Supercomputers and Job Scheduling

Lecture 19

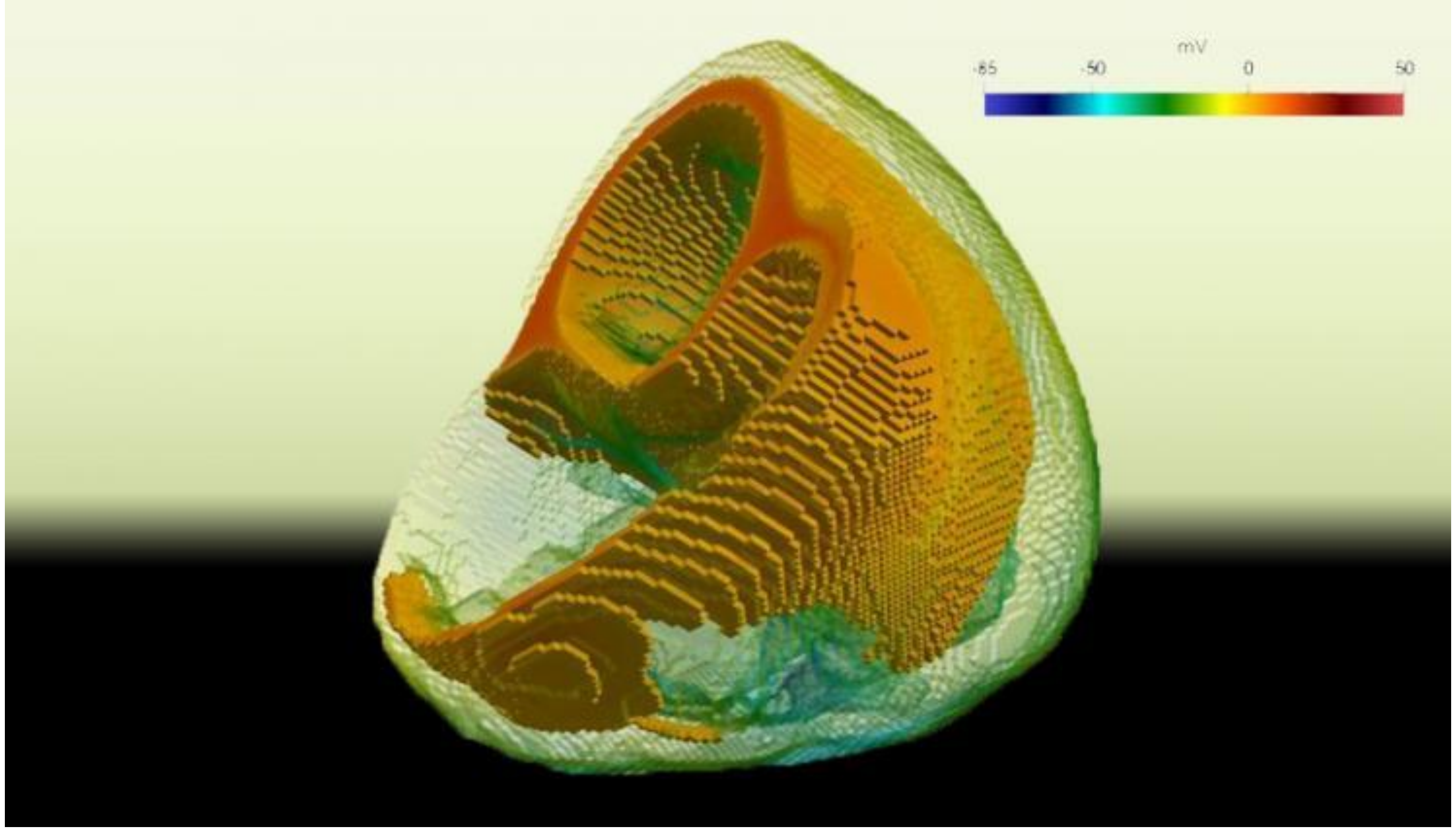
Apr 9, 2025

IBM Blue Gene/Q

- November 2011
  - 4,096-node BG/Q (Sequoia)
  - #17 on top500 at 677.10 TF
  - #1 Graph 500 at 254 Gteps (Giga traversed edges/second)
  - #1 on Green 500 list at 2.0 Gflops/W
- June 2012
  - #1 Sequoia at Lawrence Livermore National Laboratory (#13 in 2019)
    - 96K nodes, 16.3 PF Max, 20 PF Peak, 7.8 MW
  - #3 Mira at Argonne National Laboratory (#24 in 2019)
    - 48K nodes, 8.1 PF Max, 10 PF Peak, 3.9 MW
    - Decommissioned in Dec 2019

Real Applications on Sequoia

Cosmology code HACC 14 PFLOPS



Heart simulation code Cardioid 12 PFLOPS

World's Top Supercomputer Simulates the Human Heart

Sequoia, the most powerful supercomputer (for now) and a 2012 PM Breakthrough Award winner, was built to model nuclear weapons explosions. But before the machine goes fully classified, scientists used its incredible power to build a simulation of the human heat that looks down to the cellular level and predicts how a heart would respond to particular drugs.

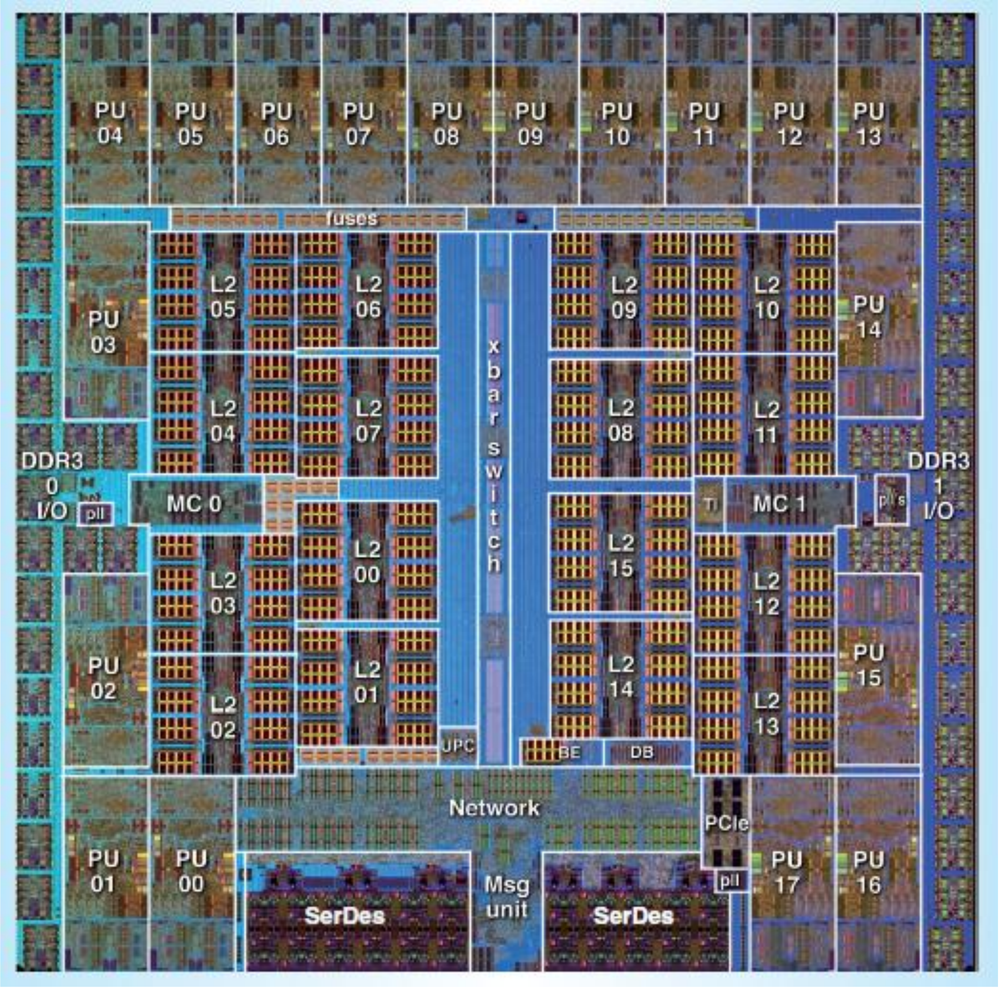
Supercomputer simulates the heart with record accuracy

The simulations divide the heart into thousands of little digital pieces, each composed of mathematical models that take input fron and send data to other pieces. Their interactions require an enormous amount of calculations, even though the model only simulates electrical activity, not physical.

The best that could be done before was to simulate pieces about 0.2mm across, and it could take 45 minutes or so to simulate a single beat. With new software called Cardioid running on the Sequoia supercomputer, not only can they simulate more accurately (the pieces are 0.1mm across, about the size of an actual heart cell), but also hundreds of times faster: now a virtual heartbeat only takes 10 seconds to create.

