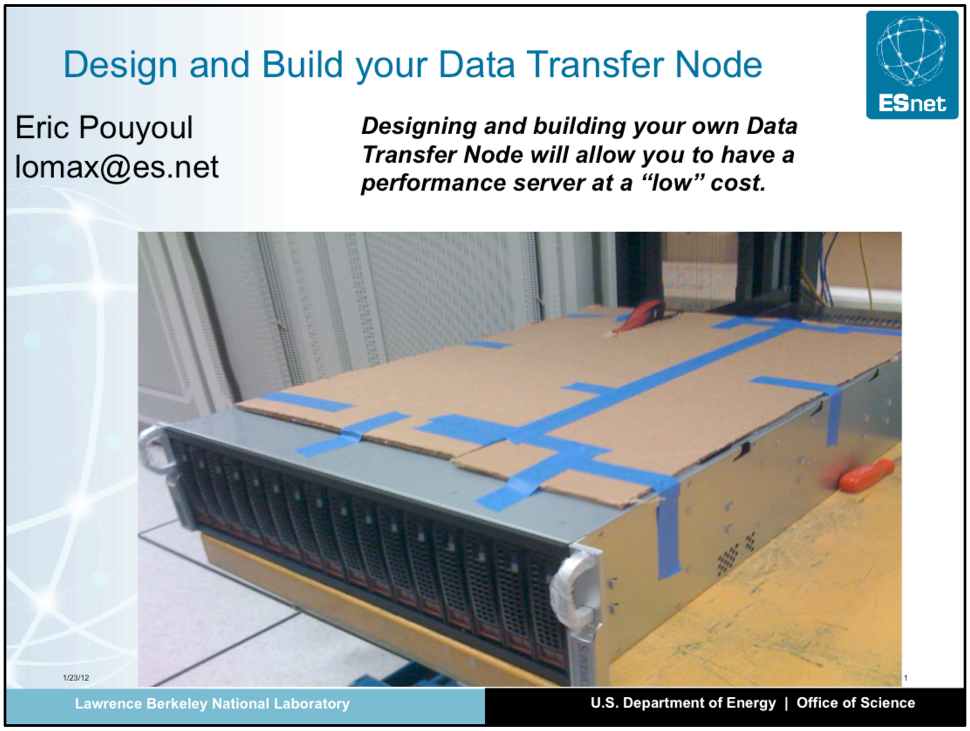
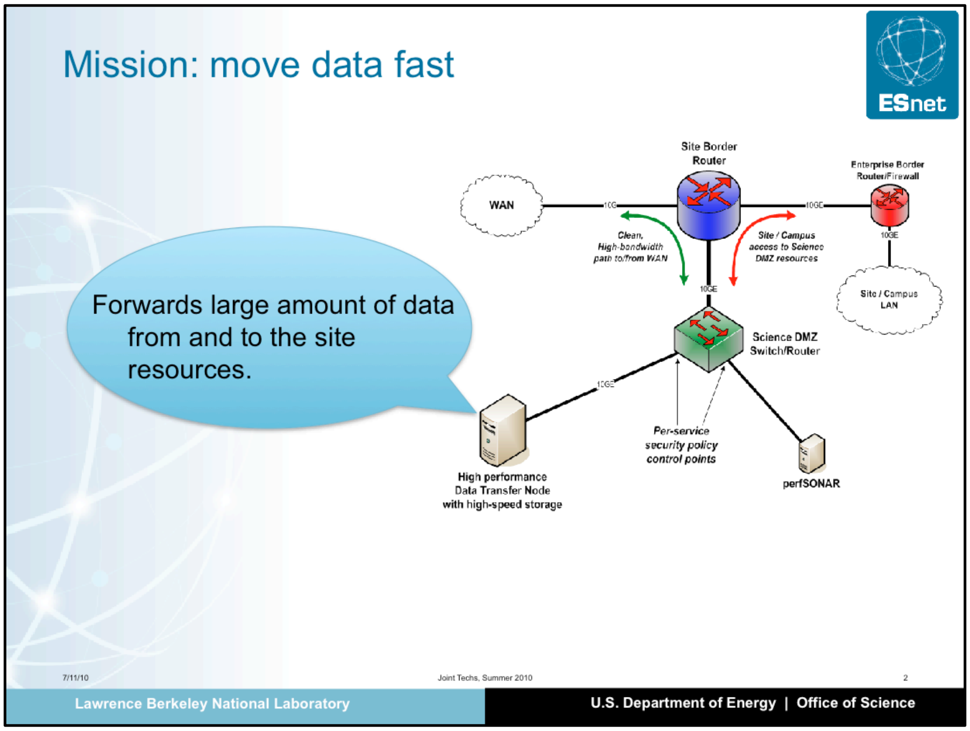


This section of the tutorial focuses on how to design and build a performance server that is dedicated to the Data Transfer function.

First we will look at the various hardware components that makes up a server, and what matters when selecting them. While we will glance over some high end, super computing, hardware, the spirit of this tutorial is have a do it yourself approach, using commodity hardware.

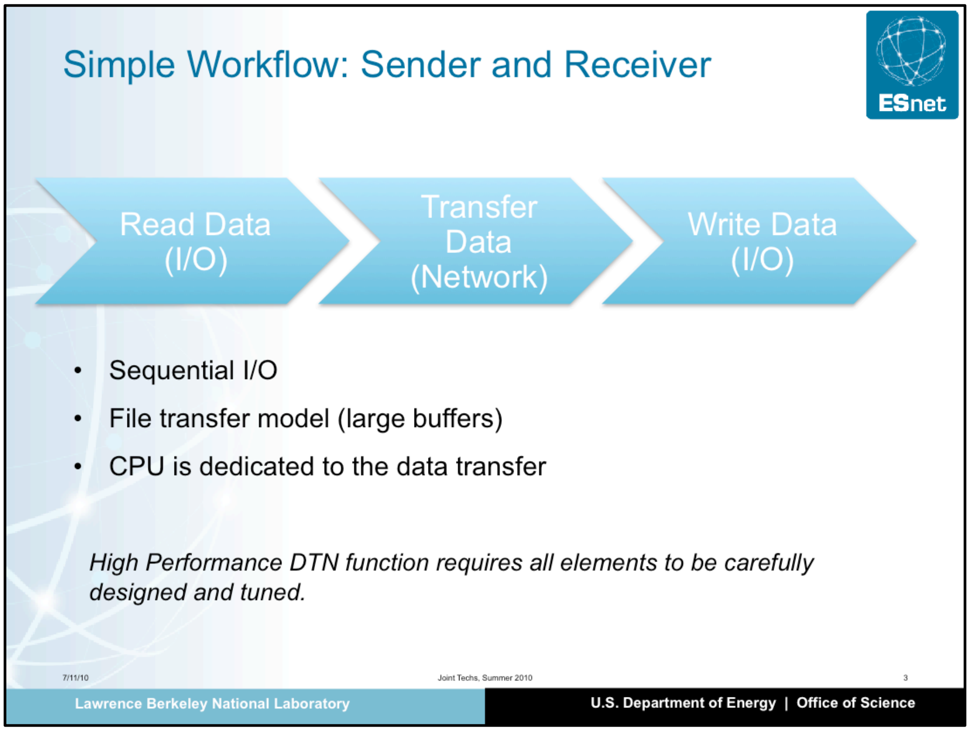
Second, we will discuss the configuration and tuning of the server, so it can perform as expected.



A Data Transfer Node has only one mission to fulfill: send large amount of data across thousands of mile as quickly as possible.

That means that the goal is to fill up the network as closely as possible as line rate.

Another consideration, when designing DTN's is deployment: because of the network topology of a Science DMZ, DTN's sometimes need to be located in racks with very little space available. Density may matter.



