PARTEC CLUSTER COMPETENCE CENTER

MANAGED HPC

Introduction to JuRoPA:

ParaStation Cluster Suite

21.05.2012

Ralph Krotz

krotz@par-tec.com

History

Components

- Communication Library
- Process Management
- MPI

ParaStation @ JSC

Infiniband

- Hardware Components
- Topology

MPI scaling

Running your program

History

ParaStation1 (1995)
- Univ. of Karlsruhe

ParaStation2 (1998)
- ParTec was founded

ParaStation3 (2001)
- Support for new interconnects – GigE

ParaStation4 (2003)
- More interconnects, new features provide a comprehensive cluster suite

ParaStation goes open source (2004)
- support contract is mandatory for enterprise customers

ParaStation V5 (2007)
- MPI2 support

ParaStation Consortium

ParaStation is open source

- Still owned by ParTec
- Co-Development by the Consortium partners
- Guarantee for other partners, if ParTec disappears
- Free for educational customers – without support
- ParTec has the exclusive right commercial utilization
- Support contract mandatory for enterprise customers
- Closed source add-ons

History

Components

- Communication Library
- Process Management
- MPI

ParaStation @ JSC

Infiniband

- Hardware Components
- Topology

MPI scaling

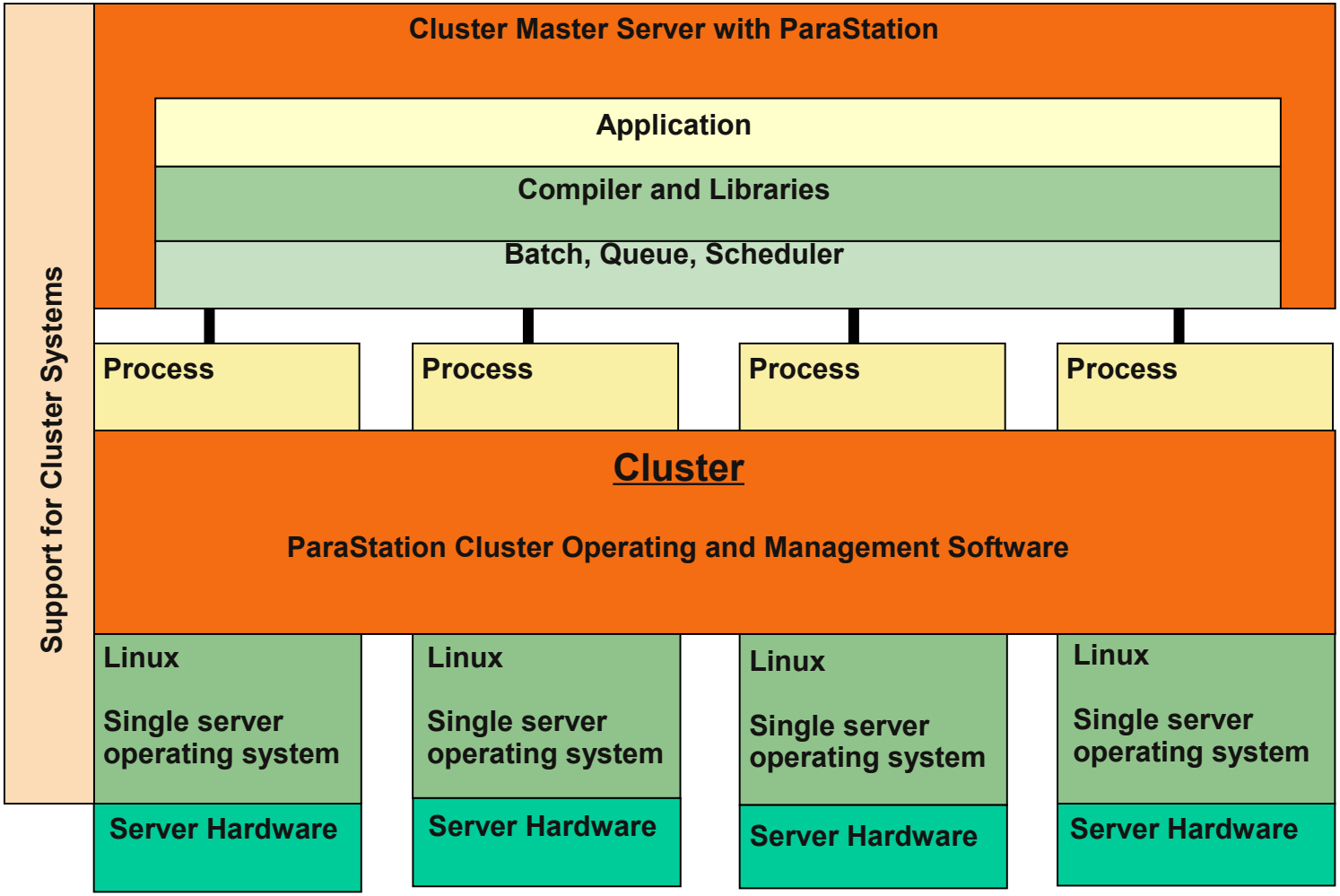
Running your program

ParaStation Components

ParaStation MPI	(psmpi)
ParaStation Process Management	(psmgmt)
ParaStation Communication Library	(pscom)
GridMonitor	(psgridmon)
ParaStation System Provisioning	(pscluster)
Toolbox	
(Batch Queuing System)	
(Standard Distribution)	

Σ ParaStation Cluster Suite

Cluster Operating and Management



ParaStation Communication Library

Supports multiple interconnects & protocols in parallel

Interconnects

- Infiniband