Credit: https://www.popularmechanics.com/science/health/a8241/worlds-top-supercomputer-simulates-the-human-heart-13989798/

BG/Q Compute Chip



The IBM Blue Gene/Q Compute Chip, IEEE MICRO, 2012

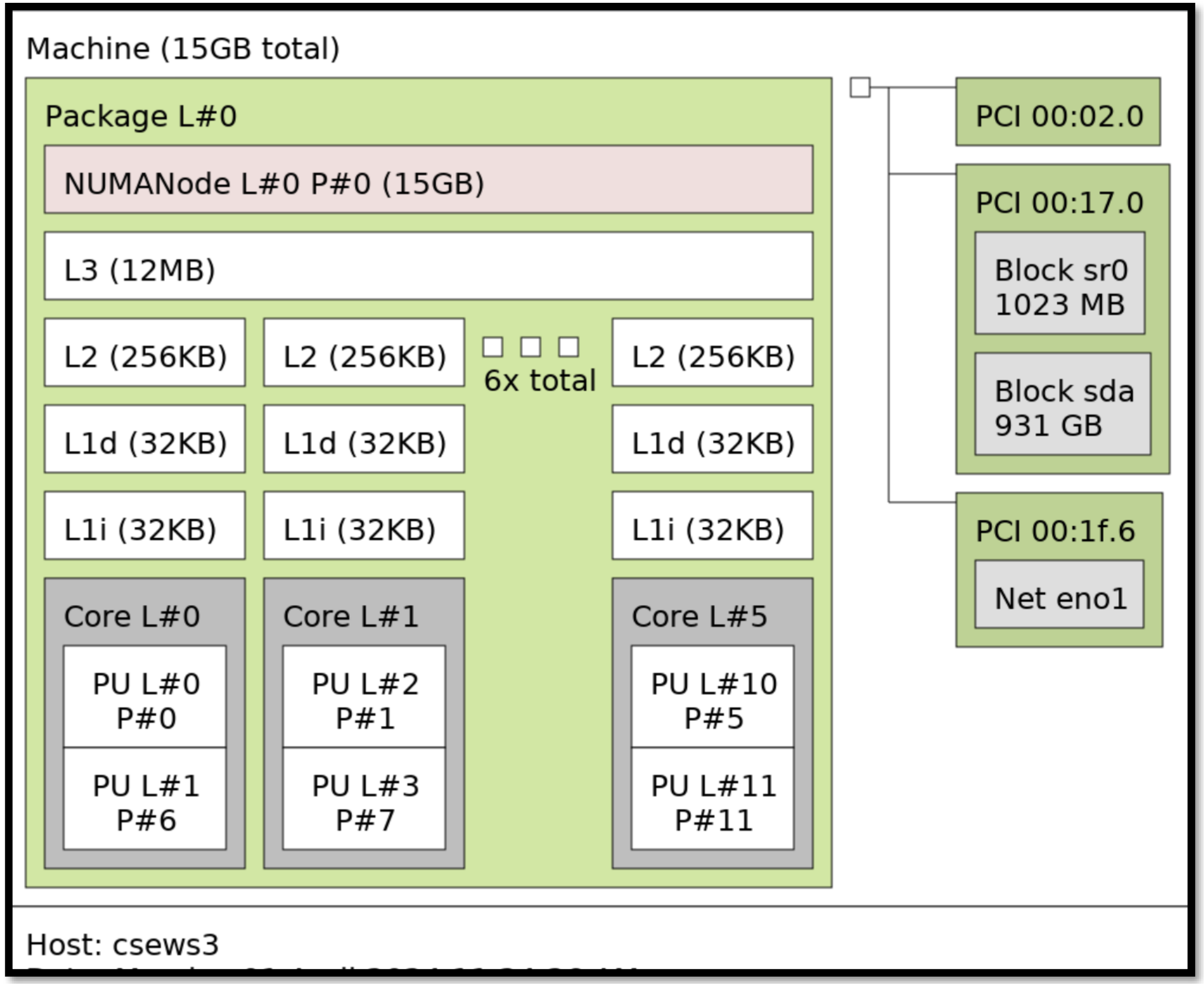
Wikipedia

Supercomputer [edit]

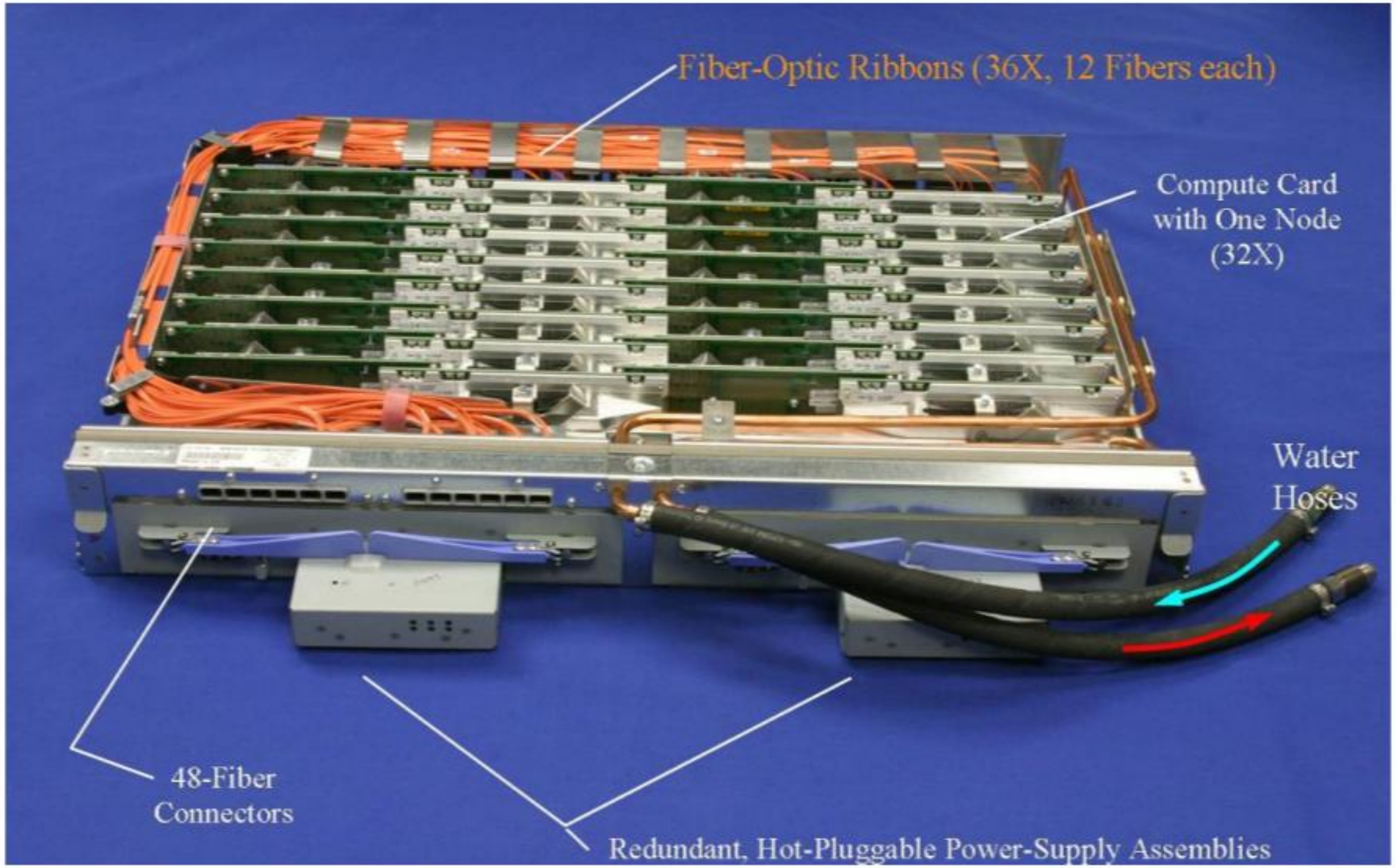
- Blue Gene/L, dual core PowerPC 440, 700 MHz, 2004
- Blue Gene/P, quad core PowerPC 450, 850 MHz, 2007
- Blue Gene/Q, 18 core PowerPC A2, 1.6 GHz, 2011

- 18.96 x 18.96 mm chip (45 nm, 1 billion transistors)
- 16 active cores, memory, cache, NoC
- PowerPC A2 Processor Core
  - 1.6 GHz
  - 64-bit Power ISA
  - In order execution
  - 4-way SMT
  - 2-way concurrent instruction issue
- Quad FPU

Machine Architecture Example (lstopo)



BG/Q Compute Node Board (32 nodes)



Interconnects in BG

- BG/P has a 3D torus with 425 MB/s per link
- BG/Q has a 5D torus with 2 GB/s per link

Why 5D torus?

