Design and Build your Data Transfer Node

Eric Pouyoul
lomax@es.net

Designing and building your own Data Transfer Node will allow you to have a performance server at a “low” cost.
Mission: move data fast

Forwards large amount of data from and to the site resources.
Simple Workflow: Sender and Receiver

- Sequential I/O
- File transfer model (large buffers)
- CPU is dedicated to the data transfer

*High Performance DTN function requires all elements to be carefully designed and tuned.*
Hardware matters

The true sources and destinations of data are often large data centers and super computers.

A slice can/could be used to perform the DTN function.

Most efficient but difficult, but not impossible to deploy.

Dedicated, commodity, servers can make excellent front-end of larger systems.
Commodity Servers: you are on your own.

Custom design: DTN’s require specific networking and I/O controllers.

*A “non high performance” dual port NIC cannot achieve line speed on both ports at the same time.*

Performance tuning: default system settings will not be adequate.

*I/O performance before tuning 700MB/sec. After tuning up to 1.6GB/sec.*

Maintenance: design choices impacts stability and operation of the DTN.

*Replacing an SSD PCI card requires to unrack and open the server.*
Designing a system for the DTN workflow

A DTN moves data from and to the network

Step 1 Storage: what type, capacity and if needed, controller.
Step 2 Networking: what protocols, optimization and NIC.
Step 3 Motherboard: what is required to move data between the subsystem.
Step 4 Operation support: monitoring, remote access.
Step 1: Storage Architectures

- **Storage system**: Fiber, Infiniband
- **Distributed file system**: Infiniband, Ethernet
- **Local storage**: Raid, SSD
- **Isolated storage system**: scp, gridftp, proprietary
- **DTN**
Performance of the storage subsystem

Performance is based upon various elements and will always be limited by one of them:

- Storage medium (HD, SSD, RAM)
- Controllers (FC, Infiniband, Ethernet, RAID)
- Server bus (PCI)
- File system (EXT4, BTRFS)
Storage Systems

Pros
• May already be deployed (legacy)
• Can scale up
• Highly Available

Cons
• Controller Bottleneck
• Expensive
• Large footprint
Networked Storages

Ethernet (iSCSI) or Infiniband (SRP) provide networked access to raw storage:

• Flexible storage scalability
• Virtualization
• Especially with SRP, high controller performance

• Requires specific controller
• Requires specific software stack
Local storage: RAID

Capacity of hard drive has increased
Form factor has decreased
Reliability has increased
Hardware RAID is efficient
Inexpensive

Local RAID storage is ideal for custom design DTN
Local Storage: RAID Levels

- RAID0: stripes data. Best performance, no reliability
- RAID1, 10: mirrors data: Best reliability, half capacity, half performance
- RAID5: Decent reliability, 2/3 of capacity. Performance varies.
- RAID6: Similar to RAID5, but supports two disk failures.
- Other RAID: vendor specific. Dedicated to a given workflow
- File System RAID: BSD’s ZFS and Linux’ BTRFS
RAID: Great (cheap) but Experiment First

- RAID is a bottleneck!
- Performance depends on RAID engine

- Select the right RAID Level
  - RAID0: need best I/O performance but can afford losing all dataset.
  - RAID5/6: need reliability, can afford to only have 2/3 of capacity and performance of RAID0.

- Select the right RAID Controller
- Plan for expansion
- Experiment on prototype first.
RAID Controllers

• Often optimized for a given workload, rarely for performance.
• RAID0 requires less CPU than other RAID levels.
• The CPU required to process queries is a factor of the number of drives.
• Each controller has its own best configuration forcing to make compromises.
SSD Storage: the wow factor
SSD: Current State

- 6 GB/sec read (PCIe 2.0 x16)! More to expect with PCIe 3.0.
- Acceptable/Excellent MTBF
- Still more expensive
- Potentially harder to deploy within standard IT
- Migration path / Hybrid – Needs high end RAID Controllers
Networking Subsystem
Network Interface Controller

NIC’s are not identical with weaknesses and strengths:

- True dual port support
- Performance tuning
- CRC offloading
- Protocol offloading (TCP, RDMA…)
- Driver support

Always make sure that the optics are compatible with the NIC.
Motherboard

The motherboard provides all the buses that connects the CPU’s, memory and controller together. It is a critical part of the server design since it can become a bottleneck.