- Shared memory
- Ethernet (GbE, 10GbE)

Protocols

- VAPI (IB)
- DAPL (10GbE, IB)
- TCP/IP
- p4sock (GbE, 10GbE)

Used by MPI

- Uses „best“ interconnect found at startup
- Efficient usage of resources (buffers)

ParaStation MPI

ParaStation MPI library:

- Implements MPI2 (MPIch2 (version 1.2.1p1, 1.4.1p1))
- Uses *pscom* library
- all interconnects from *pscom*
- Supports all MPIch-tools (Tracing, Debugging, ...)
- “One” executable for all interconnects:
  - *Communication path automatically selected at run-time*
  - *Optionally configurable by the user*
  - *Takes care of network problems/bad host adapters*
- Interactive usage incl. resource management
  - *Multi-user environment*
  - *No overbooking of nodes*

Supports various compilers:

- GCC, Intel
- PGI, Pathscale

ParaStation MPI

MPI Features required by Juropa

MPI must scale to more than 25000 processes .

MPI should be optimized for the environment – low memory consumption
(< 0.5MB/connection)

MPI with less memory consumption / connection = More memory for the linpack = better performance & higher efficiency

Only establish "used" connections. (establish the IB connection with the first byte send – **on demand**)

ParaStation MPI:

-More memory for your application
-Better use of your IB fabric

ParaStation Process Management

Daemon concept

- ParaStation **psid** on each cluster node
- Resource monitoring (node, processes, load, …)
- Process startup
- Process monitoring/termination
- Setting up user environment
- Efficient and scalable communication subsystem for inter-daemon messages (RDP, only one socket per daemon)
- No dedicated master daemon - fault tolerance

ParaStation Process Management

Job monitoring (psid)

