s	63.0 Gb/s	126.0 Gb/s

Table 5. DIMM speeds.

For example, PCI Express version 2.0 has 5 Gigatransfers per second. Four lanes of PCI Express 2.0 gives 20 Gbit/s. Because PCI Express version 2.0 uses 8b/10b encoding (the bus needs to send 10 bits of encoded data for every 8 bits of unencoded data) you loose 20of 20 Gbit/s = 16 Gbit/s.

While the table does take the link layer encoding overhead into account, the data layer also adds some overhead. Each Transaction Layer Packet (TLP) has a size of 128 bytes to 4096 bytes, and has 24 bytes of overhead (8 bytes checksum and sequence number; 16 bytes header information). So the effective peak data rate is roughly between 0.81 (104/128-th) and 0.99 (4072/4096-th) of the above reported rates.

A potential pitfall is that some PCI Express 2.0 network cards only run at PCI Express 1.0 frequency (which is 2.5GHz). For example, the Intel 82598EB 10 Gigabit Ethernet Controller has an interface type of "PCIe v2.0 (2.5GT/s)". Below is an example of an `lspci -vv` output for this network card:

```
04:00.0 Ethernet controller: Intel Corporation 82598EB 10-Gigabit AF Network Connection (rev 01)
       Subsystem: Intel Corporation Device a05f
       Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR- FastB2B- DisINTx+
       Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort- >SERR- <PERR- INTx-
       Latency: 0, Cache Line Size: 64 bytes
       Interrupt: pin A routed to IRQ 33
       Region 0: Memory at df1a0000 (32-bit, non-prefetchable) [size=128K]
       Region 1: Memory at df1c0000 (32-bit, non-prefetchable) [size=256K]
       Region 2: I/O ports at ece0 [size=32]
       Region 3: Memory at df19c000 (32-bit, non-prefetchable) [size=16K]
       Capabilities: [40] Power Management version 3
               Flags: PMEClk- DSI+ D1- D2- AuxCurrent=0mA PME(D0+,D1-,D2-,D3hot+,D3cold-)
               Status: D0 PME-Enable- DSel=0 DScale=1 PME-
       Capabilities: [50] Message Signalled Interrupts: Mask- 64bit+ Queue=0/0 Enable-
               Address: 0000000000000000  Data: 0000
       Capabilities: [60] MSI-X: Enable+ Mask- TabSize=18
               Vector table: BAR=3 offset=00000000
               PBA: BAR=3 offset=00002000
       Capabilities: [a0] Express (v2) Endpoint, MSI 00
               DevCap: MaxPayload 256 bytes, PhantFunc 0, Latency L0s <512ns, L1 <64us
                       ExtTag- AttnBtn- AttnInd- PwrInd- RBE+ FLReset-
               DevCtl: Report errors: Correctable+ Non-Fatal+ Fatal+ Unsupported+
                       RlxdOrd+ ExtTag- PhantFunc- AuxPwr- NoSnoop+
                       MaxPayload 256 bytes, MaxReadReq 512 bytes
               DevSta: CorrErr- UncorrErr- FatalErr- UnsuppReq- AuxPwr- TransPend-
               LnkCap: Port #1, Speed 2.5GT/s, Width x8, ASPM L0s L1, Latency L0 <4us, L1 <64us
                       ClockPM- Suprise- LLActRep- BwNot-
               LnkCtl: ASPM Disabled; RCB 64 bytes Disabled- Retrain- CommClk+
                       ExtSynch- ClockPM- AutWidDis- BWInt- AutBWInt-
               LnkSta: Speed 2.5GT/s, Width x4, TrErr- Train- SlotClk+ DLActive- BWMgmt- ABWMgmt-
       Capabilities: [100] Advanced Error Reporting <?>
       Capabilities: [140] Device Serial Number 05-36-27-ff-ff-21-1b-00
       Kernel driver in use: ixgbe
       Kernel modules: ixgbe_new, ixgbe_old, ixgbe
```