A Data Transfer Node is not a processing, rendering, server. Its only workflow is:

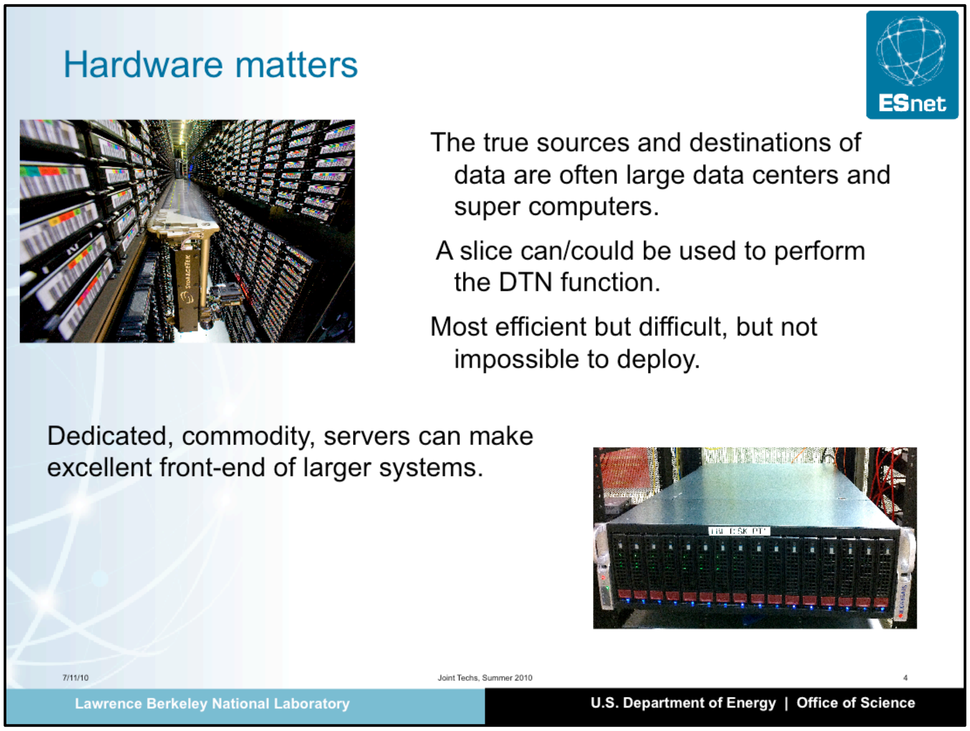
Sender host:

1) read data from the storage subsystem
2) send it to the receiver host

Receiver host:

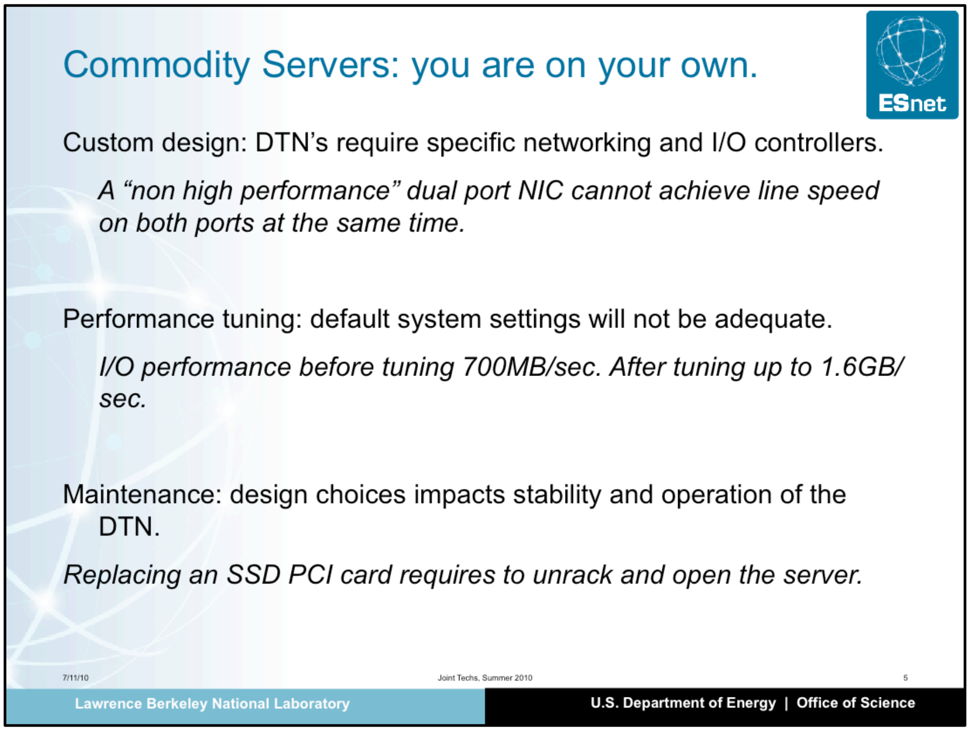
1) read data from the network
2) write it onto the storage subsystem

We will focus only on this workflow: while the Data Transfer Node is tuned to perform at its best for this workflow, it may perform poorly for other workflows: the DTN is a dedicated host.



The size of science data is huge. But, depending on the type of science, huge may mean terrabytes, petabytes, or more. Some Data Transfer Nodes may have to be deployed as a slive of a datacenter or supercomputer for the very large datasets. Those super DTN's are outside the scope of this tutorial: we will focus on DTN's that can scale up to a dozen of terrabyte: scaling up means adding more servers, not increasing the capacity of it.

Typically, a 6TB system, with a 20G network capability costs about $10,000.



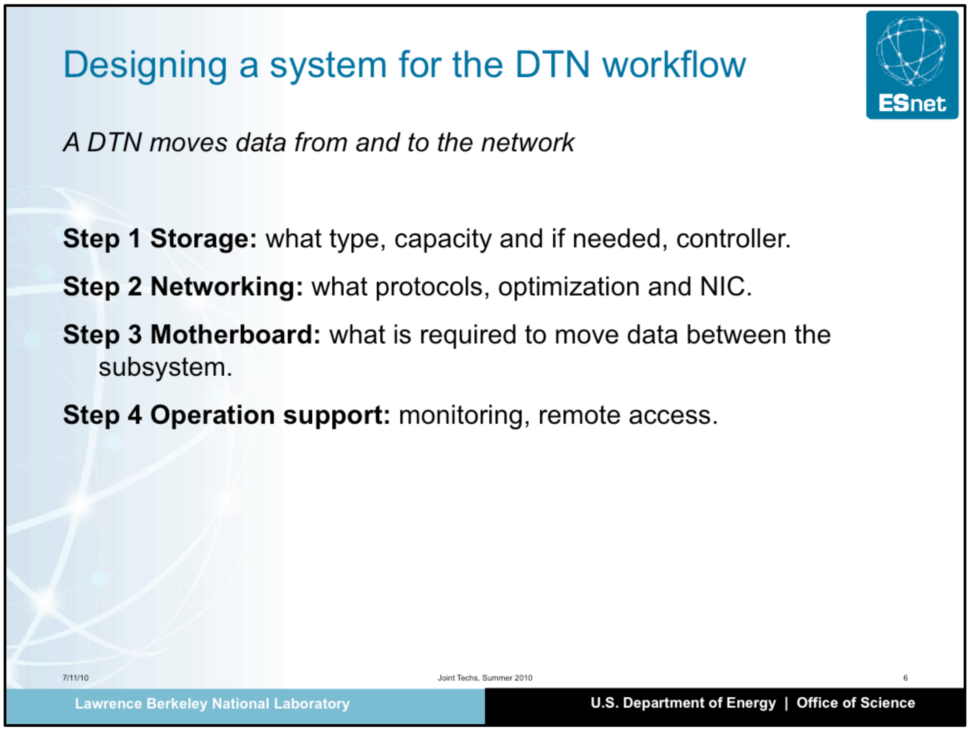
Very few vendor sale high end servers that are capable of being a DTN right out of the box, and those servers are very expensive.

A more typical way is to “design” the server, by selecting all the elements to put together (motherboard, cpu, raid controller, NIC’s…).

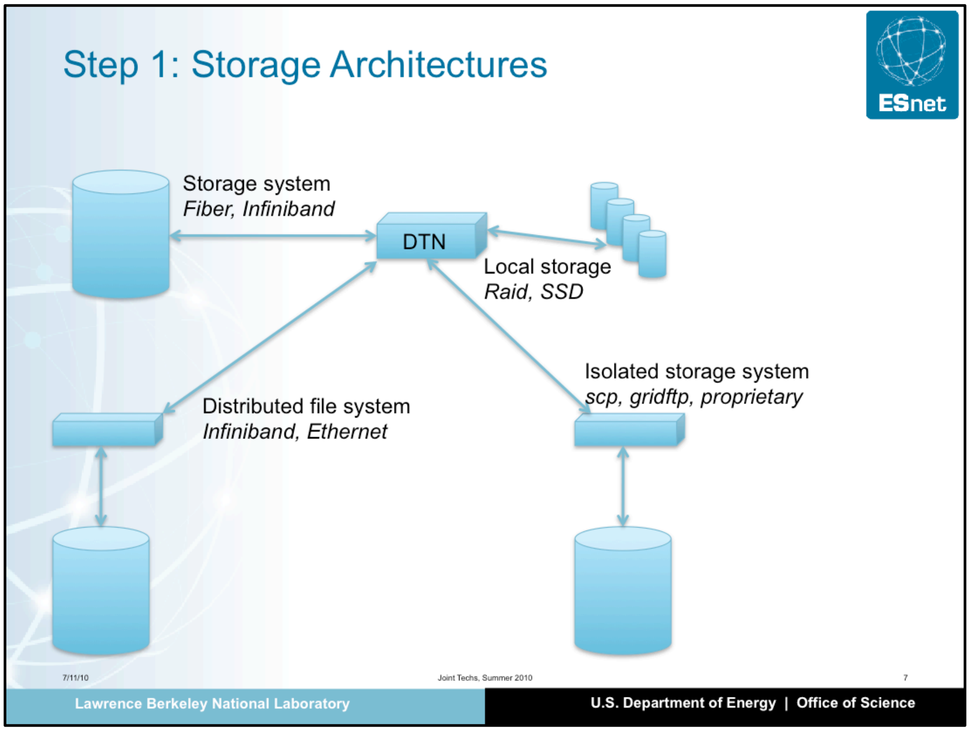
This allows to build exactly what is needed: you can get what you need for less.