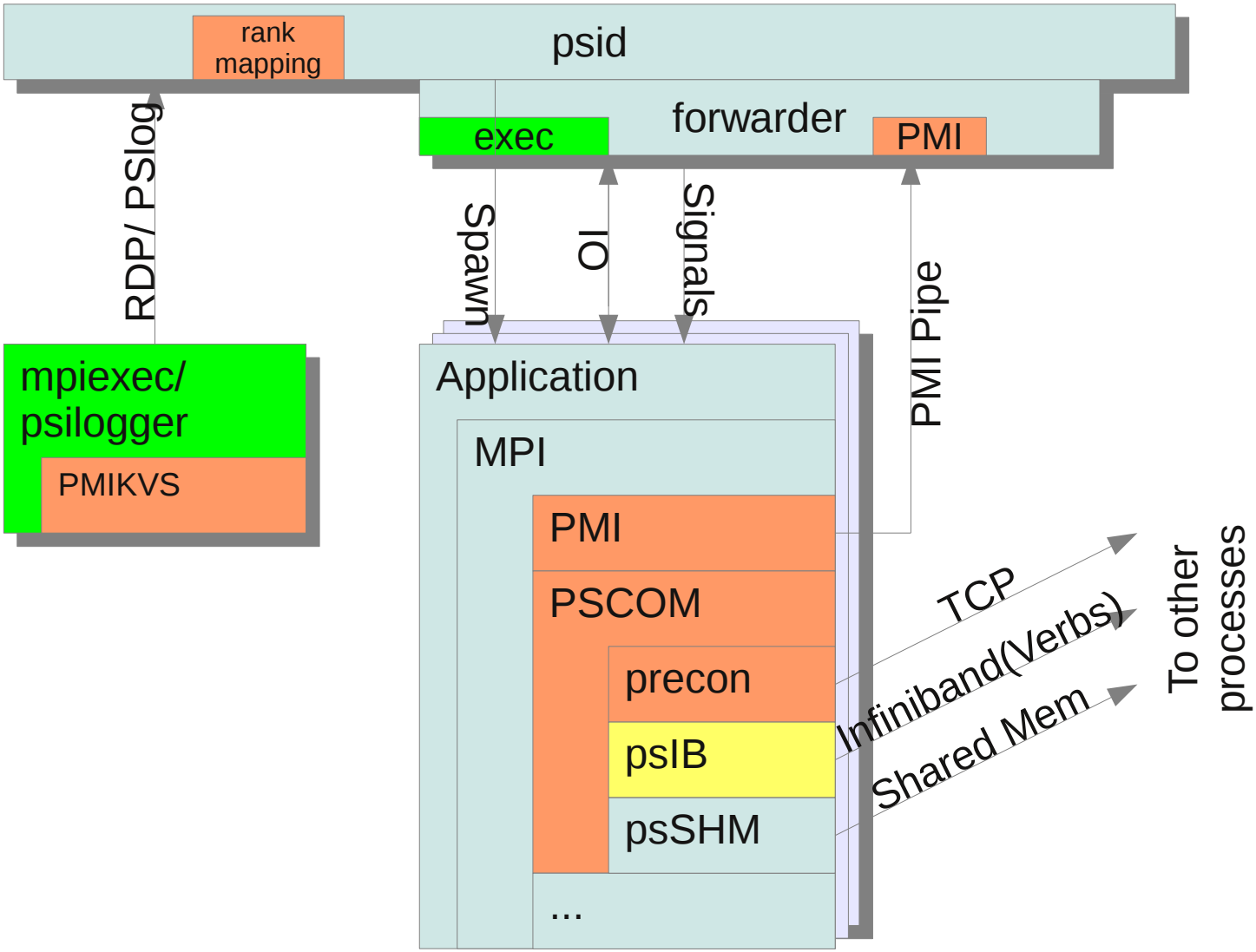
- All processes within all jobs are continuously monitored
- Jobs are cleaned up, if
  - *node dies*
  - *process dies*
  - *job terminates*
- All processes belonging to a particular job are terminated!
- All resources are available for next job
- Job accounting