- Lower diameter, higher bisection width, lower latency than 3D torus
- High nearest neighbour bandwidth

BG/Q Messaging Unit and Network Logic

- A, B, C, D, E dimensions (5D torus)
  - Last dimension E is of size 2 (reduces wiring)
  - Link chips on each node board connect via optics to node boards on other midplanes
  - Dimension-order routing
- On-chip per hop latency: 40 ns (20 network cycles)
  - 16x16x16x12x2 P2P latency is about 2.6 μs
  - 0.6 μs at 1 hop, 1.17 μs at 13 hops
- Injection and reception FIFOs (More than half latency incurred here)
  - Packets arriving on A- receiver are always placed on A- reception FIFO

BG/Q Network Device

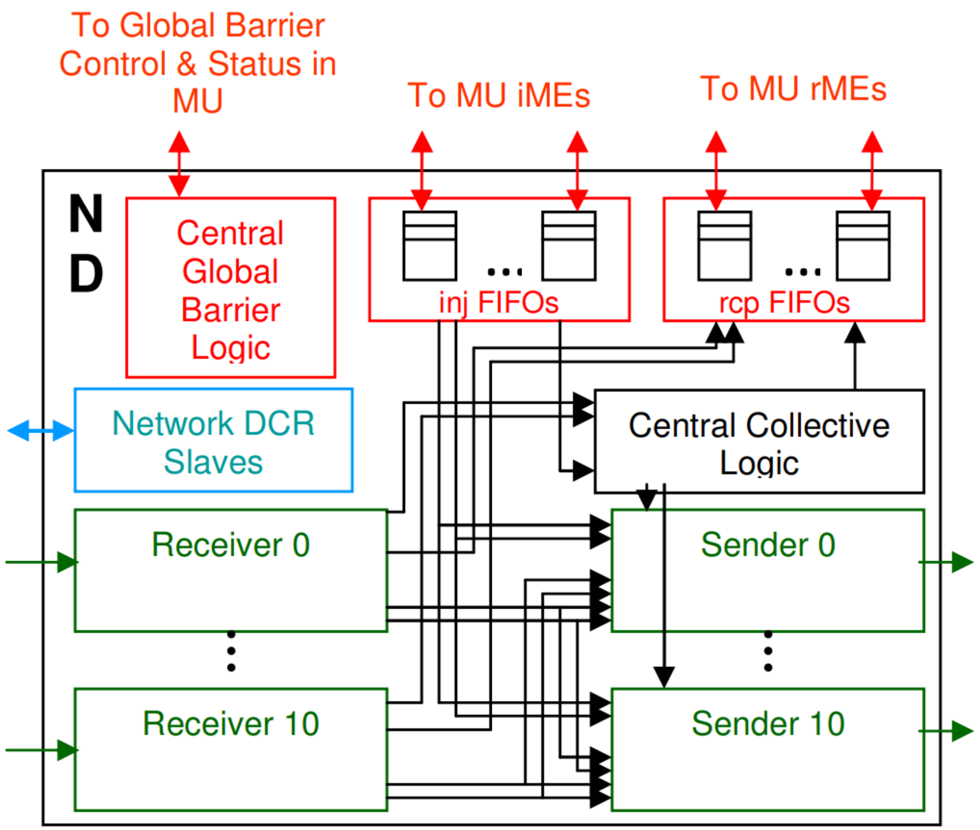
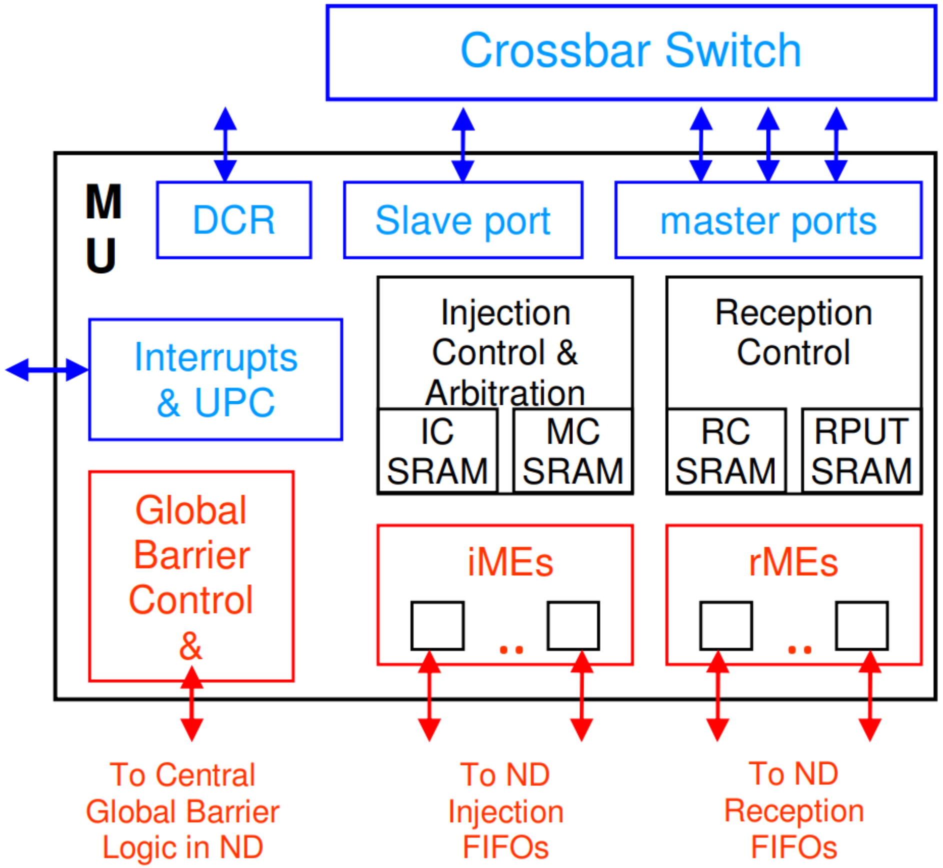


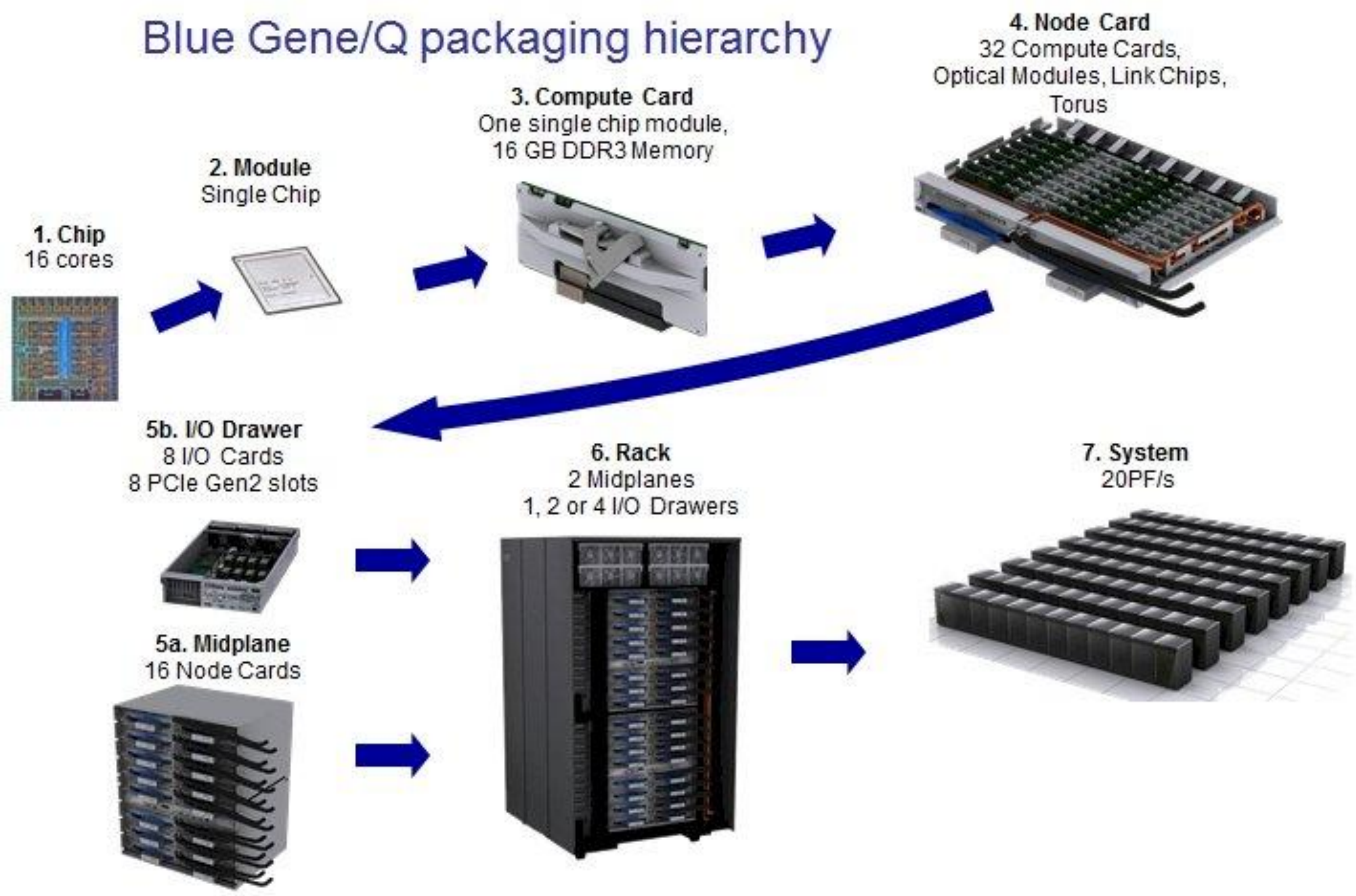
Figure 1. The BG/Q Network Device (ND) Router Logic



Messaging Unit (MU)