However, custom design means that there is little support, especially if the system does not work as expected. Lot of time and effort will have to be spent to design the first server.

When designing a DTN, it is also important to keep in mind that it will deployed. Remote access, power, cooling and maintenance needs to be thought about early on: the server, eventually, may have to be vetted before being racked.



In a nutshell, the motherboards moves data from/to the storage to/from the network. It is critical that this can be accomplished efficiently.

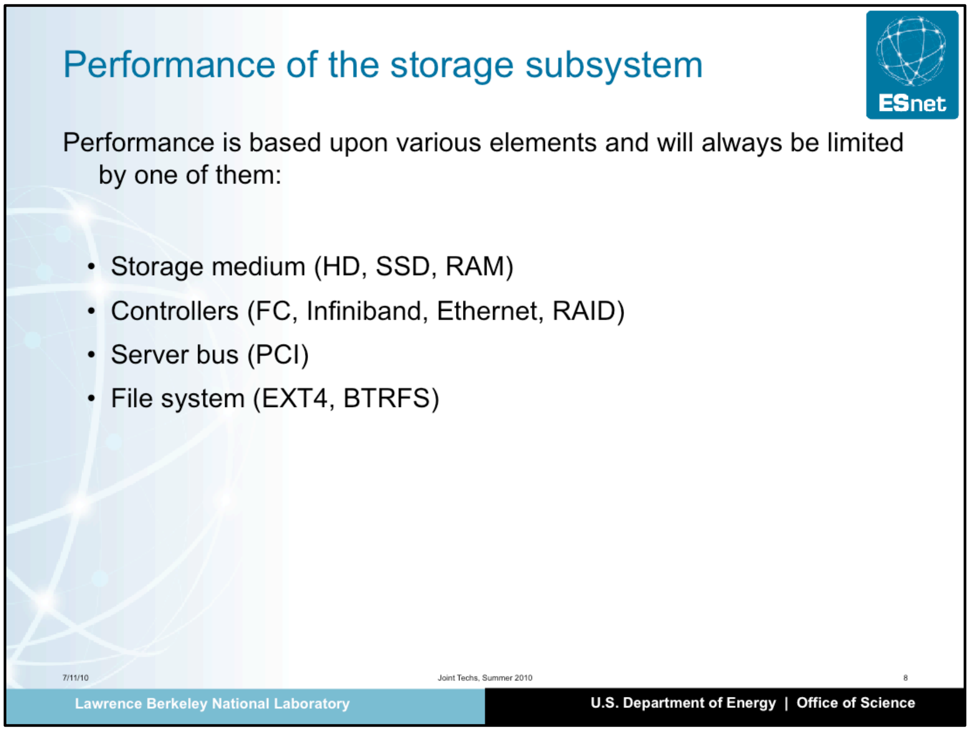


Depending on the deployment environment, a DTN may have different type of storages.

Storage Systems: Those are usually more massive systems (EMC, Netapp, DDN, etc) providing raw volumes to servers. The connectivity is usually fiber, but it can also be infiniband and even ethernet. Typically this architecture benefits storage capacity.
However, it requires, usually, a dedicated HCA and sometimes, a special software stack (OFED)

Dsitributed FileSystem: this is similar to the storage system, except that the exported volumes are not RAW but file system (PNFS, Lustre, GPFS…). This set up is typical of a tiered system: data is acquired and processed, and stored. Then the DTN read from the shared storage.

Local storage: the storage system (just a bunch of drives / JBOD) is packaged with the server. That can from 12 drivers up to 48 drives depending on vendor/model. In addition, external chassis with more drives can be added, connected to the server with SAS or FC. This is ideal for standalone servers since it does not require plumbing for the storage subsystem. However, maintenance is typically more difficult since it does not have all the tooling that usually comes with storage systems.



Performance of a storage subsystem varies depending on its type and architecture.

Type:
Hard drives are cheapwith high capacity. However, their performance is low. Good drives, on average, can do 130MB/sec read or write.
SSD are expensive and have low capacity but they are fast.

Architecture
RAID controller (disk controllers) can be a bottleneck. Some controllers are optimized.

File System: using a file system typically introduces an overhead in the I/O performance. Bad file system such ext3 may use up to 40% overhead. Good file systems (EXT4, BTRFS, ZFS) can almost reach bare metal performance.

Storage Systems

Pros

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

Cons

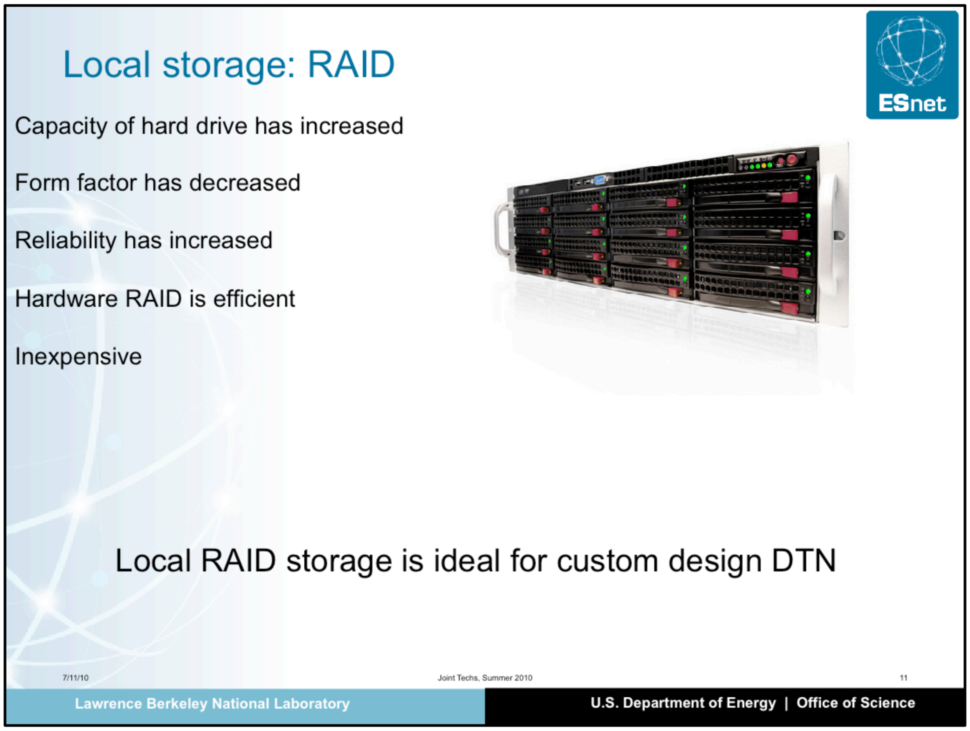
- Controller Bottleneck
- Expensive
- Large footprint

Networked Storages

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

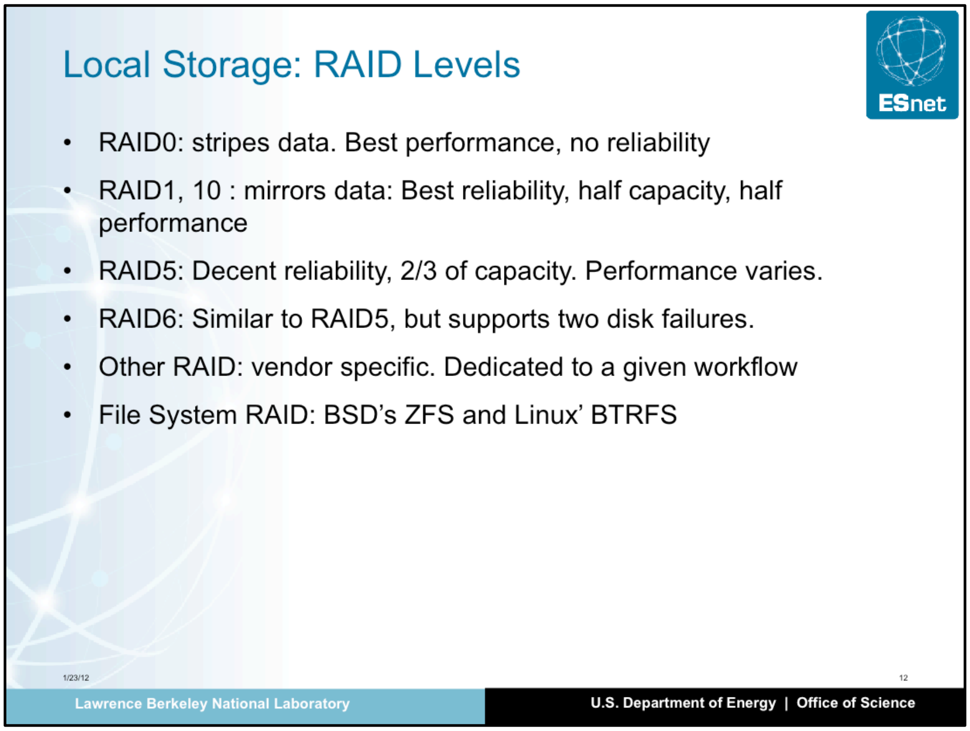
- Flexible storage scalability
- Virtualization
- Especially with SRP, high controller performance

- Requires specific controller
- Requires specific software stack



RAID controllers are capable of high performance while offloading the CPU with the disk operations.