ParaStation Process Management

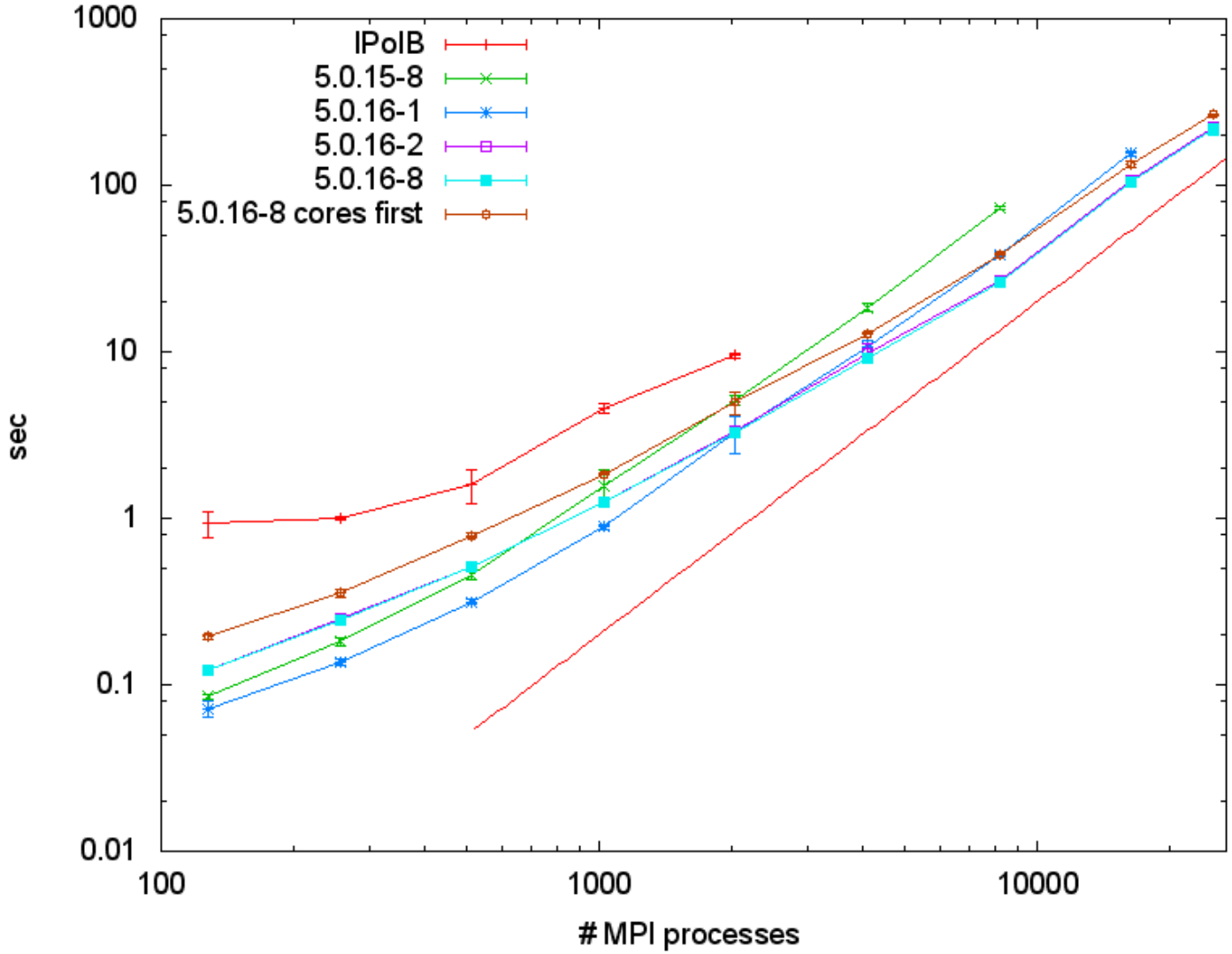
Process placement / load balancing

- Only available nodes are used, dead nodes are ignored!
- Only one process per CPU (default)
- Current load is observed
- Exclusive use of nodes possible
- Explicit node list generated by batch subsystem
- Integrated with TORQUE & Moab
  - *psmom plug-in to psid*
  - *psmom provides communication between psid and batch system*

MPI: Application startup



MPI: Application startup times



- History
- Components
  - Communication Library
  - Process Management
  - MPI
- ParaStation @ JSC
- Infiniband
  - Hardware Components
  - Topology
- MPI scaling
- Running your program

ParaStation @ JSC

ParaStation in Production since 2004

Various Clusters use ParaStation today

JUDGE	2472 cores Intel X5650 412 GPUs Nvidia M2050	IB QDR
Softcomp	500 cores AMD64	GigE / InfiniPath
JUGGLE	176 cores AMD64	InfiniPath

History

Components

- Communication Library
- Process Management
- MPI

ParaStation @ JSC

Infiniband

- Hardware Components
- Topology

MPI scaling

Running your program

Why QDR Infiniband ?