You can see that the LnkCap is giving a Speed of 2.5GT/s, despite the branding as an PCIe version 2.0 card. In this case, the 10GE card is placed in a x4 lanes slot, as seen at the LnkSta output, which means that the card is not able to use its full 10 Gb/s, but is theoretical limited to 8 Gb/s. The LnkCap in the above report gives the capability of the network card, not of the network bus. In order to find the capabilities of the network bus, use lspci -vt to determine the bus where the card is located:

```
$ lspci -vt
-[0000:00]-+-00.0  Intel Corporation 5520 I/O Hub to ESI Port
           +-01.0-[0000:01]--+-00.0  Broadcom Corporation NetXtreme II BCM5709 Gigabit Ethernet
           |                 \-00.1  Broadcom Corporation NetXtreme II BCM5709 Gigabit Ethernet
           +-03.0-[0000:03]--
           +-04.0-[0000:04]----00.0  Intel Corporation 82598EB 10-Gigabit AF Network Connection
```

And than query the details of the slot:

```
$ sudo lspci -vv -s 04
00:04.0 PCI bridge: Intel Corporation 5520/X58 I/O Hub PCI Express Root Port 4 (rev 13)
       ....
                LnkCap: Port #4, Speed 5GT/s, Width x4, ASPM L0s L1, Latency L0 <512ns, L1 <64us
                        ClockPM- Suprise+ LLActRep+ BwNot+
                LnkCtl: ASPM Disabled; RCB 64 bytes Disabled- Retrain- CommClk+
                        ExtSynch- ClockPM- AutWidDis- BWInt- AutBWInt-
                LnkSta: Speed 2.5GT/s, Width x4, TrErr- Train- SlotClk+ DLActive+ BWMgmt- ABWMgmt-
```

This shows a LnkCap of 5GT/s with x4 lanes.

6 Hardware Detection Tools

6.1 General Hardware Configuration

The following tools are designed to display the hardware specs of your server. lshw and hwinfo are fairly complete in their probing. In this example you see a partial output of lshw:

```
      *-pci:2
           description: PCI bridge
           product: 5520/X58 I/O Hub PCI Express Root Port 4
           vendor: Intel Corporation
           physical id: 4
           bus info: pci@0000:00:04.0
           version: 13
           width: 32 bits
           clock: 33MHz
           capabilities: pci msi pciexpress pm bus_master cap_list
           configuration: driver=pcieport-driver
           resources: irq:58 ioport:e000(size=4096) memory:df100000-df1fffff
         *-network
              description: Ethernet interface
              product: 82598EB 10-Gigabit AF Network Connection
              vendor: Intel Corporation
              physical id: 0
              bus info: pci@0000:04:00.0
              logical name: eth5
              version: 01
              serial: 00:1b:21:27:36:05
              width: 32 bits
              clock: 33MHz
              capabilities: pm msi msix pciexpress bus_master cap_list ethernet physical fibre
              configuration: autonegotiation=off broadcast=yes driver=ixgbe driverversion=2.0.75.7-NAPI duplex=full
      firmware=1.7-0 ip=192.168.88.140 latency=0 link=yes multicast=yes port=fibre
              resources: irq:33 memory:df1a0000-df1bffff memory:df1c0000-df1fffff ioport:ece0(size=32)
      memory:df19c000-df19ffff
```

You can use the option `-businfo` you get a detailed bus address:

```
Bus info          Device      Class      Description
=====================================================
pci@0000:04:00.0  eth5        network    82598EB 10-Gigabit AF Network Connection
```

Another useful option is to user the class, you can see in the command above that this card is from class network. So if you use `lshw -class network`, you get all the information of only the classes network. `hwinfo` gives some more detailed information. To just check the output of `hwinfo` could be an overkill of information, to get a quick overview you can better use `hwinfo --short`, which gives a much shorter overview. There you can see the all the hardware items. As in `lshw` you can use a kind of class as well, in `hwinfo` this is called `hw_item`. Use `hwinfo --<hw_item>` to display this specific list of items. e.g. `hwinfo --network` gives:

```
44: None 05.0: 10701 Ethernet
  [Created at net.124]
  Unique ID: AmUr.ndpeucax6V1
  Parent ID: 75la.9oyU3hChuy1
  SysFS ID: /class/net/eth5
  SysFS Device Link: /devices/pci0000:00/0000:00:04.0/0000:04:00.0
  Hardware Class: network interface
  Model: "Ethernet network interface"
  Driver: "ixgbe"
  Driver Modules: "ixgbe"
  Device File: eth5
  HW Address: 00:1b:21:27:36:05
  Link detected: yes
  Config Status: cfg=new, avail=yes, need=no, active=unknown
  Attached to: #25 (Ethernet controller)
```