Of course the choice of controller matters. Look at reviews and experiment with loaners when possible.



RAID0 is the right choice when try to get the maximum storage performance at the lowest cost (the lowest number of drives). A single drive failure will cause the entire volume to be lost.

RAID10 is the best choice for reliability (a single disk failure is fully recoverable) but is twice as expensive (it needs twice as many disks as RAID0)

RAID 5,6 and other specialty levels: those levels are compromises between performance, reliability and cost. Often, those are the right choices but quality and power of the RAID controller impacts more performance. In other words, a decent but not exceptional RAID controller may perform very well in RAID0 and poorly in RAID5. Performance RAID5.6 do exist, however, but are typically in the $2,000 price range while good RAID controller (good at RAID0) typically cost less than $1,000.

Note that some file systems, namely ZFS and BTRFS implements RAID in software and are good at it. If the server is powerful (enough cores, at least 16), those file system may perform better than a RAID controller. But they will use much more CPU on the server.

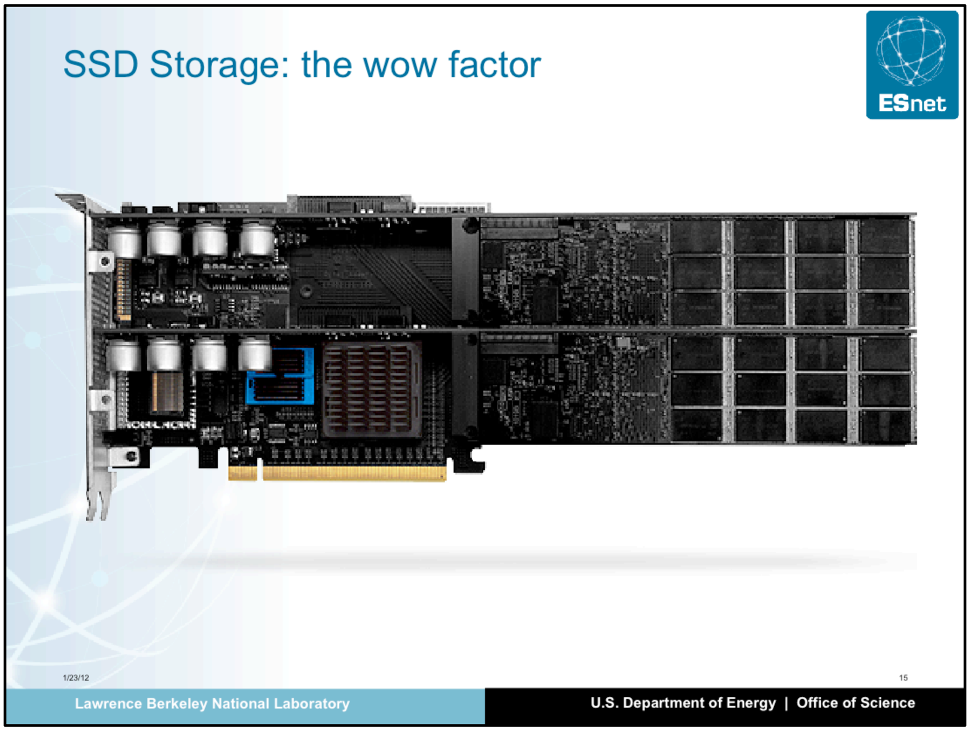
RAID: Great (cheap) but Experiment First

- 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.



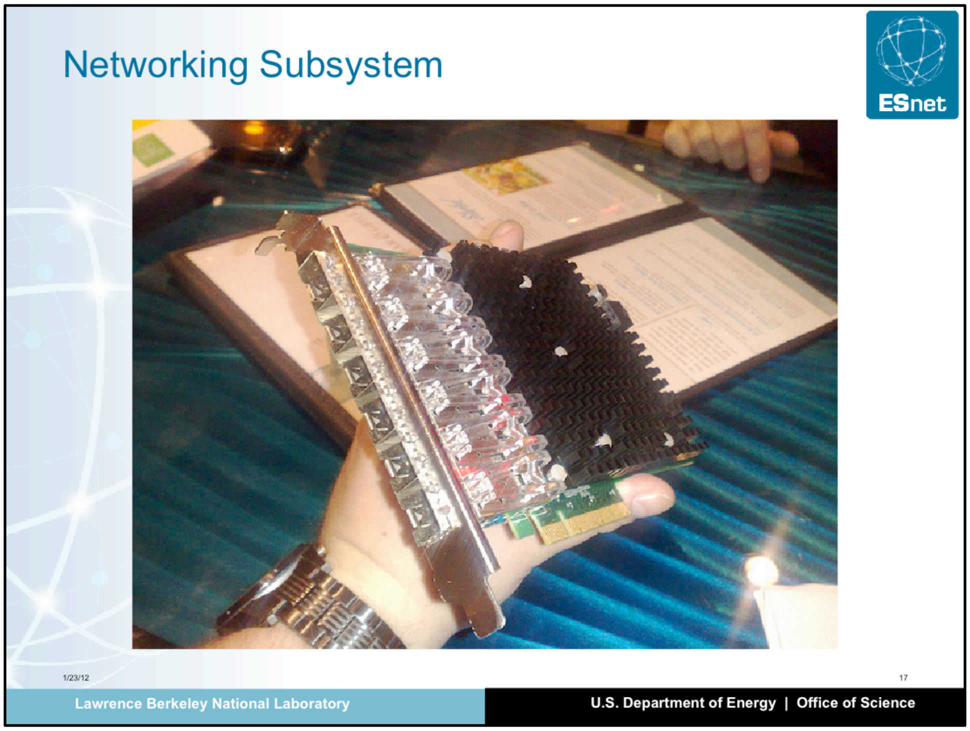
SSD, cost much more than HD, but a much faster. They come in different packages:

-PCIe card: some vendors (Fusion I/O) build PCI cards with SSD. The current maximum capacity is 1TB per card. Since those cards are PCIe, the data transfer between the main memory and the SSD is just limited by the SSD speed and the PCIe speed: in other words, it is really fast (several GB/sec per card). The drawbacks are that 1TB uses a PCIe slot. This design often means that a PCI extender is needed, but if space and performance is an issue, this is the best solution. Those SSD cards can also be an deployment issue: replacing a failed card means that the server must be open.

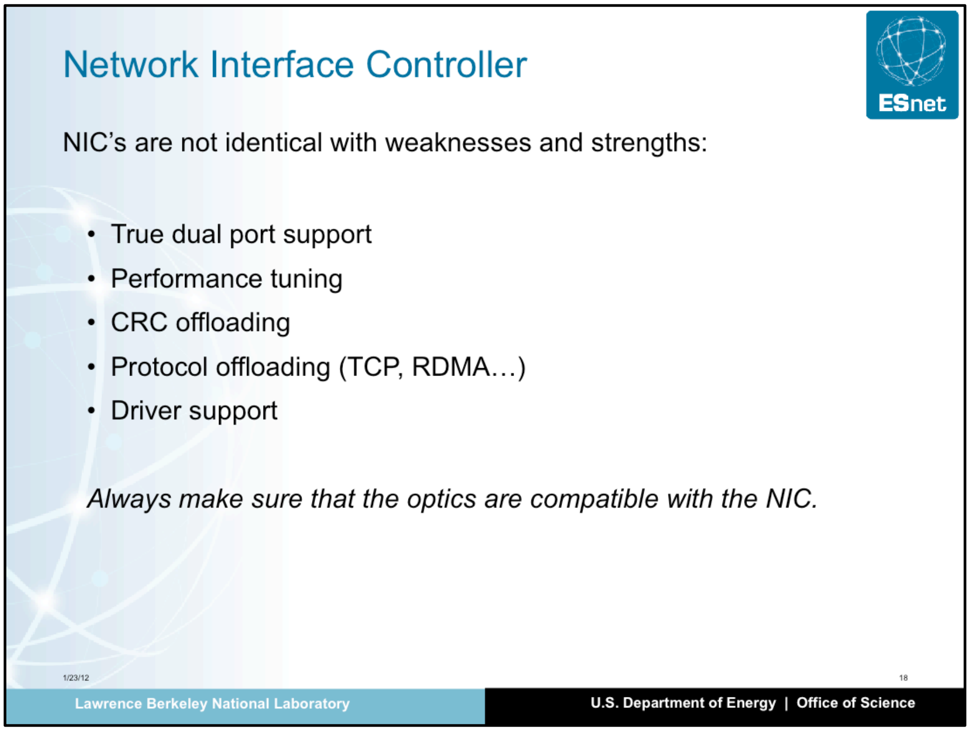
-- HD replacement: some vendors (IBM, WD, etc) have product that a physical replacement for HD: same form factor, same connectivity (SAS, SATA…). Thi allows for easier migration path from HD to SSD, but the performance is limited by the controller. Also, not all controllers are good at controlling SSD drives: always make sure that the controller is “SSD capable”.