BG/Q Hierarchy

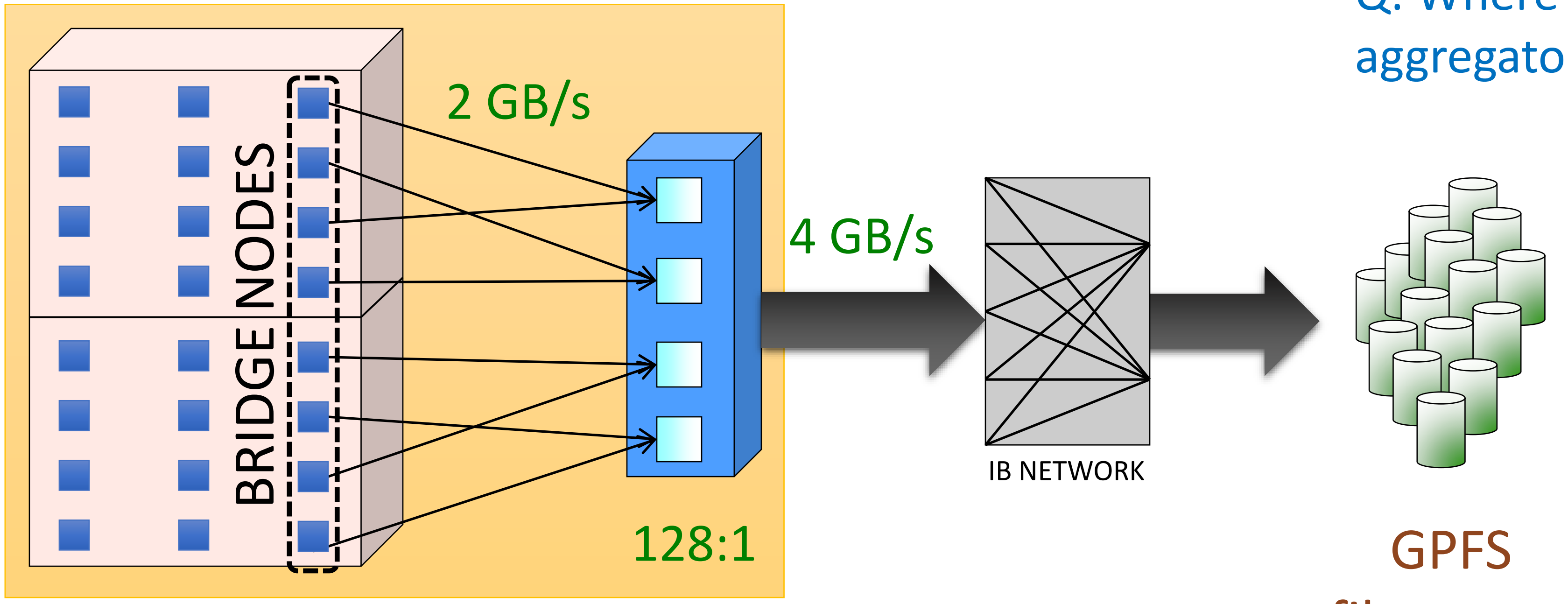
1 Rack (1024 nodes)->
2 Midplanes (512 nodes)->
16 Node boards (32 nodes)



BG/Q – I/O Node Architecture

Q: Where should the aggregators be placed?

512 compute nodes



GPFS filesystem

Compute node rack

1024 compute nodes
16 bridge nodes

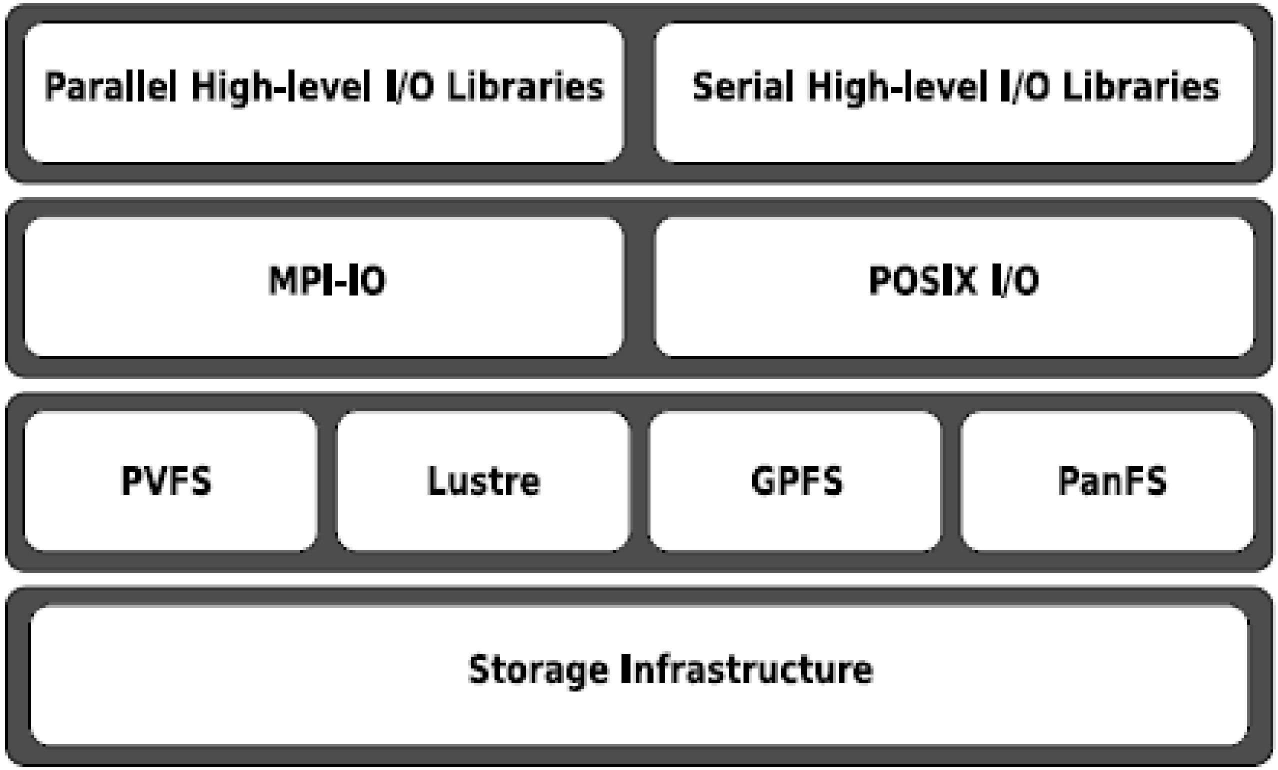
I/O nodes

2 bridge nodes
connect to 1 I/O node

References for BG/Q (Optional Reading)

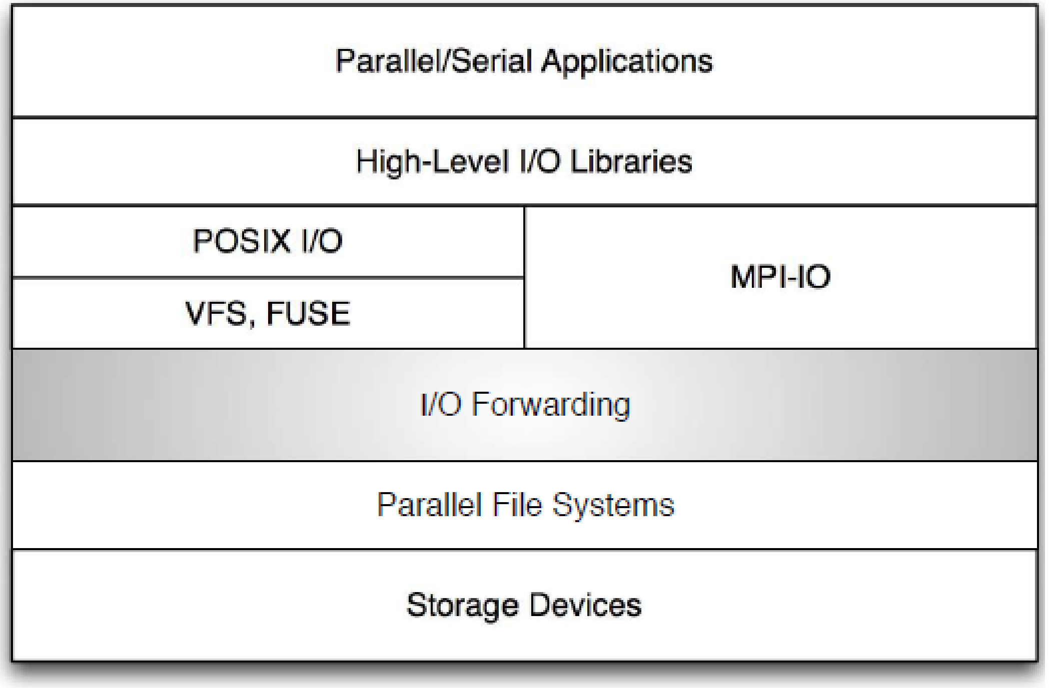
- The IBM Blue Gene/Q Compute Chip, IEEE MICRO, 2012.
- The IBM Blue Gene/Q Interconnection Fabric, IEEE MICRO, 2012.
- The IBM Blue Gene/Q Interconnection Network and Message Unit, SC 2011.
- Looking Under the Hood of the IBM Blue Gene/Q Network, SC 2012.
- IBM System Blue Gene Solution: Blue Gene/Q Application Development, IBM Redbooks, 2013.