When selecting a motherboard, pay attention to:

- PCIe subsystem
- Memory type and size
- Architecture (AMD vs Intel)
- Chipset
PCle Subsystem (1)
PCIe subsystem (2)

PCIe bandwidth

- PCIe 2.0: (500 MB/sec per lane)
- Typical up to 4 GB/sec (8 lanes or x8)
- High end up to 8 GB/sec (16 lanes or x16)
- PCIe 3.0: doubles bandwidth

Motherboards provide PCIe slots. Slots are defined by:

**Form factor:** that is the length of the slots, referred as the number of PCI lane is can support. For instance, a 16 lanes controller’s connector is twice as long as a 8 lane controller.

**Wired lanes:** not all lanes of the slot may be wired. For instance, some 16 lanes controller may only have 8 lanes wired

**Plan for enough PCI slots with appropriate number of lanes.**
Intel: SuperMicro X8DAH+-F
AMD: SuperMicro H8DG6-F
Low performance: SuperMicro X7DWT
Memory

Memory bandwidth (stream benchmark)

• typical 8 GB/sec
• High end 31 GB/sec

Memory type:

• DDR2 if moderate memory usage, DDR3 if heavy memory usage.
• Be aware of best price / capacity.
• Always follow motherboard, chipset recommendations for best performance.

Memory Size:

• Enough memory for application: never swap
• Plan for I/O cache (raw, files system) if needed
AMD or Intel?

- Currently, Intel has a faster bus (QPI) than AMD’s HT’s
- Faster clock on Intel
- More cores on AMD
- Memory can be cheaper on AMD (AMD support DDR2)
- AMD typically supports architecture much longer than Intel (backward compatibility).

AMD and Intel alternates as the leader in performance computing (look at manufacturing problems, etc)
Tuning the Data Transfer Host
Tuning

Defaults are usually not appropriate for performance.

What needs to be tuned:

• BIOS
• Firmware
• Device Drivers
• Networking
• File System
• Application
Tuning Methodology

At any given time, one element in the system is preventing it from going faster.

**Step 1:** Unit Tuning: focusing on the DTN workflow, experiment and adjust element by element until reaching the maximum bare metal performance.

**Step 2:** Run application (all elements are used) and refine tuning until reaching goal performance.
BIOS Tuning

*Each BIOS, even from the same vendor is different. Experimentation is necessary.*

- Default as often incorrect
- Hyperthreading: disable, we want real cores.
- CPU frequency scaling: disable, as well as all energy saving features: we want the full power all the time.
- Check memory bus speed (force to max.)

- Configure remote console, remote power control (IPMI): you will reboot your server many times per hour.
Disk Performance Issues

Disks are mechanical data storages. Their performance depend on:

- Disk speed (Rotation Per Minute): 7,200, 10,000 or 15,000 rpm
- Geometry
- Sequential and random access (head seek)
- Sustained and Peak performance

How to build a high performance I/O subsystem:

- Partitioning (short-stroking)
- Workflow optimization (readahead, filesystem)
- Use of caches
- More disks!
RAID and Performance

- Right RAID Level?
- Need a better controller?
- Need better drives?
- Adjust strip size when possible
- Disable any "smart" controller built-in options

*Experiment in various configuration: each RAID controller has a sweet spot.*
Tool: vmstat

From man page:

reports information about processes, memory, paging, block IO, traps, and cpu activity.

• Showed true I/O operation
• Shows CPU bottlenecks
• Shows memory usage
• Shows locks

$ vmstat 1

procs ----------memory-------- swap-- io---- system-- cpu------
  r  b  swpd free  buff  cache si  so  bi  bo  in  cs  us  sy  id  wa  st
 0 0 0 22751248 192800 1017000 0 0 0 0 4 7 0 0 100 0 0
I/O testing tool: dd

From man page: “convert and copy a file”
- Generate I/O traffic
- Control over block size, seek
- Input and output agnostic (raw or file)
- Can be used in parallel

```
$ dd if=/storage/data1/test-file1 of=/dev/null bs=4k
13631488+0 records in
13631488+0 records out
55834574848 bytes (56 GB) copied, 54.1224 seconds, 1.0 GB/s
```
Example of a “dd test”

```bash
# dd of=/dev/null if=/storage/data1/test-file1 bs=4k &
# dd of=/dev/null if=/storage/data1/test-file2 bs=4k &

# dd of=/dev/null if=/storage/data2/test-file1 bs=4k &
# dd of=/dev/null if=/storage/data2/test-file2 bs=4k &

# dd of=/dev/null if=/storage/data3/test-file1 bs=4k &
# dd of=/dev/null if=/storage/data3/test-file2 bs=4k &
```
Example vmstat / dd