In addition, you can also grep the boot messages. For example, to get information about the CPU:

```
dmesg | grep -v "cpu"
```

While the tools usually work fine, it is recommended to determine the model of your server or motherboard, and download the documentation from the Internet. The drivers, libraries or kernel used by the above tools may not support all probes, and the manual may give specific details of how to configure the hardware for performance tuning (including helpful pictures of where each part is located on the motherboard). Further tools include:

- dmidecode
- discover
- lshal
- systool
- cat /proc/cpuinfo

6.2 PCI Bus Probing

`lshw` and `hwinfo` can both query PCI, IDE and SCSI busses, but specific tools for each bus, such as lspci and lsusb often give more details. Here is a list of other tools:

- lspci
- lsusb
- lsdev
- scanpci

Especially `lspci -vv` gives you lots of information, when run as root. With the extra option `-s` you can also select a spcific device. With the option `lspci -vt` you can see the three how the busses are connected. Sometimes this can give you a quick overview and clear picture that multiple network cards use the same bus, as in this example:

```
opti@flits:~$ lspci -vt
-[0000:00]-+-00.0  Intel Corporation 5520 I/O Hub to ESI Port
           +-01.0-[0000:01]--+-00.0  Broadcom Corporation NetXtreme II BCM5709 Gigabit Ethernet
           |                 \-00.1  Broadcom Corporation NetXtreme II BCM5709 Gigabit Ethernet
           +-03.0-[0000:03]--
           +-04.0-[0000:04]----00.0  Intel Corporation 82598EB 10-Gigabit AF Network Connection
```

6.3 Kernel Modules

In one of our previous outputs we saw that a module was used for the 10GE interface, this module was named ixgbe. If you want more info about this module (e.g. the driver version which is used), you can use the command modinfo. An example output of this command is (this output is not complete):

```
$ modinfo ixgbe
filename:       /lib/modules/2.6.31-21-generic/kernel/drivers/net/ixgbe/ixgbe.ko
version:        2.0.75.7-NAPI
license:        GPL
description:    Intel(R) 10 Gigabit PCI Express Network Driver
author:         Intel Corporation, <linux.nics@intel.com>
srcversion:     704DFDBC607494AFB438E2A
```

7 Solid State Disks

Solid state disks (SSDs) use non-volatile NAND (flash) memory instead of the rotating magnetic platters of a mechanical hard disk drive (HDD). The advantages of Solid State Disks over Hard Disk Drives are:

- High reliability (due to the lack of moving parts),
- Low energy consumption
- Near-zero access times
- Faster read and write speed

The disadvantages of Solid State Disks are:

- Write Endurance (the number of write operations is limited for flash-based SSD)
- Cost (currently SSD are 10-20 times as expensive as HDD, but it is expected that these prices will drop quickly).

Each memory block in a SSD could only be rewritten 100.000 times before it fails, giving a lifetime of a few years for an individual block. After 2006, SSD are overprovisioned and algorithms are implemented to manage bad blocks, resulting in effective endurance of 1-5 million cycles before the drive as a whole fails. Even with continuous write/erase 24 hours a day, this gives a lifetime of multiple decades. So in practice, write endurance is no longer an issue.

There are two types of flash-based SSDs available on the market, single-level cell (SLC) and multi-level cell (MLC) SSDs. SLC SSDs (such as the Intel X25-E) achieve higher write performance,

while MLC SSDs (such as the Intel X25-V and X25-M) have lower cost and more capacity. The file read performance is roughly the same for MLC and SLC SSDs.