I/O Stack



E.g.: NetCDF

I/O Forwarding



Source: Ohta et al., “Optimization Techniques at the I/O Forwarding Layer”

- I/O requests forwarded to dedicated I/O nodes by compute nodes
- I/O nodes redirect I/O requests to the backend parallel file systems
- Reduces the number of clients accessing the file systems
- Can reduce the file system traffic by aggregating and reordering I/O requests
- I/O forwarding scheduler can exploit the global view of parallel applications to sort and merge I/O requests more effectively

High Data Throughput

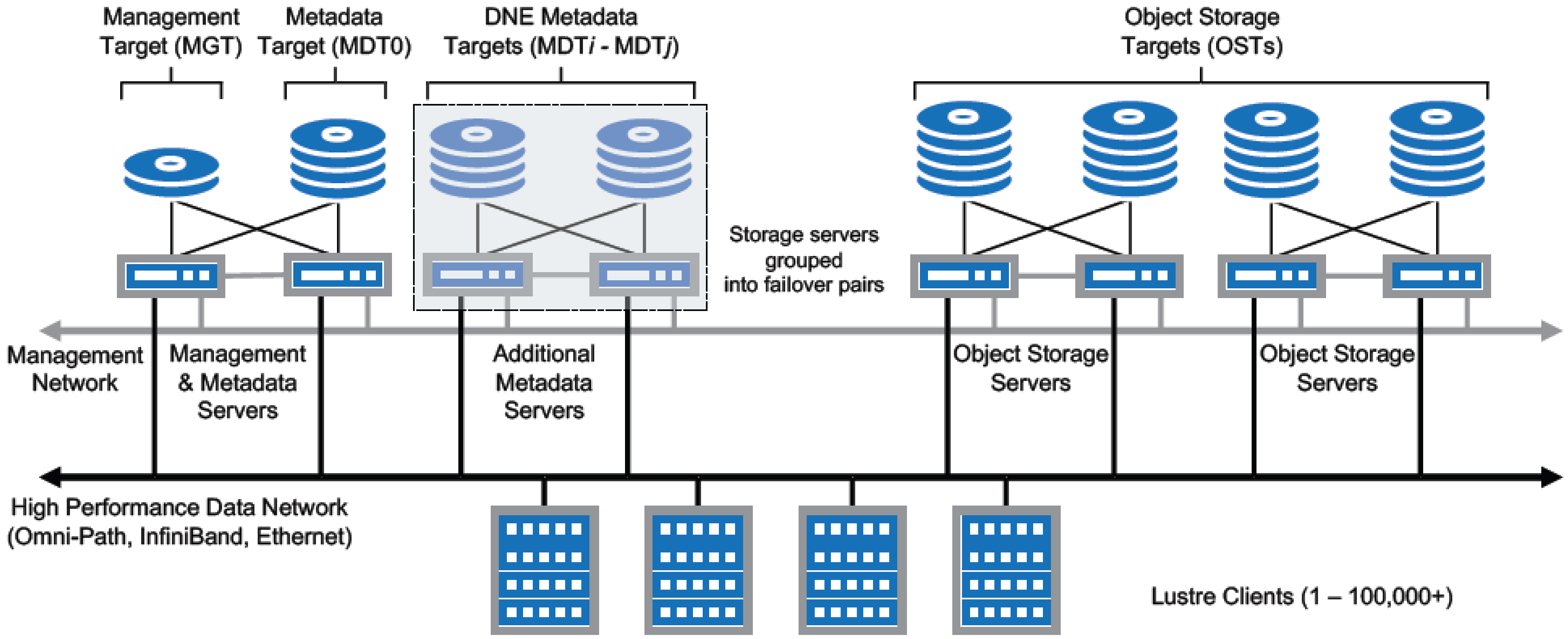
- I/O forwarding from compute to I/O nodes
- Multiple I/O servers to manage the data storage
- A large file may be striped across several disks

Lustre File System

- Parallel file system
- Used in 15/30 top 500 supercomputers
- POSIX-compliant file system
  - presents a unified file system interface to the user
- Object-based filesystem
  - “A storage object is a logical collection of bytes on a storage device” [1]
  - Composed of data, attributes, metadata
  - Files distributed across multiple objects
- Scalability due to object storage and division of labor
- No file server bottleneck

[1] Mesnier et al., Object-Based Storage, IEEE Communications Magazine, 2003

Lustre Scalable Storage Architecture



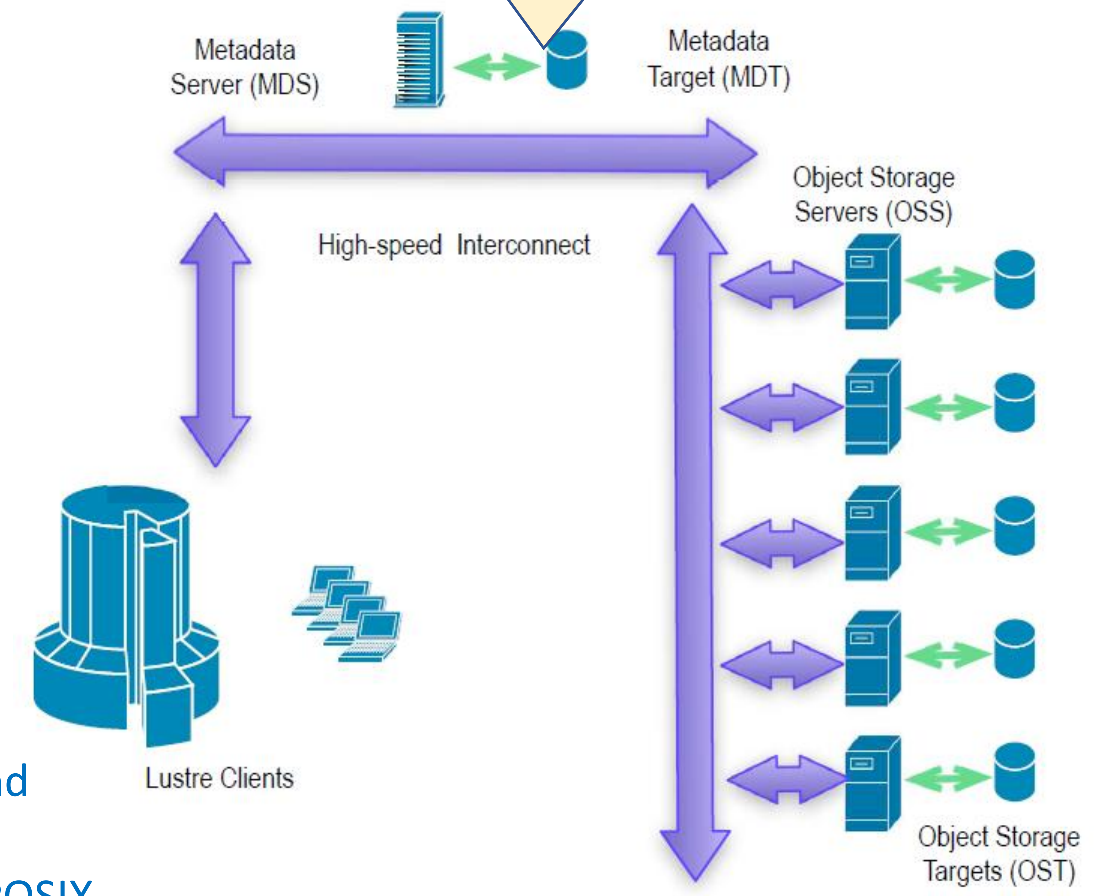
"Lustre can deliver more than a terabyte-per-second of combined throughput." -- http://wiki.lustre.org/images/6/64/LustreArchitecture-v4.pdf

Lustre

Three components

- Metadata servers (MDS)
- Object storage servers (OSS)
  - Object storage targets (OST)
- Clients

Stores file metadata, such as file names, directory structures, and access permissions



Lustre clients access and concurrently use data through the standard POSIX I/O system calls.

Source: Understanding Lustre Internals

Lustre Components

Metadata Server (MDS)

- File operations (create, open, read etc.) require metadata stored on MDS
- Handles metadata requests - file lookups, file and directory attribute manipulation

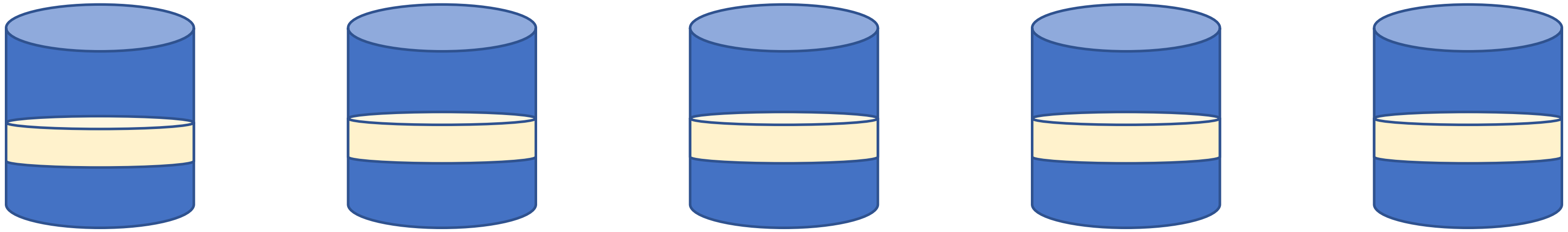
Object Storage Server (OSS) and Object Storage Targets (OST)

- Each file is composed of data objects striped on one or more OSTs
- Responsible for actual file system I/O

Lustre Client

- Queries MDS
- Retrieves the list of OSTs and sends request to the OSTs

Lustre Striping



- Stripe size
- Stripe count/width

Obj 1 => OST A
Obj 2 => OST B
Obj 3 => OST C
Obj 4 => OST D
Obj 5 => OST E

Lustre Striping Example

```
[pmalakar@cn364 testq]$ lfs setstripe -c 10 testmpiio.out
[pmalakar@cn364 testq]$ lfs getstripe testmpiio.out
testmpiio.out
lmm_stripe_count:   10
lmm_stripe_size:    1048576
lmm_pattern:        1
lmm_layout_gen:     0
lmm_stripe_offset:  1
        obdidx           objid           objid           group
             1         4673479        0x474fc7               0
             0         4600893        0x46343d               0
            20         4551236        0x457244               0
             3         4701254        0x47bc46               0
            21         4479152        0x4458b0               0
            19         4696884        0x47ab34               0
             5         4704057        0x47c739               0
             7         4647142        0x46e8e6               0
            16         4640736        0x46cfe0               0
            18         4595400        0x461ec8               0
```

Example: File striped across 10 OSTs. Each OST stores 1 MB objects.