# vmstat 1

procs --------memory-------- ---swap-- -----io---- --system-- -----cpu------

```
       r  b  swpd   free   buff  cache  si  so  bi  bo  in  cs  us  sy  id  wa  st
---------- --------  --------  --------  --------  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---
 6  0  0 150132 215204  23428260 0  0  0  0  16431 2245  0  13  86  0  0
 2  3  0 1692948 218924  21920000 0  0  4428 499712 24599 5341  1  29  65  6  0
 2  5  0 1610216 222512  22001264 0  0  3532 725012 25230 5363  0  15  75 10  0
 4  5  0 720020  224532  22865412 0  0  2048 847296 24566 4277  0  13  65 22  0
 3  7  0 1917556 225440  21686980 0  0  1672 1099036 27333 4314  0  17  60 23  0
 6  7  0 1419324 225496  22180252 0  0  1312704 29410 25386  0  24  45 31  0
 3  6  0 391860  225560  23182336 0  0  4  1261536 25797 27532  0  20  48 32  0
 8  4  0  80624  224672  23486864 0  0  4  1296932 26799 3373  0  22  52 26  0
 3  6  0  88860  224184  23475516 0  0  4  1322248 28338 3529  0  22  51 27  0
```

1/22/12
I/O Testing Tips

- Two windows, one with `dd`, one with `vmstat`
- Influence of the read and write caches
- Flush caches before running tests:
  ```
  # echo 3 > /proc/sys/vm/drop_caches
  ```
- Discussion on data size: three times the memory size
- Influences of the block size: use block size that matches application’s pattern
- Remote Console (IPMI)
Linux I/O Scheduler

- I/O scheduler: different policies. Default policy is “fair” meaning bad for performance. Typically deadline scheduler is better for performance, but favors the most I/O hungry application.

In /boot/grub/grub.conf:

```
title CentOS (2.6.35.7)
root (hd0,0)
kernl /vmlinuz-2.6.35.7 ro root=/dev/VolGroup00/LogVol00 rhgb quiet
elevator=deadline
initrd /initrd-2.6.35.7.img
```
I/O Tuning: readahead

- Necessary optimization when workload is mostly sequential read
- Needs to be experimented with
- Does not always play nice with hardware optimization (but is often better than hardware optimization)
- Needs to be done at each boot of the server (add configuration in /etc/rc.local)
- Interesting reading

```
/sbin/blockdev --setra 262144 /dev/sdb
/sbin/blockdev --setra 262144 /dev/sdc
/sbin/blockdev --setra 262144 /dev/sdd
```
File Systems Performance

- Very few file systems are designed for high performance
- EXT4 is currently the fastest production file system for Linux.
- ZFS provides “smart” software RAID and compression on Solaris
- BTRFS: bleeding edge, integrates RAID and compression on Linux
- File systems must be tuned for performance
- Compromise performance versus data reliability: be careful for what you ask for!
File System Optimization (1)

- File System independent optimization (in /etc/fstab)
  /dev/sdb1 /storage/data1 ext4 noatime,nodiratime 0 0

- File System specific optimization (EXT4)
  /dev/sdb1 /storage/data1 ext4
  inode_readahead_blks=64,data=writeback,nobh,barrier=0,commit=300,noatime,nodiratime 0 0

Inode_readahead: useful when directories have lots of files
Data=writeback: metadata is written onto the disk in a “lazy” mode
barrier=0: does no longer enforce journal write ordering.
File System Optimization (2)

- Necessary in order to get performance close to bare metal
- Be careful what you ask for: some of the optimization may render the file system less reliable in case of crashes
NIC Tuning

As the bandwidth of a NIC goes up (10G, 40G), it becomes critical to fine tune the NIC.

- Lot of interrupts per second (IRQ)
- Network protocols require to process data (CRC) and copy data from/to application’s space.
- NIC may interfere with other components such as the RAID controller.

*Do not forget to tune TCP and other network parameters as described previously.*
Handling flood of frames: Interrupt Affinity

- Interrupts are triggered by I/O cards (storage, network). High performance means lot of interrupts per seconds
- Interrupt handlers are executed on a core
- By default, core 0 gets all the interrupt, or, interrupt are dispatched in a round-robin fashion among the core: both is bad for performance:
  - Core 0 get all interrupt: with very fast I/O, the core is overwhelmed and becomes a bottleneck
  - Round-robin dispatch: very likely, the core that executes the interrupt handler will not have the code in its L1 cache.
  - Two different I/O channels may end up on the same core.
A simple solution: interrupt binding