SSD based on DRAM instead of NAND (flash) memory are also available, and have an even better performance. DRAM based SSD require a battery for preserving data, and are roughly 10 times as expensive as flash-based SSDs. DRAM-based SSDs are only used for small datasets with many reads and writes (such as databases) that require extremely high reliability (such as banks).

Disk Type	**Read Speed**	**Write Speed**	**Read Access Time**	**Write Access Time**
High-Capacity Hard Disk Drive	70 MiByte/s	70 MiByte/s	11 ms	6 ms
High-Performance Hard Disk Drive	120 MiByte/s	120 MiByte/s	6 ms	5 ms
Multi-level Cell SSD	250 MiByte/s	70 MiByte/s	0.2 ms	1-20 ms
Single-level Cell SSD	250 MiByte/s	70 MiByte/s	0.2 ms	1-20 ms
DRAM-based SSD	500 MiByte/s	300 MiByte/s	<0.03 ms	0.1 ms

Table 6. Typical sustained disk read and write speed and random access time.

Sequential access write time for flash-based solid state disks is only marginally slower than sequential access read times (the throughput access time reported by manufacturers refer to sequential access times). However, the random access write time is considerable larger, and heavily depends on the file size to be written The reason is that flash-based solid state disks can not modify data in-place. Instead, if a page (4 kiByte) in block (256 kiByte)is to be overwritten, the new page is written in a previously emptied block, and the other 63 pages in the block are copied to the new block, than the data pointer is changed to the new block and the old block is erased. If the page was previously empty, the write can be very fast (typically 250 s), but if the page needs to be overwritten, up to 64 pages must be read and written (typically taking 64 times 25+250 = 17.6 ms) and a block must be deleted (typically taking 2ms).

8 RAID controller

Disks are connected to one or more high-speed disk (SAS+SATA) controllers for best performance. SATA has a bus speed of either 3 Gbit/s or 6 Gbit/s (358 MiByte/s or 715 MByte/s), so it is recommend to only connect a single disk per SATA channel.

When properly tuned, we achieved the best disk performance using software RAID. An untuned software RAID performs slightly worse than hardware RAID. This means we could suffice with a simple disk (SAS+SATA) controller, configured in JOBD (Just A Bunch of Disks) mode: all disks were visible to the operating system.

The software RAID to use depends on the file system. Modern file systems such as ZFS and BtrFS have RAID-like functionality build in. For best performance, use this functionality. For file systems without RAID-like functionality, such as XFS, we achieved better performance with LVM (logical volume manager) than with md (Linux' Multiple Device driver).

File System	**RAID system**
ZFS	raidz2 (ZFS built-in)
BtrFS	built-in RAID
XFS	LVM (separate)

Table 7. Advised Software RAID.

9 File System

The file system and operating system have a major impact on the disk performance, even with the same hardware. Our results have showed that ZFS on FreeBSD clearly gives the best performance. XFS and BtrFS on Linux follows. Due to licensing issues, ZFS is not available on Linux, so it remains unclear if the performance difference is caused by the file system of the operating system. Likely, both determine the performance.