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



The networking subsystem is the second subsystem after the storage that is critical and will be a bottle neck. The choice of NIC will impact performance.



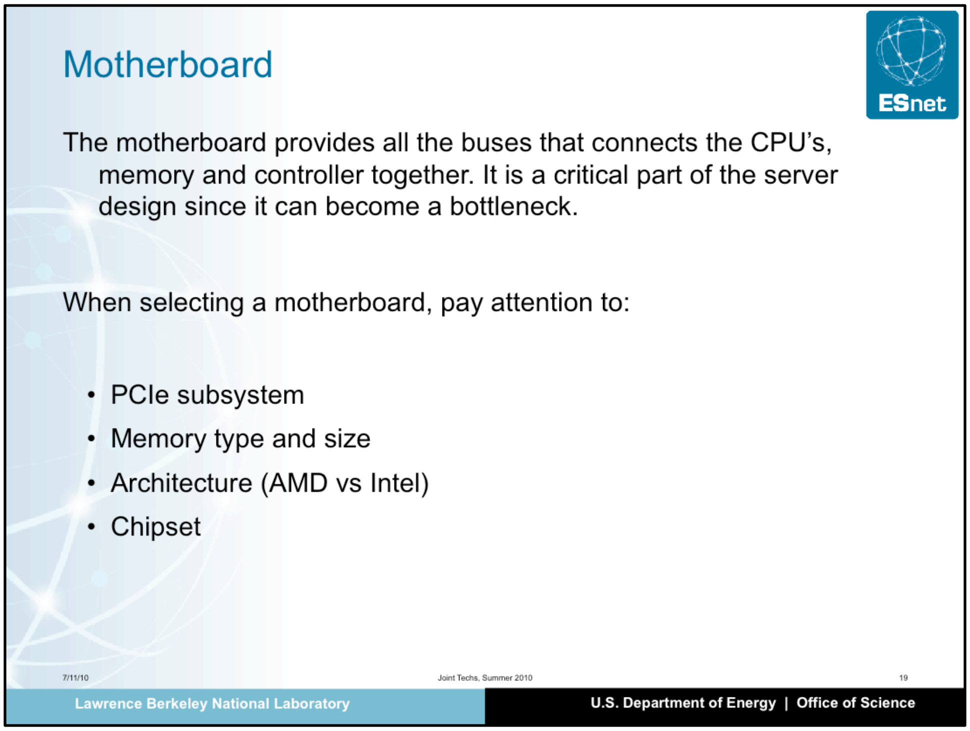
NIC vendors seems to specialize in a given market:

Myricom: some of best performance per port, simple controller. Limited support for exotic protocols.

Chelsio: very large support for protocols (iwarp), and protocol offloading.

Intel: robust driver, supports some third party

Mellanox: somewhat a new player in Ethernet. Converges Infiniband and Ethernet. Excellent support for OFED. Support Layer 2 RDMA (RoCE)



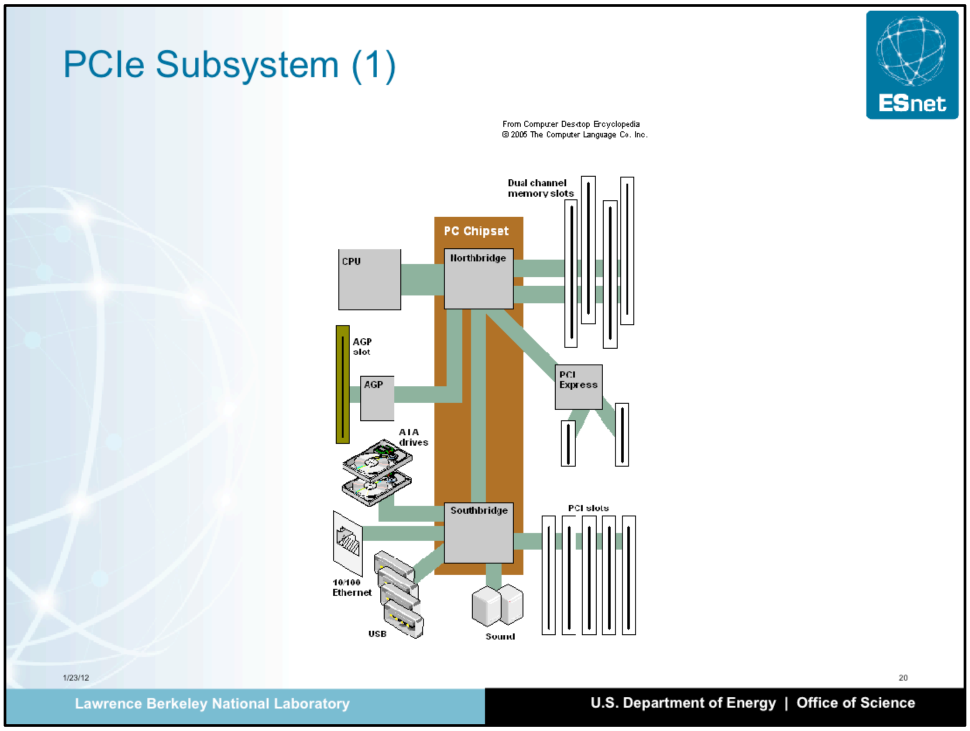
The motherboard not only incorporates the CPU's and memory, it is also providing all the busses between the various component of the server. An inadequate motherboard can become a major bottleneck. It is, then, very important to correctly select it. The questions to ask while selecting are:

How many PCI cards will I need ? How many lanes each ?

What is the aggregate throughput I need on my PCI cards ?

How many cores do I need ? At what speed ?

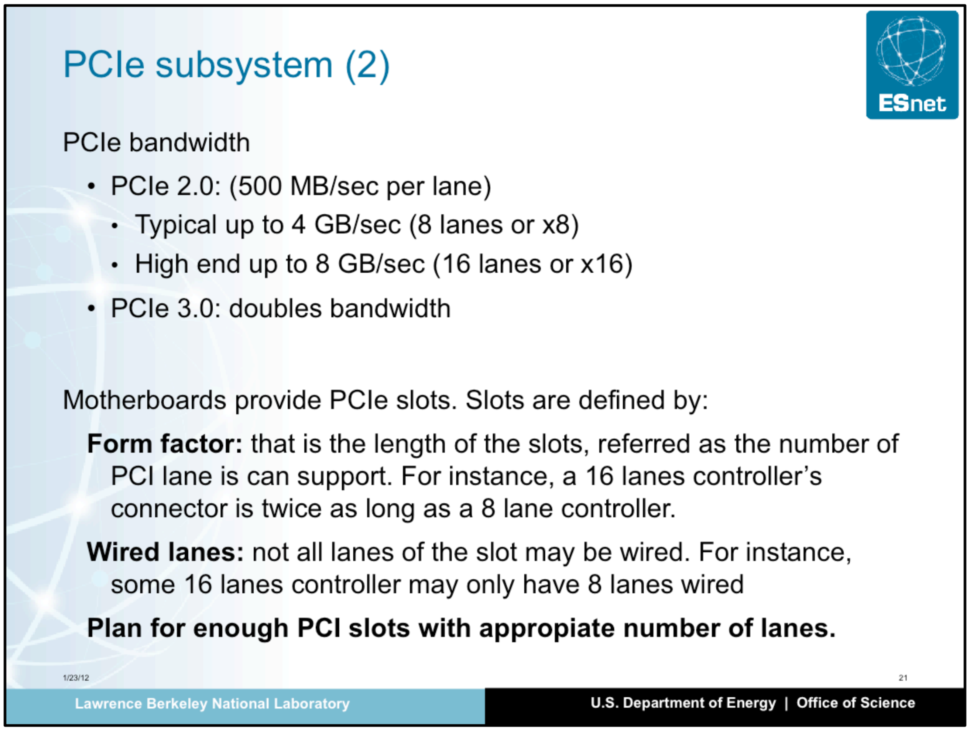
What kind of remote access (maintenance) do I need.