Lustre Striping Parameters

lfs setstripe –S <size> -c <count> filename

```
[pmalakar@cn364 testq]$ rm testmpiio.out
[pmalakar@cn364 testq]$ time dd if=/dev/zero of=testmpiio.out bs=10M count=1000
1000+0 records in
1000+0 records out
10485760000 bytes (10 GB) copied, 18.2025 s, 576 MB/s

real    0m18.205s
user    0m0.004s
sys     0m10.042s
[pmalakar@cn364 testq]$ rm testmpiio.out
[pmalakar@cn364 testq]$ lfs setstripe -S 2M -c 10 testmpiio.out
[pmalakar@cn364 testq]$ time dd if=/dev/zero of=testmpiio.out bs=10M count=1000
1000+0 records in
1000+0 records out
10485760000 bytes (10 GB) copied, 10.4116 s, 1.0 GB/s

real    0m10.420s
user    0m0.003s
sys     0m10.406s
```

Striping Benefit

8 MB

```
Time 0.010138
Time 0.013419
Time 0.027182
Time 0.075958
Time 0.219819
Time 0.333267
```

256 MB

Stripe count = 1

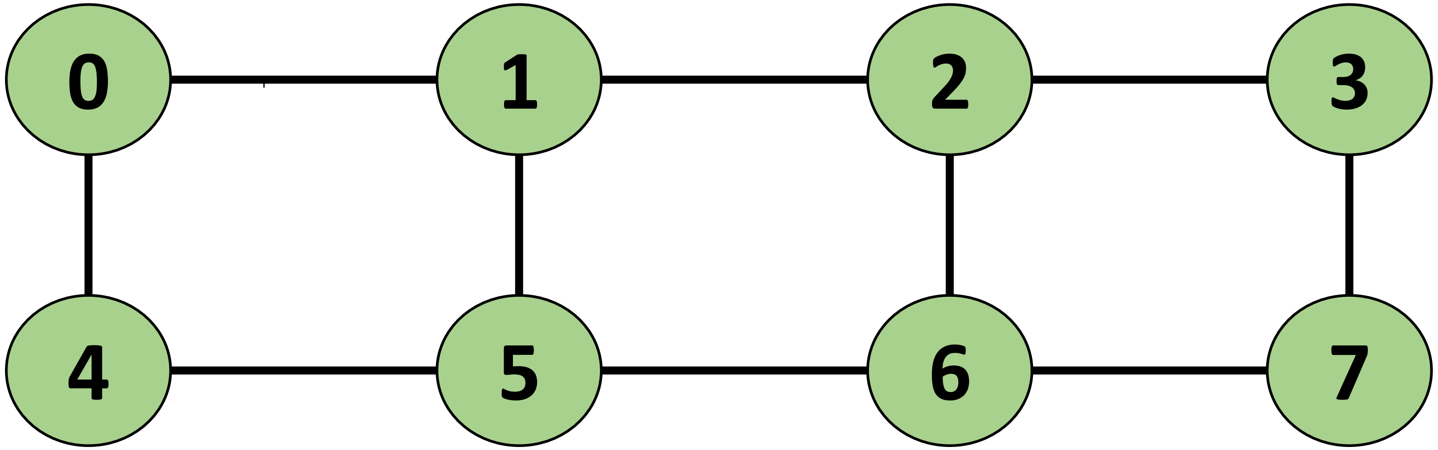
8 MB

```
Time 0.020716
Time 0.025181
Time 0.035053
Time 0.063688
Time 0.220986
Time 0.223855
```