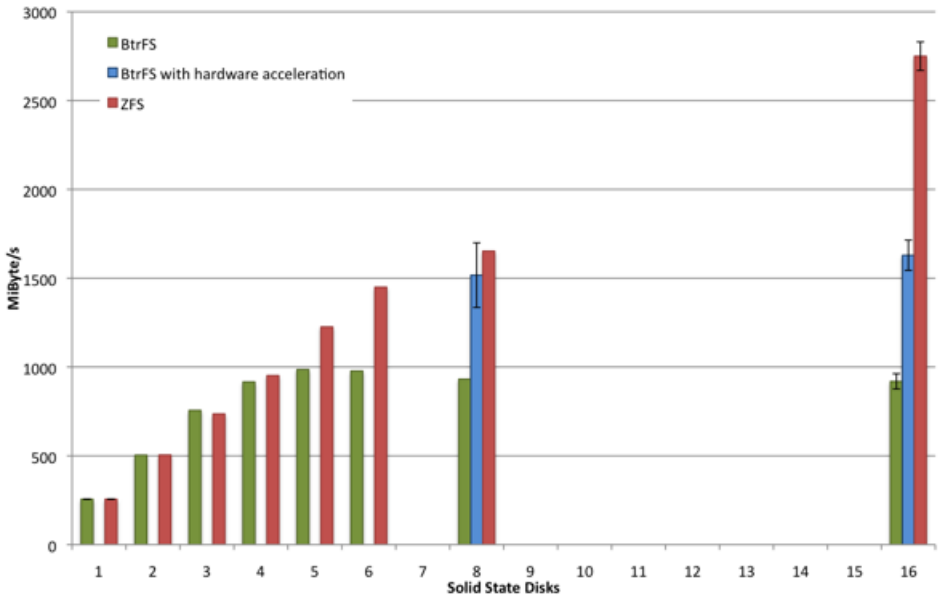

Fig. 3. Sustained read performance with multiple SSD using ZFS and BtrFS.

BtrFS is advertised as the Linux replacement for ZFS. In our experience, this claim is certainly not true yet. This is of course not surprising, given the maturity of ZFS versus the beta state of BtrFS. While BtrFS and XFS can already reach the same performance levels, we found it harder to tune BtrFS. This situation may change in the years to come.

Figure 3 lists the sustained read performance for ZFS and BtrFS. The green bars shows the untuned BtrFS performance, the red bars the untuned ZFS performance. The blue bars show the tuned BtrFS performance, which improved, but not to the levels of ZFS.

Read Performance Test	**8 SSD**	**16 SSD**
Theoretical limit (8/16 times single disk)	2040 MiByte/s	4080 MiByte/s
Block level test with FIO		3301 MiByte/s
ZFS (out of the box)	1653 MiByte/s	2750 MiByte/s
BtrFS with CRC checksumming in hardware	1517 MiByte/s	1629 MiByte/s
XFS	1518 MiByte/s	
BtrFS without checksumming	1563 MiByte/s	1586 MiByte/s
BtrFS with latest patches		1012 MiByte/s
BtrFS untuned read speed	932 MiByte/s	919 MiByte/s
BtrFS untuned write speed		1596 MiByte/s

Table 8. Performance levels for different sustained read disk tests (empty blocks have not been measured, all test were repeated four times, standard deviation is 5-10% of the reported value).

Most file systems, including ZFS and BtrFS, are still geared towards optimization for hard disk drives. The issues with hard disk drives (e.g. fragmentation) are very different from the issues with solid state disks (e.g. inability to directly overwrite data). We expect to see performance improvements when solid state disks become the dominant storage system for consumer hardware, and file systems are subsequently adjusted.

9.1 BtrFS Tuning

It is expected that BtrFS performance will greatly increase, so best you consult the developers to tune your performance. (For example, in our case, the untuned read performance was worse than the write performance, so there is lots of room for improvement).

If you use solid state disks, the ssd option may improve performance (in our setup, it made no difference):

```
mount -t btrfs -o ssd /dev/sdb /mnt/ssd
```

The most significant performance hit we encountered was due to the checksumming of each block done by BtrFS. Without hardware support, this posed a significant burden on the CPU. Intel has hardware support for checksumming, but for some reason, their module was not loaded by default. The checksumming module without hardware support is named `crc32c`, the checksumming module with Intel hardware support is named `crc32c_intel`.

To use the module with hardware support, add `crc32c_intel` to `/etc/modules` and reboot.

Or without a reboot:

```
umount /mnt/btrfs_disk1 /mnt/btrfs_disk2 ...
rmmod btrfs
rmmod libcrc32c
rmmod crc32c
modprobe crc32c_intel
mount /mnt/btrfs_disk1 /mnt/btrfs_disk2 ...
```

9.2 Block size and RAID

The block size for hard disk drives is typically 512 bytes. The block size for solid state disks is 512 kiByte, the page size for SSD is 4 kiByte, both much larger than the HDD block size.

For best performance, the file system block size, the RAID chunk size, and the application I/O size should all correspond.

file system block size the smallest node size (in bytes) on a file system. On most file systems this is between 512 byte and 4 kiByte. On Linux, the file system block size is maximised by the page size of the kernel memory (4 kiByte in default kernels).