The Chipset is the component in the server that handles all the I/O. In other words, it is responsible for moving data from the PCI cards and the CPU.

Some chipset are better than others (read review), but the most important part of the chipset is the maximum number of PCI lanes it can handle.

Also, depending on how the chipset and the PCI bus is wired, the architecture may or may not fit your needs. It is then important to look at the schematic of the motherboard to see if the I/O subsystem can provide the required performance.



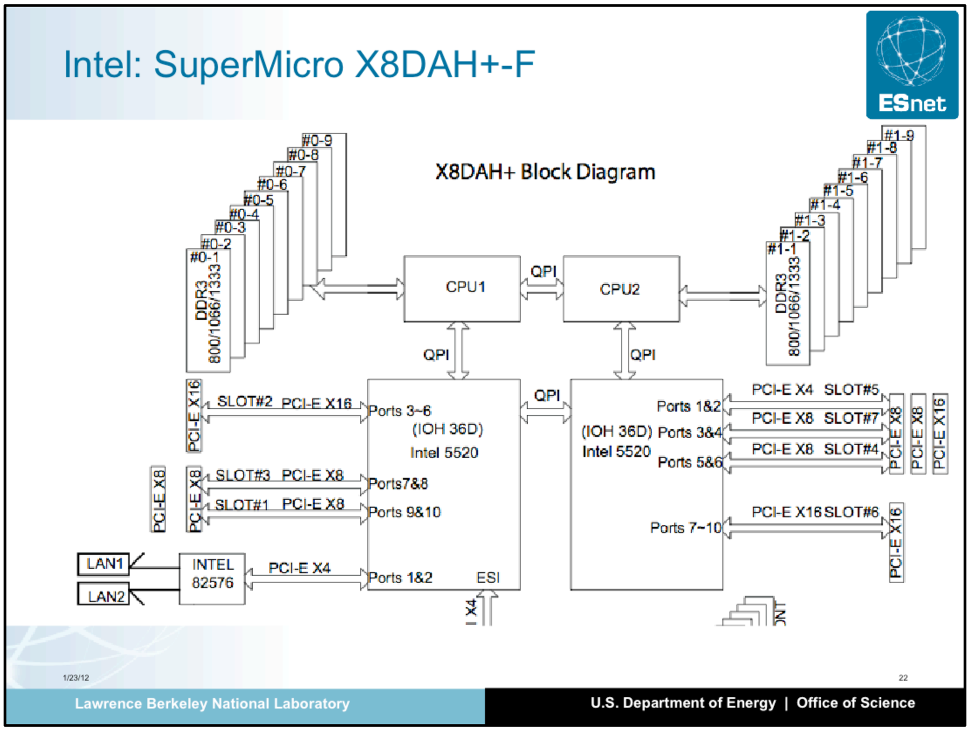
In order for the overall system to performance to the specifications, it is critical to set the card into the appropriate slot:

The slot must have wired the correct number of lanes. With a PCIe Gen2 system, most cards are x8 (8 lanes). Some cards such as a 6 x 10GE port or a SSD Fusion I/O card are 16 lanes. Be careful when selecting a PCI slot:

some motherboards have slot that look like x8 or x16, but a fewer number of lanes are really wired. Typically the board will say something like “x4 in a x8 slot”

If you are running out of slots, there are products that adds an external chassis with just an array of PCI slots. Those are named “PCI extender”

PCIe Gen3 is coming ! This will multiply by 2 the PCI throughput. While this is very exiting (DTN do need PCIe Gen3), wait until the second generation of Gen3 motherboards come out: you do not want to hit bios bugs or hardware bugs. But again, Gen3 PCI is much needed considering the data size and the modern WAN capability (100G fiber)

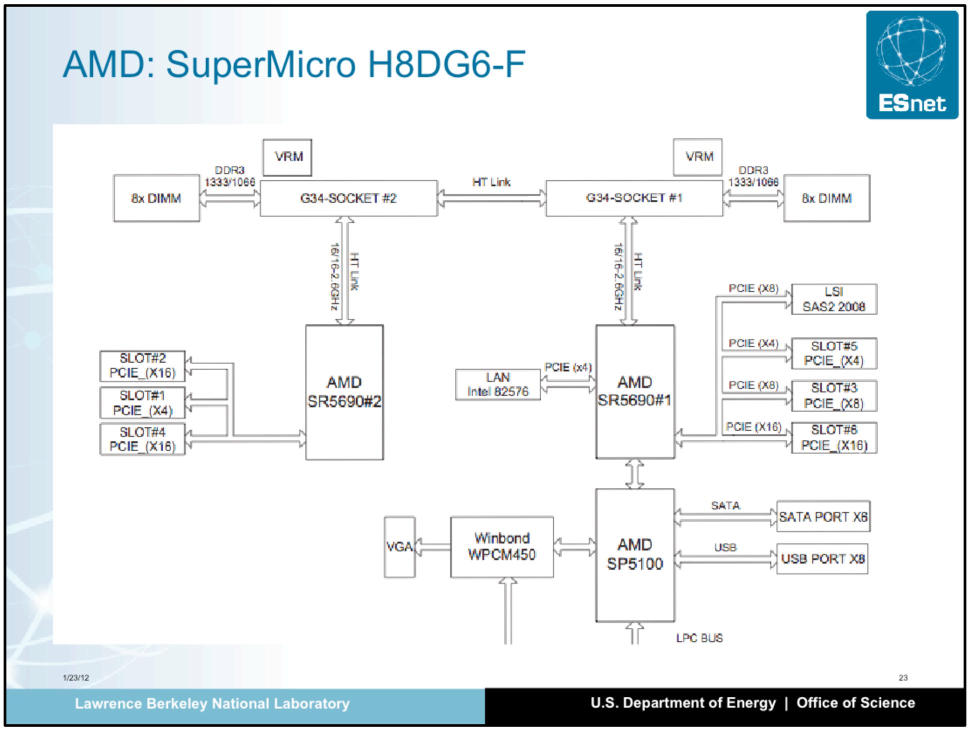


Notice the two independent I/O path:

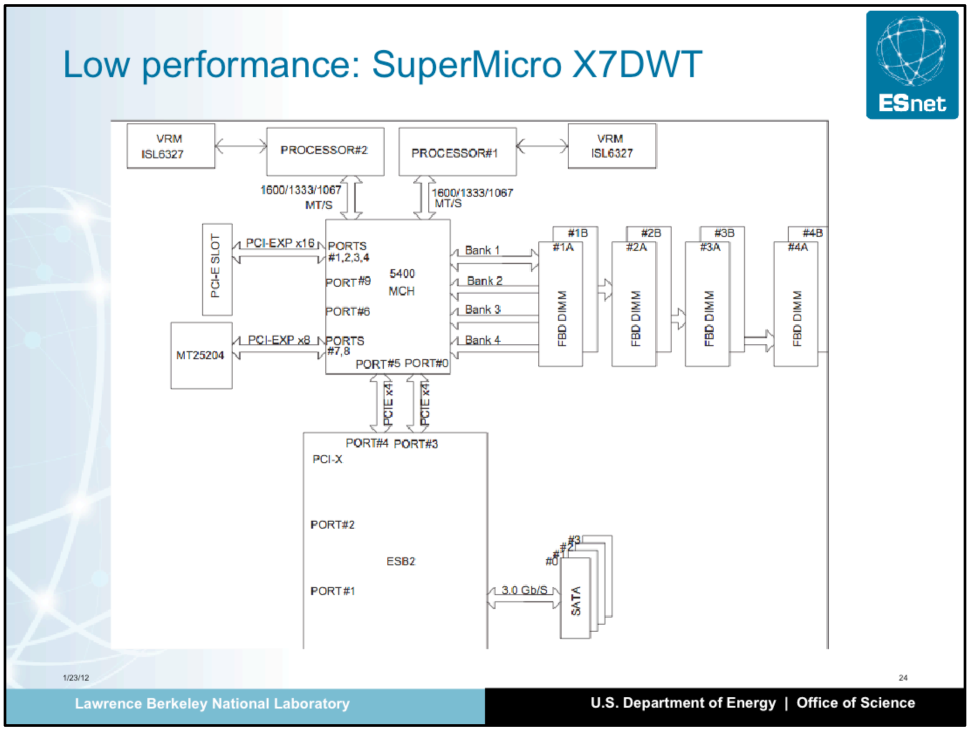
Memory <-> CPU <-> Chipset <-> PCI card

This architecture is good because it allows two split the I/O and networking cleaning without congestion point.