- Each interrupt is statically bound to a given core (network -> core 1, disk -> core 2)
- Works well, but can become an headache and does not fully solve the problem: one very fast card can still overwhelm the core.
- Needs to bind application to the same cores for best optimization: what about multi-threaded applications, for which we want one thread = one core?
PCI optimization: MSI-X

- Extension to MSI (Message Signaled Interrupts)
- Increases the number of interrupt “pins” per card
- Associates rx/tx queues to a given core
- Allows to stitch together on the same core, the thread that runs the program and the asynchronous event it may receive (incoming network packets, asynchronous I/O…), resulting in maximizing L1 cache hit.
- Requires Chipset, card, and operating support.
- Optimized for Linux’ kernel > 2.6.26
- This is a major optimization: on a system with 4 x 10G ethernet, performance gain can be up to 20%
/proc/interrupts

# cat /proc/interrupts displays interrupts statistics on which core each of the interrupts are being executed.

- Find cores that are overloaded with interrupts
- Find the interrupts number of given queues (per interface)

By default, the Linux distribution is configured for automatically balance IRQ’s across cores. This must be disable:

The linux service irqbalance must be turned off:

# chkconfig irqbalanced off
/proc/irq/<number>/smp_affinity

For a given interrupt, it is possible to know on which core(s) it is bound:

```
# cat /proc/irq/32/smp_affinity
```

will return a cpu mask in hex. Examples of masks are:

- 2 (hex) = 10 (bin) = core 1 (first core is 0)
- 3 (hex) = 11 (bin) = core 0 and core 1
- ffff (hex) = 11111111 11111111 = all cores

Binding IRQ 32 to core 7 is done by:

```
# echo 80 > /proc/irq/32/smp_affinity
```

Core 7 = 10000000 (bin) = 80 (hex)
Interrupt Coalescence

Avoid flooding the host system with too many interrupts, packets are collected and one single interrupt is generated for multiple packets.

- Not all NIC support it
- 75-100 micro-seconds timeout
- Can be critical for high performance NIC (10Gb, 40Gb…)
TCP Autotuning Settings:
http://fasterdata.es.net/TCP-Tuning/

Linux 2.6: add to /etc/sysctl.conf

```
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
# autotuning min, default, and max number of bytes to use
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 65536 16777216
```

FreeBSD 7.0+: add to /etc/sysctl.conf

```
net.inet.tcp.sendbuf_max=16777216
net.inet.tcp.recvbuf_max=16777216
```

Mac OSX: add to /etc/sysctl.conf

```
kern.ipc.maxsockbuf=16777216
net.inet.tcp.sendspace=8388608
net.inet.tcp.recvspace=8388608
```

Windows XP

- use “DrTCP” (http://www.dslreports.com/drtcp/) to modify registry settings to increase TCP buffers

Windows Vista/Windows 7: autotunes by default, 16M Buffers
Selecting TCP Congestion Control in Linux

To determine current configuration:

- sysctl -a | grep congestion
- net.ipv4.tcp_congestion_control = cubic
- net.ipv4.tcp_available_congestion_control = cubic reno

Use /etc/sysctl.conf to set to any available congested congestion control.

Supported options (may need to enabled by default in your kernel):

- CUBIC, BIC, HTCP, HSTCP, STCP, LTCP, more..
- E.g.: Centos 5.5 includes these:
  - CUBIC, HSTCP, HTCP, HYBLA, STCP, VEGAS, VENO, Westwood

Use modprobe to add:

- /sbin/modprobe tcp_htcp
- /sbin/modprobe tcp_cubic
Additional Host Tuning for Linux

Linux by default caches ssthresh, so one transfer with lots of congestion will throttle future transfers. To turn that off set:

\[ \text{net.ipv4.tcp_no_metrics_save} = 1 \]

Also should change this for 10GE

\[ \text{net.core.netdev_max_backlog} = 250000 \]

Warning on Large MTUs:

- If you have configured your Linux host to use 9K MTUs, but the MTU discovery reduces this to 1500 byte packets, then you actually need 9/1.5 = 6 times more buffer space in order to fill the pipe.

- Some device drivers only allocate memory in power of two sizes, so you may even need 16/1.5 = 11 times more buffer space!
Application Tuning

• Depends on application

• Bind the application threads to the right core: the thread that is sending or receiving from the network should be running on the same core as the IRQ for that network interface. (the unix command `taskset` is useful)

• Threads that are doing disk I/O should be running on the same core as where the RAID controller IRQ is bound.

• Applications must not use more memory than what is physically available (no swap).