For most high-performance applications, you should chose a file system block size of 4 kiByte.

disk count the number of disks in a RAID array

data disk count the number of data bearing disks in the RAID array. For example, a 4-disk RAID-5 array has 3 data-bearing disks.

physical block size the smallest size (in bytes) for a low-level read or write operation on a disk. For HDD, this is typically 512 bytes. For SSD, this is 4 kiByte (for read operations) or 512 kiByte (for write operations).

The above data is fixed, and you need it to determine the optimal chunk size:

chunk size (also known as stripe unit size) the size (in bytes) of a physical chunk on a single disk.

stripe size the size (in bytes) of a stripe across all disks. Equal to the chunk size times the data disk count.

stride The number of file system blocks in a chunk (some sources use stride for the number physical blocks in a chunk).

stripe width The number of file system blocks in a stripe (some sources use stride for the number physical blocks in a stripe).

Choosing the chunk size (and thus stripe size, stride and stripe width) is key to performance tuning, and depends on your application.

If your application has many read/write operations with relative small amounts of data, you should make sure that each I/O operation is handled by a single disk, and all I/O operations are spread over the disks. In this case, choose your chunk size equal to or larger than the average I/O request size of your application.

If your application has few read/write operations with relative large amounts of data, you should make sure that each I/O operation is distributed over all disks. In this case, choose your stripe size equal to or smaller than the average I/O request size of your application.

For optimal performance, also make certain that:

- The chunk size is a multiple of the physical block size of the disk.
- The chunk size is a multiple of the file system block size

For example, let's imagine a RAID-0 array of 8 SSDs. In this case, the physical block size is 4 kiByte, thus the chunk size should be a multiple of 4 kiByte. Furthermore, imagine an application that performs read() operations with 16 MiByte of data. Thus we need to choose the stripe size equal to or less than 16 MiByte, or a chunk size equal to or less than 2 MiByte. So we can choose between a chunk sizes of 4, 8, 16, 32, 64, 128, 256, 512, 1024 or 2048 kiByte. It is best to test a few of these scenarios, and pick the one that gives the best performance.

9.3 File Deletion on Solid State Disks

If a file is deleted, the file system typically removes the pointer in the B-tree, but does not really erase the data on disk itself. As mentioned earlier, solid state disks can not directly overwrite data, and in general, performance decreases if a given block is re-used often. If the SSD firmware is told which blocks are actually erased, the algorithm can more effectively spread the wear levelling, resulting in a performance improvement of up to 10-20%.

The **trim** function in the `data set management` command of the ATA specification was introduced in 2007 to let the Operating System signal the SSD firmware which data blocks have been deleted by the file system. The helps the SSD firmware to manage the data blocks on the disk more efficiently.

Operating support for the TRIM function was added to most operating systems in 2010. Further information can be found in Intel's Solid-State Drive Optimizer[1], which adds TRIM support for Windows.

10 Disk Performance Tools

`IOzone` is a widely used tool to test read/write performance of a file system. `fio` is a tool to test more low-level disk performance.

IOzone requires that a write test that creates a test file before any read tests can be performed. For example: For example, a first run of iozone is:

```
iozone -s 32G -r 1024 -i 0 -i 1 -w -f /mnt/ssd/iozone.tmp
```

With consecutive runs of iozone (only read testing, no write tests, using existing test file)

```
iozone -s 32G -r 1024 -i 1 -w -f /mnt/ssd/iozone.tmp
```

While testing, make sure that your test file is significantly larger than the cache and system memory, otherwise you risk testing your memory performance instead of the file performance (unless of course you want to test the caching performance).

[1] `http://download.intel.com/design/flash/nand/mainstream/Intel_SSD_Optimizer_White_Paper.pdf`

11 CPU and Interrupt Affinity

Modern servers include multiple CPUs (and GPUs). For best performance, the processes should be distributed over the different CPUs. Pinning of a certain process to a certain CPU is called CPU affinity.