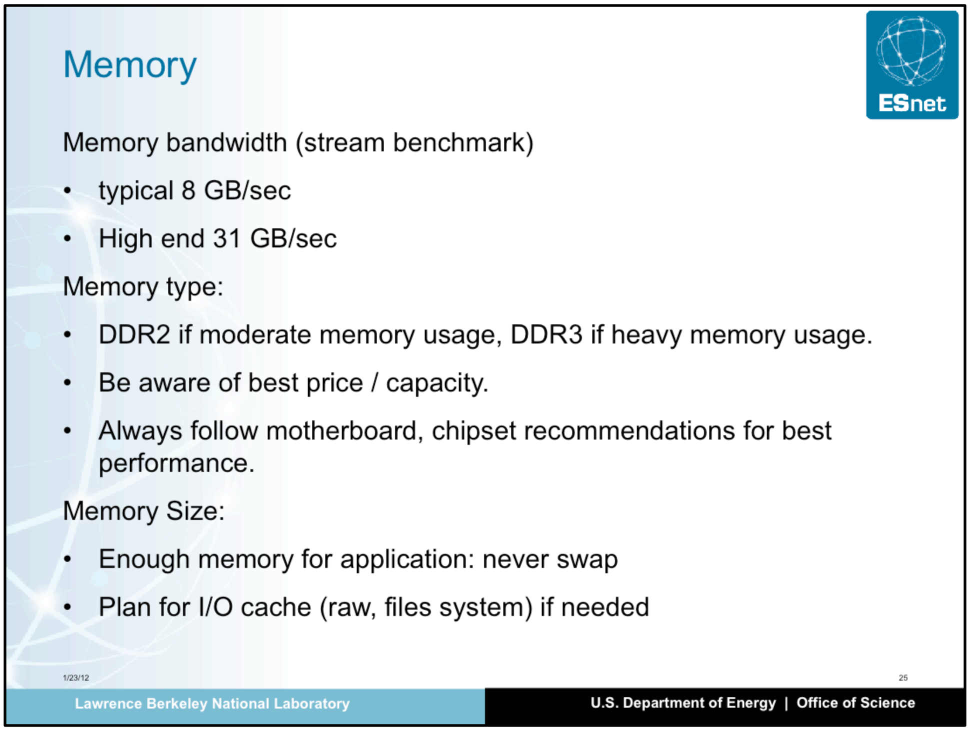
Note the number of lanes of each of the PCI slots



Similar architecture than the previous board but for Intel



This is not a bad motherboard, just not designed for performance. It has only one chipset (but still two processors). It also has a single memory bank.



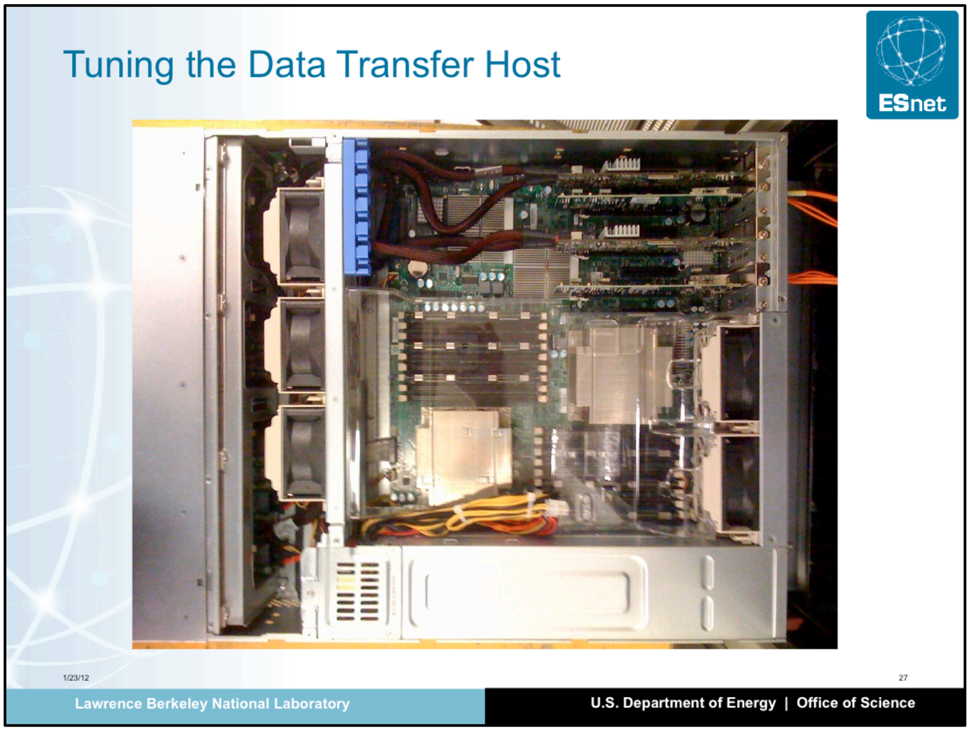
Remember that memory is used for several functions:

1) Application
2) I/O Write/Read cache (the more memory for the cache, the better the system will handle performance spikes. A good DTN would typically have 10G of write cache
3) Network buffers.

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)



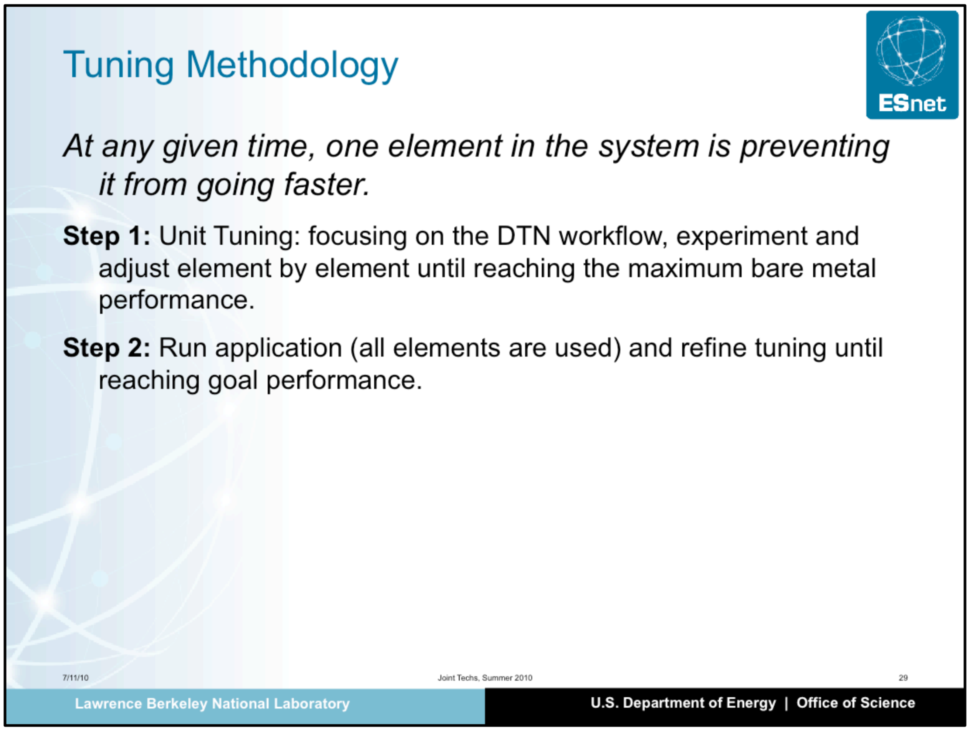
At this point the DTN is designed and assembled to your specification. The next step is to configure, tune the entire system, so it performs as expected. If the DTN is correctly designed, in other words, the hardware is capable of delivering the required performance, with patience and methodology,