256 MB

Stripe count = 18

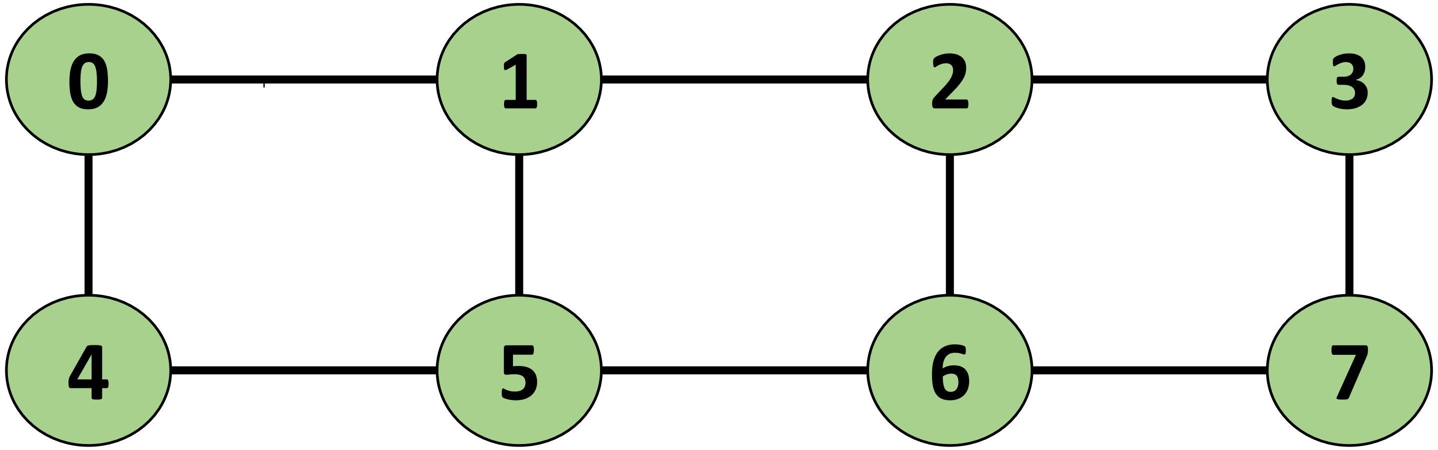
Dimension-order Routing

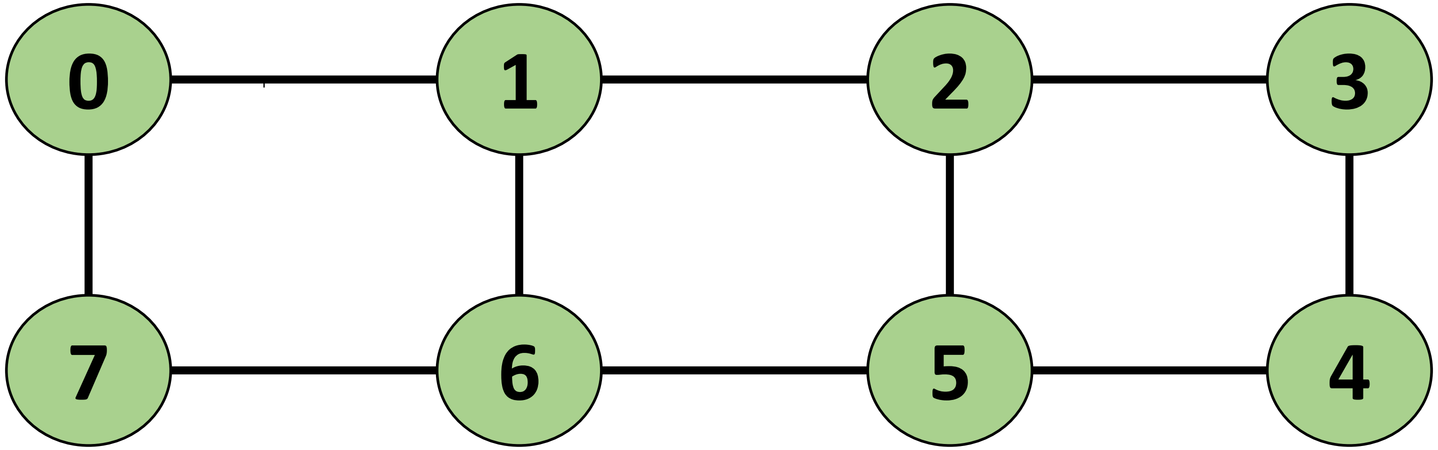


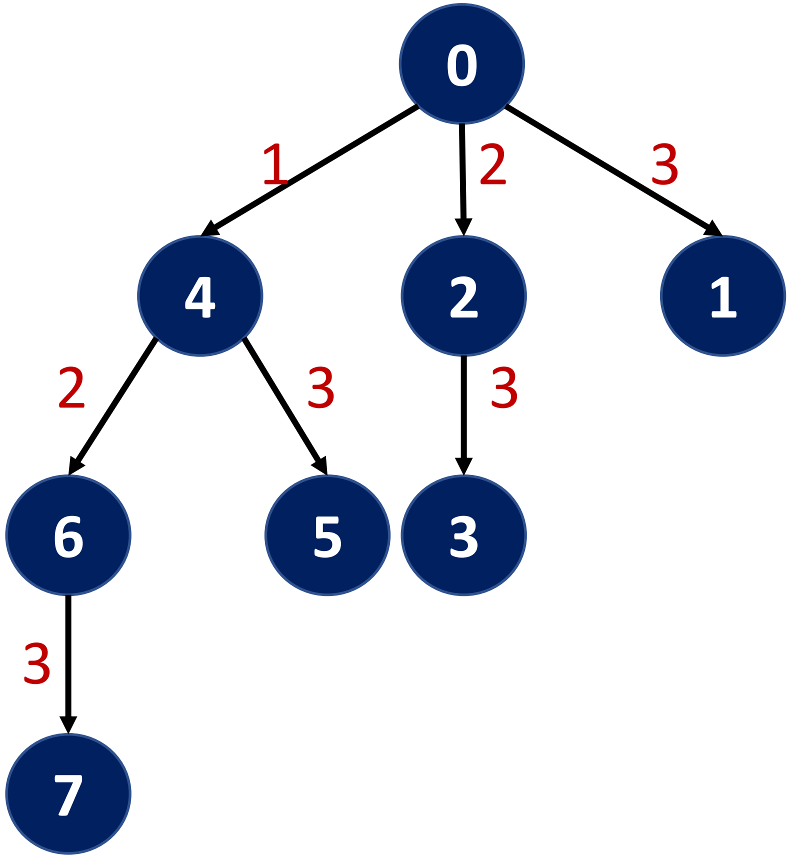
0 -> 2
XY: 0 -> 1 -> 2
YX: 0 -> 1 -> 2

0 -> 5
XY: 0 -> 1 -> 5
YX: 0 -> 4 -> 5

Bcast Communications







Supercomputer Job Allocation

Batch Queueing Systems

- Schedules jobs based on queues
- Has full knowledge of queued, running jobs
- Has full knowledge of the resource usage
- Often combination of best fit, fair share, priority-based
- Designed to be generic, can be customized
- Suited to meet demands of the scheduling goals of the centre
- Typically FIFO/FCFS with backfilling

Workload managers/Schedulers

- Portable Batch System (PBS)
- LoadLeveler
- Application Level Placement Scheduler (ALPS)
- Moab/Torque
- Simple Linux Utility for Resource Management (SLURM)

Desirable Features of Scheduler

- Fair
- Simple
- Low average queue wait times
- High system utilization
- Provide optimum performance for all kinds of jobs
- Support different job classes (interactive vs. batch)
- Provide priority for special jobs

David Lifka, The ANL/IBM SP Scheduling System, JSSPP 1995

ANL IBM SP System Observations (Typical User Requirement)

Required Nodes	Required Time
1 - 8 nodes	8 - 48 hours
16 - 32 nodes	1 - 8 hours
64 - 128 nodes	30 minutes - 3 hours

Users were asked to use their scheduler and provide feedback

FCFS with Backfilling

- FCFS scheduling
  - Poor system utilization
- Backfilling – to overcome inefficiency of FCFS
- Scan the queue of jobs for a job that does not cause the first queued job to wait for any longer than they otherwise would
- Improve system utilization
- Lower queue waiting times

Backfilling – 128-node Example

128

User Name	Number of Nodes	Number of Minutes	Job Status
User A	32	120	Startable
User B	64	60	Waiting
User C	24	180	Waiting
User D	32	120	Waiting
User E	16	120	Waiting
User F	10	480	Waiting
User G	4	30	Waiting
User H	32	120	Waiting

96

User Name	Number of Nodes	Number of Minutes	Job Status
User A	32	120	Running
User B	64	60	Startable
User C	24	180	Waiting
User D	32	120	Waiting
User E	16	120	Waiting
User F	10	480	Waiting
User G	4	30	Waiting
User H	32	120	Waiting

User Name	Number of Nodes	Number of Minutes	Job Status
User A	32	120	Running
User B	64	60	Running
User C	24	180	Startable
User D	32	120	Waiting
User E	16	120	Waiting
User F	10	480	Waiting
User G	4	30	Waiting
User H	32	120	Waiting

32

User Name	Number of Nodes	Number of Minutes	Job Status
User A	32	120	Running
User B	64	60	Running
User C	24	180	Running
User D	32	120	Blocked
User E	16	120	Ineligible
User F	10	480	Ineligible
User G	4	30	Startable
User H	32	120	Waiting

8

User Name	Number of Nodes	Number of Minutes	Job Status
User A	32	120	Running
User B	64	60	Running
User C	24	180	Running
User D	32	120	Blocked
User E	16	120	Ineligible
User F	8	480	Startable
User G	4	30	Waiting
User H	32	120	Waiting

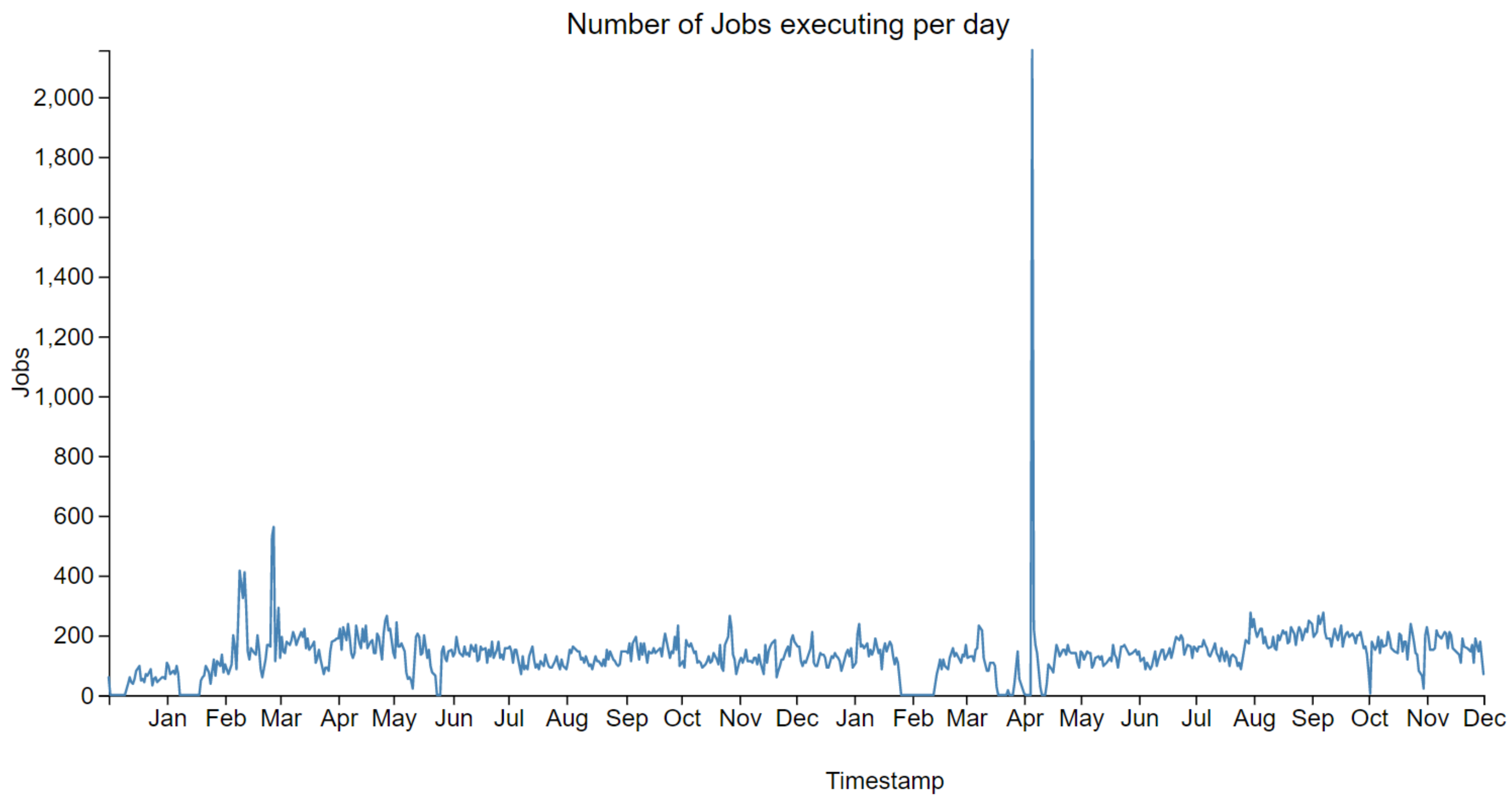
0

Scheduler Queues

- Jobs are submitted to a queue
- Different queuing policies (decided by the administrator)
- Multiple queues in some systems
  - Based on the usage
  - Queue waiting time different
  - Static vs. dynamic partitioning

Anomaly

From: 01/12/2015 To: 01/12/2017 Type : Jobs Executing per day Submit



Jobs executing per day on HPC2010

An Example Scheduling Policy (144 nodes)

1. Prime time, 6 AM to 6 PM

   a. When less than 113 nodes in use:
      1-32 node jobs limited to < 4 hours.
      >32 node jobs limited to < 10 minutes.

   b. When more than 112 nodes are already in use:
      jobs limited to < 10 minutes. This maintains high availability on the last 32 nodes.