Similarly, memory affinity is pinning a certain process to a given memory region; interrupt affinity is pinning certain interrupts to a given CPU; and disk affinity is pinning certain files on a given physical disk.

Disk affinity is trivial. CPU affinity and interrupt affinity can improve performance, especially in a system with limited bandwidth on the system or front size busses.

11.1 Monitoring Tools

To show the CPU usage use top or mpstat:

- `top` (press 1 to show all individual CPUs)
- `mpstat -A` (also shows interrupts)

To show the interrupts, use `mpstat` as above or (on Linux):

```
cat /proc/interrupts
```

11.2 CPU Affinity

Use `taskset` to pin a certain process to a certain processor. In this example, `myapp` is pinned to CPU ID 02, and `iperf` to either CPU ID 01 or ID 02 (the given number is a mask). To get the IDs of the CPUs (and cores), `cat /proc/cpuinfo`.

```
taskset 02 ./myapp
taskset 03 iperf -i 4 -t 300 -c 145.146.100.133  -l 62K -w 350K
```

In our experience, CPU affinity gives stable (and thus repeatable) results, but only increases the performance by small margin.

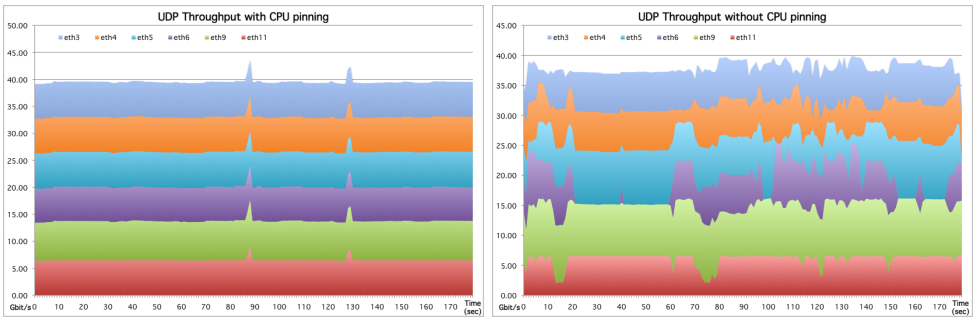

Fig. 4. Difference between pinning and no pinning of 6 UDP streams.

11.3 Interrupt Affinity

Interrupts signal a CPU for incoming (I/O) events that need to be handled. For example, incoming network traffic yields an interrupt. On Linux, the `irqbalance` can be used to automatically distribute incoming interrupts among the processors. Without `irqbalance`, all interrupts are by default handled by CPU ID 00. Interrupt affinity is also known as IRQ affinity.

If you want to handle interrupt affinity manually, disable `irqbalance`, and specify a given CPU ID mask for each process.

First determine which IRQ you want to pin:

```
cat /proc/interrupts
```

To pin IRQ 63 to CPUs 0 to 2 (bit mask 0x07):

```
echo 07 > /proc/irq/31/smp_affinity
```

12 Network Performance

There are three widely used network throughput measurement tools. Which one to use is mainly a matter of taste. The features of all three are very similar:

- iperf
- nuttcp
- netperf

In order to measure network throughput between two servers, you need to start the tool in server mode on one side and in client mode on the other side. Below is an example of how to measure TCP throughput with iperf. The measurement is run for 100 seconds (-t 100), and every second (-i 1) the current throughput is printed. In order to get high throughput you need to use large packet sizes, e.g. 63 KB (-l 63K). It is also important to use a large enough socket buffer size. Try increasing the buffer size until you get optimum throughput. The example below uses a buffer size of 200 KB (-w 200K).

```
sender:     iperf -t 100 -i 1 -c 10.0.0.1 -l 63K -w 200K
receiver:   iperf -s -w 200K
```

The equivalent `nuttcp` syntax is:

```
sender:     nuttcp -T 100 -i 1 -t -l 63k -w 200k -R 10G 10.0.0.1
receiver:   nuttcp -S
```