Tuning

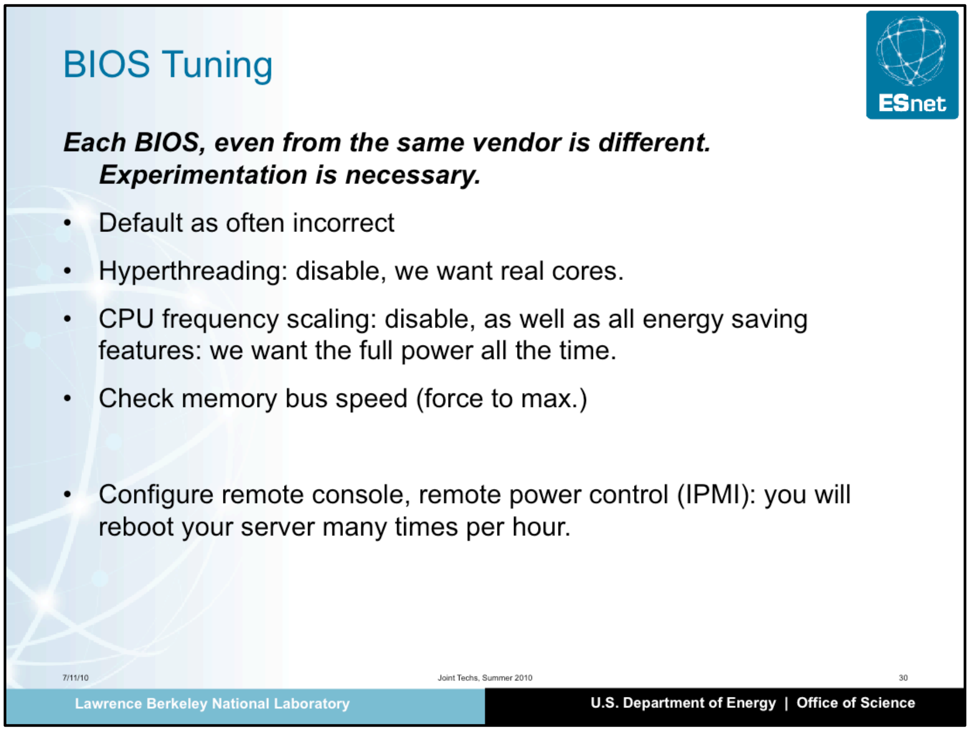
Defaults are usually not appropriate for performance.

What needs to be tuned:

- BIOS
- Firmware
- Device Drivers
- Networking
- File System
- Application

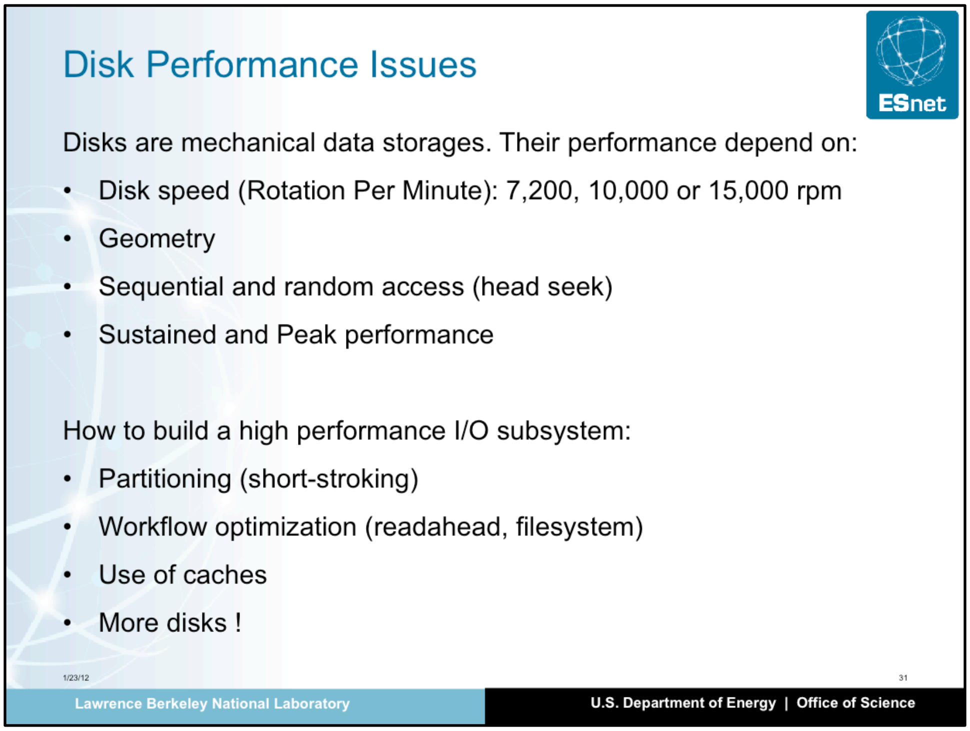


A lot of time can be spent tuning a system and it is easy to not make progress. It helps to use a methodology which is based working on one element of the system at the time, gathering and recording measurements.



BIOS tuning can be painful. A wrong setting can have dramatic effect on performance, but also on stability of the system. The goal is to make the behavior of the hardware as predictable as possible and to run it at maximum performance.

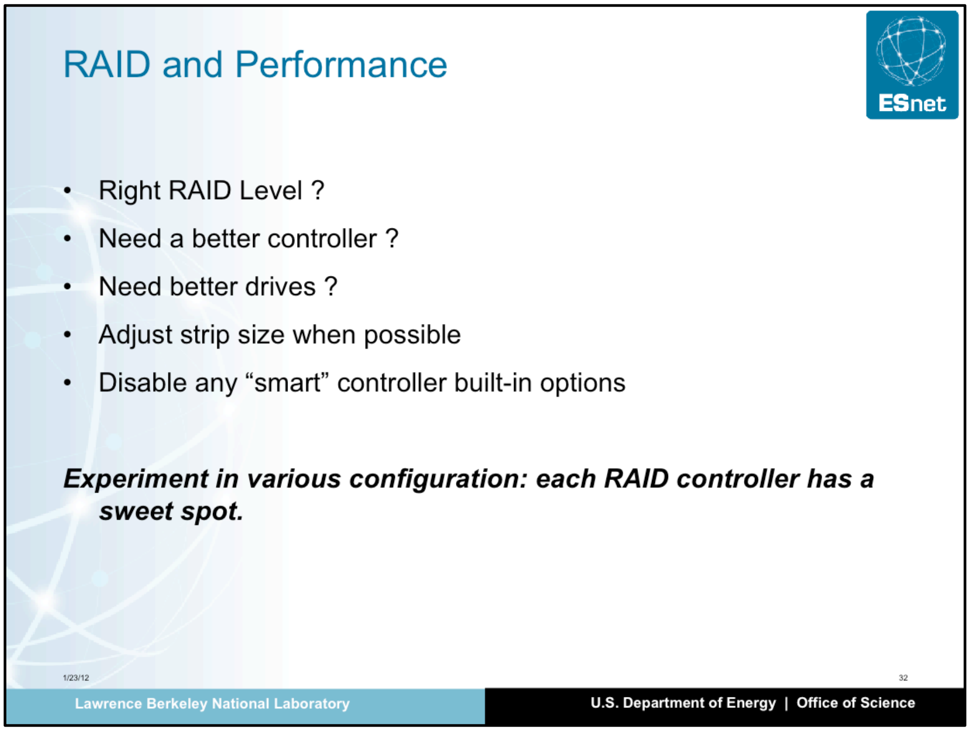
Before changing a BIOS setting, always note what it the current state: you may need to return to a previous state of the BIOS if you make an error.



Designing a RAID system is almost an art: there are so many constraints that while it is almost always possible to optimize the storage subsystem, it is almost always impossible to get what you really want. When working on the storage subsystem, ask yourself the following questions:

- how many files do I need to send or receive at the same time
- based on the maximum network throughput, how fast a file must be read or written ?
- how large is the average file ?
- how reliable the storage must really be ?
- do the files compress well ?
- how will you answer to those same questions in one year, two yers, four years ?

Fortunately optimizing storage is perhaps one of the performance work that is the most publically documented (blogs, storage vendors…). A rule of thumbs is when using hard drives, you should get at least about 130MB/sec per disk..



Disk controllers, RAID or not, are usually designed for “entreprise”. This usually means that the controller is often configured with RAID 5 or 6. As a consequence, controller are most of the times, not capable of running all the drives at full speed: in entreprise context, there are always a few drives that are hot replacement. A rule of thumb is that if a controller is said to handle up to X drives, it can handle up to 2X/3 drivers at full speed.

Some high end controller (Areca for instance) are specifically designed for a workflow similar to a DTN workflow: sequential read/write. They may have Gigabytes of SRAM for internal buffering, PCIe x16..

Finally, some RAID controller are specifically designed to scale up. In addition to wire internal drives, they can control external drives, directly or in a daisy chain manner

Tool: vmstat

From man page:

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

- Shown 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
```

34

1/23/12

Lawrence Berkeley National Laboratory

U.S. Department of Energy | Office of Science

Example of a “dd test”

```
# 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     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     0 1296932 26799 3373  0 22 52 26  0
 3  6      0  88860 224184 23475516    0    0     0 1322248 28338 3529  0 22 51 27  0
```

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)

  kernel /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
  http://www.kernel.org/doc/ols/2004/ols2004v2-pages-105-116.pdf

```
/sbin/blockdev --setra 262144 /dev/sdb
/sbin/blockdev --setra 262144 /dev/sdc
/sbin/blockdev --setra 262144 /dev/sdd
```

File Systems Performance

ESnet

- 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 requires 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:

```
net.ipv4.tcp_no_metrics_save = 1
```

Also should change this for 10GE

```
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).

Useful Links