Henderson, “Job Scheduling Under the Portable Batch System”, JSSPP 1995.

An Example Scheduler Script

```
foreach job {
 if (job_state == "Q") {
  if ((totpool - usepool) > 32) {
   if ((nodect<33)&&(walltime>4h))
    continue;
   if ((nodect>32)&&(walltime>10m))
    continue;
  } else {
   if ((nodect>=32)||(walltime>10m))
    continue;
  }
  if (anodes =="yes") {
   run;
   break;
  } else if (anodes == "never")
   delete JID REASON;
 }
}
} else if ((DAY>=Mon)&&(DAY<=Fri) &&
   ((NOW>=4:00:00)&&(NOW<16:00:00)) ||
   ((NOW>=18:00:00)&&(NOW<22:00:00))){
# Interactive night
 foreach job {
  if ((job_state == "Q") &&
      (queue_type == "E")) {
```

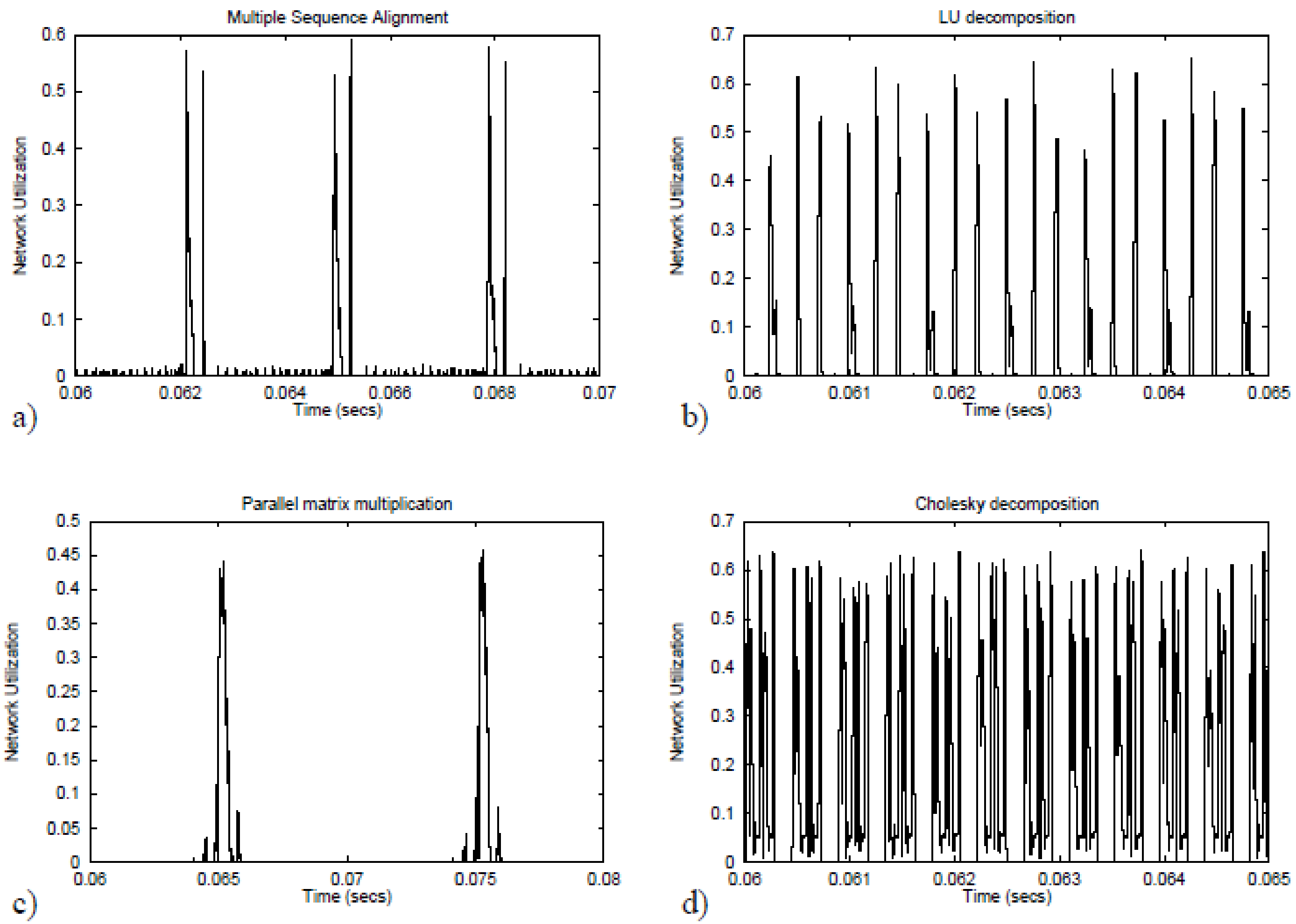
Henderson, "Job Scheduling Under the Portable Batch System", JSSPP 1995.

Resources Required

- Number of nodes
- Wall-clock time
- Users are charged for node-hours/core-hours

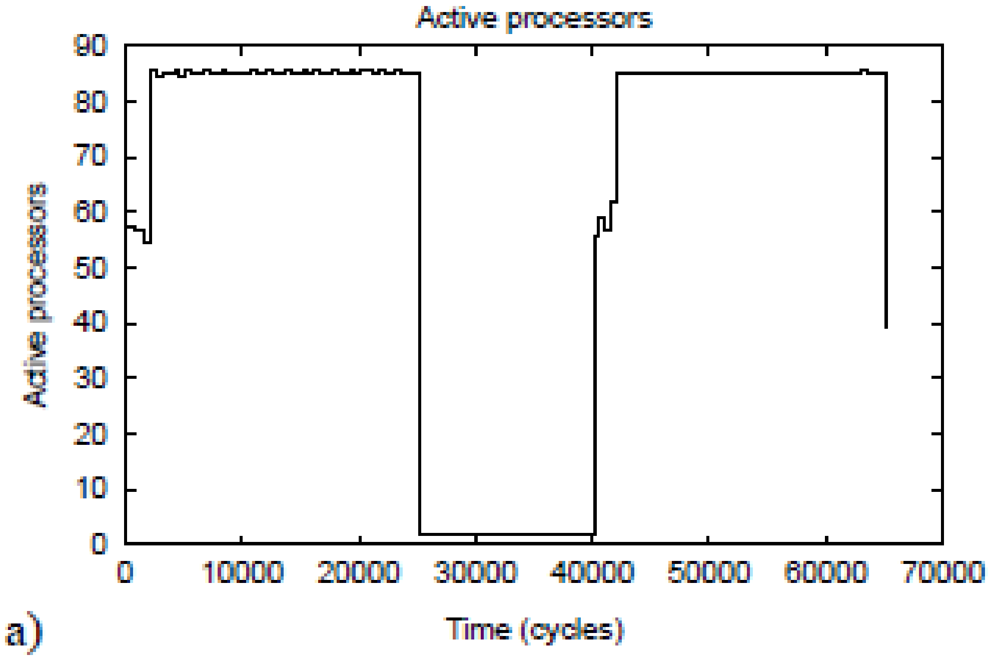
What is missing?

Network Utilization in Different Applications

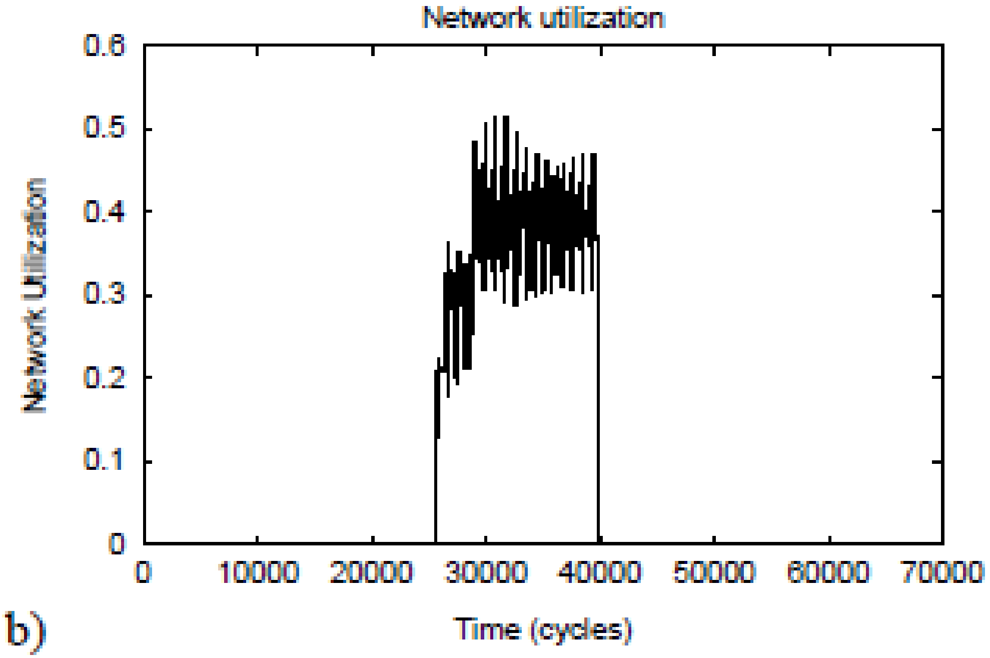


Petrini and Feng, Time-Sharing Parallel Jobs in the Presence of Multiple Resource Requirements, JSSPP 2000

Network Utilization in FFT



a)



b)