For UDP the options are:

```
sender:     iperf -u -i 1 -t 100 -c 10.0.0.1 -l 63K -w 200K -b 10G
receiver:   iperf -u -s -l 63K -w 200K
sender:     nuttcp -u -i 1 -T 100 -t -l 63k -w 200k -R 10G 10.0.0.1
receiver:   nuttcp -u -S
```

`netstat` gives useful information about network I/O statistics. The example below shows how to get statistics about all incoming out outgoing packets (IP part), and the various protocols (ICMP, TCP, UDP). Thing to look for are IP packets dropped, IP fragments failed, IP re-assemblies required and ok, UDP receive errors and SndbufErrors. Also check that IP packet statistics correspond to TCP or UDP statistics (note that several IP datagrams might be assembled into 1 UDP packet).

```
 $ netstat -s
 Ip:
   94923448 total packets received
   0 forwarded
   0 incoming packets discarded
   94916410 incoming packets delivered
   74733541 requests sent out
   273 outgoing packets dropped
   7360 reassemblies required
   326 packets reassembled ok
   21191575 fragments received ok
   1952857 fragments failed
   141571054 fragments created
 Icmp:
   2404 ICMP messages received
   1815 input ICMP message failed.
   ICMP input histogram:
       destination unreachable: 423
       timeout in transit: 1797
       echo requests: 139
       echo replies: 45
```

```
  313 ICMP messages sent
  0 ICMP messages failed
  ICMP output histogram:
      destination unreachable: 102
      echo request: 72
      echo replies: 139
IcmpMsg:
      InType0: 45
      InType3: 423
      InType8: 139
      InType11: 1797
      OutType0: 139
      OutType3: 102
      OutType8: 72
Tcp:
  2824 active connections openings
  280 passive connection openings
  10 failed connection attempts
  512 connection resets received
  1 connections established
  95658252 segments received
  51102689 segments send out
  604892 segments retransmited
  182 bad segments received.
  1311 resets sent
Udp:
  5269 packets received
  0 packets to unknown port received.
  0 packet receive errors
  23775230 packets sent
  SndbufErrors: 2
UdpLite:
TcpExt:
  27 ICMP packets dropped because they were out-of-window
  986 TCP sockets finished time wait in fast timer
  4 time wait sockets recycled by time stamp
  23209 delayed acks sent
  5 delayed acks further delayed because of locked socket
  Quick ack mode was activated 440 times
  1610 packets directly queued to recvmsg prequeue.
  198356 bytes directly in process context from backlog
  750668 bytes directly received in process context from prequeue
  4080508 packet headers predicted
  875 packets header predicted and directly queued to user
  38019718 acknowledgments not containing data payload received
  54389232 predicted acknowledgments
  170871 times recovered from packet loss due to fast retransmit
  2417 timeouts after reno fast retransmit
  120 timeouts in loss state
  174084 fast retransmits
  307348 retransmits in slow start
  786 other TCP timeouts
  119974 classic Reno fast retransmits failed
  41 connections reset due to unexpected data
  5 connections reset due to early user close
  19 connections aborted due to timeout
IpExt:
  InMcastPkts: 4741
  OutMcastPkts: 347
  InBcastPkts: 108
  InOctets: 806747296
  OutOctets: -812905527
  InMcastOctets: 420338
  OutMcastOctets: 29392
  InBcastOctets: 32170
```

`ethtool -S` gives packet statistic information of the NIC. Incoming and outgoing packets in the NIC should correspond to the TCP/IP statistics of the kernel as shown by `netstat -s`.

13 Further Reading

Further information on server performance tuning is available at:
`https://noc.sara.nl/wiki/Server_Performance_Tuning`
`http://kb.pert.geant.net/PERTKB/WebHome`
`http://fasterdata.es.net`

http://www.redhat.com/promo/summit/2008/downloads/pdf/Thursday/Mark_Wagner.pdf

Acknowledgments

We like to thank all who contributed to this effort, in particular our colleagues Sander Boele, Pieter de Boer, Igor Idziejczak, Walter de Jong, Tijs de Kler, Mark Meijerink, Paul Melis, Hanno Pet, Mark van de Sanden, Dennis Stam, and Paul Wielinga.

This work was funded by SURFnet as part of the GigaPort3 project.