Naive answer:

- Higher bandwidth (now at 32 Gbit/s)
- Lower latency (Mellanox claims less than 1.0 µsec)

Actual reasons:

- Huge fabrics build out of small building-blocks
- “Small” crossbar-switches (few dozen ports)
- Specific topology: Clos, fat-tree, Ω-network,....
- QDR has 36-port building blocks (24-ports in earlier versions)
- Reduced complexity
- QDR supports adaptive routing
- IB's static routing might lead to congestion

Infiniband debugging

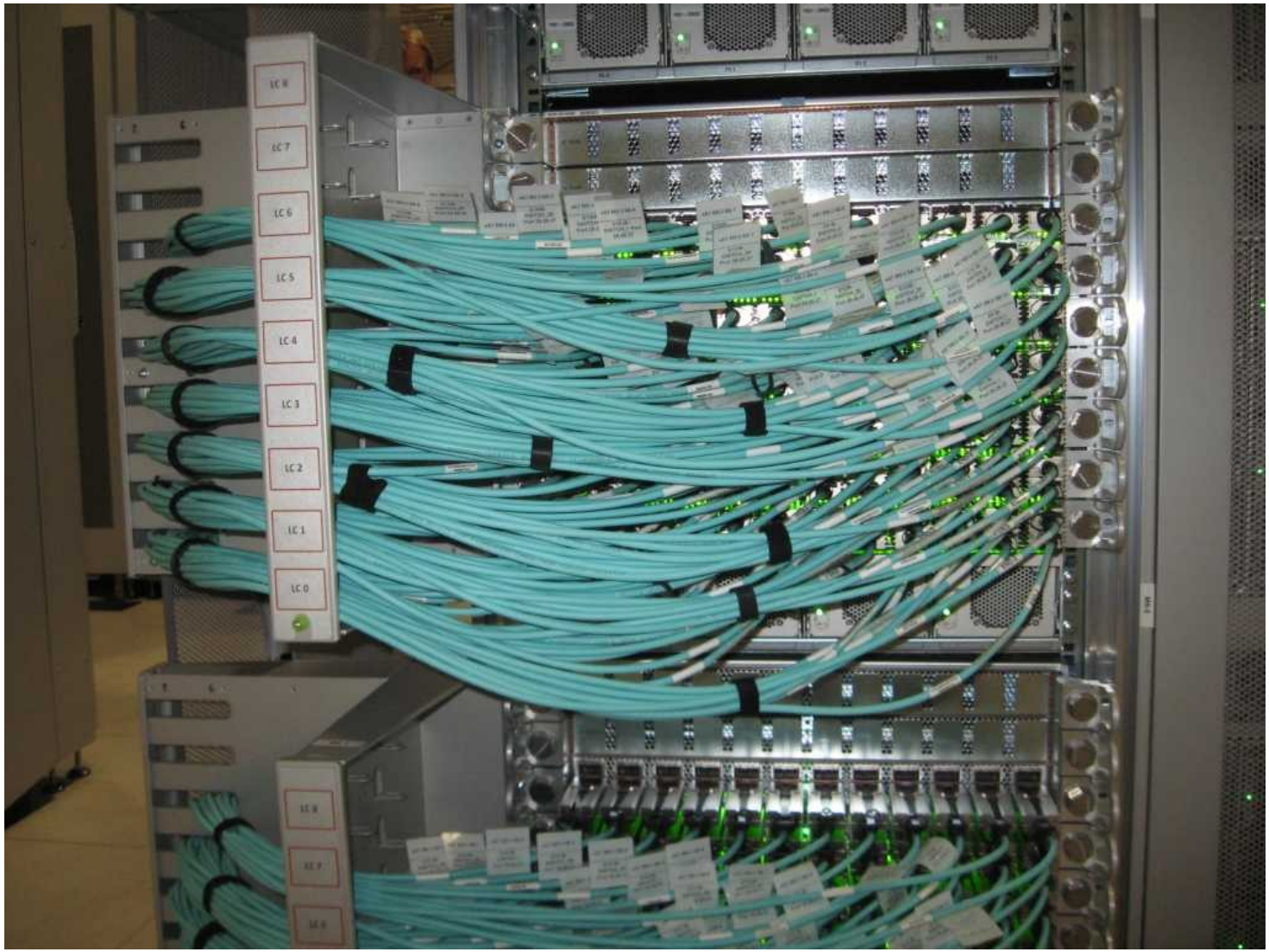
Deal with this:

Extract of output of ibdiagnet:

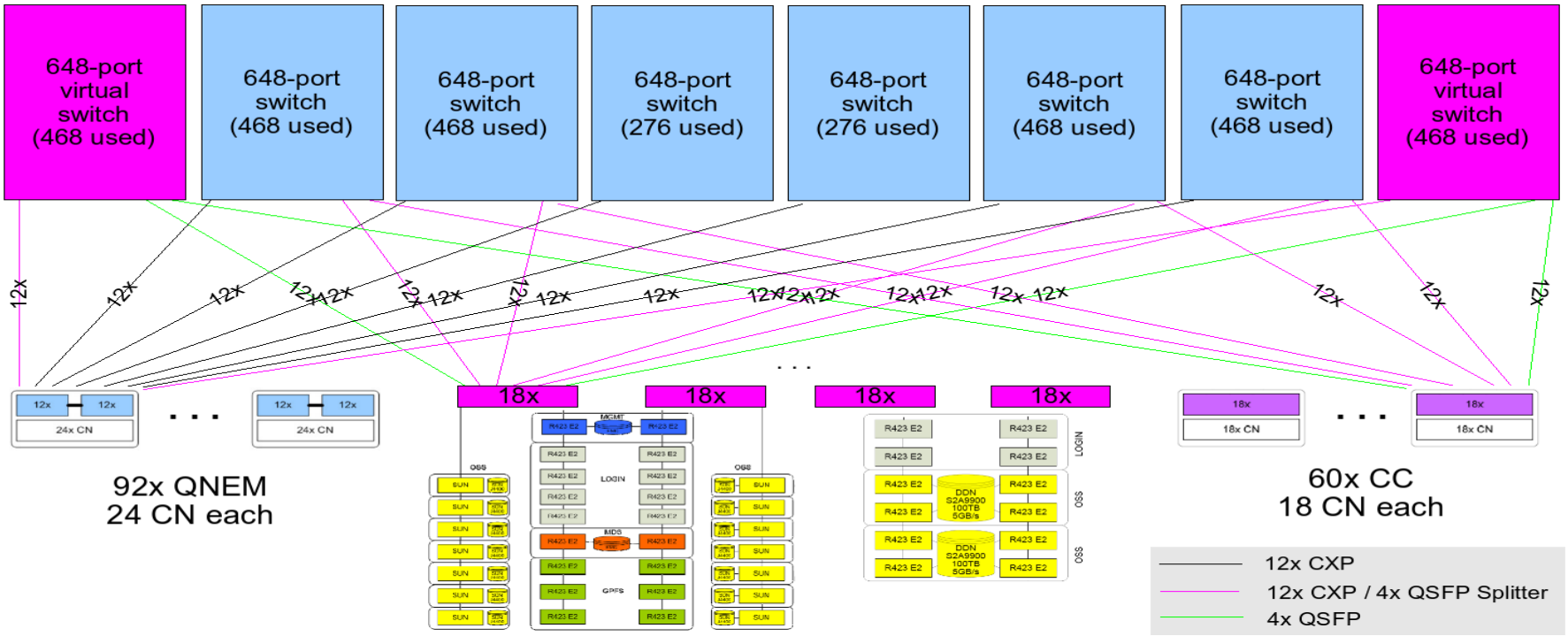
```
-I- Using port 1 as the local port
-W- Local system name was not specified (-s option).
    "jj27y01/U1" will be used as the local system name.
-I- Discovering ... 3881 nodes (588 Switches & 3293 CA-s) discovered.
```

Very large # of IB components – need a Plan to manage this complexity and quickly identify & locate faulty components.

M9 switch



IB Topology - Heterogeneous



23 x 4 QNEM modules, 24 ports each

6 x M9 switches, 648 ports max. each, 468/276 links used

Mellanox MTS3600 switches (Shark), 36 ports, for service nodes

4 Compute Sets (CS) with 15 Compute Cells (CC) each

CC with 18 Compute Nodes (CN) and 1 Mellanox MTS3600 (Shark) switch each

Virtual 648-port switches constructed from 54x/44x Mellanox MTS3600

Results

Single HPL benchmark run – across entire Heterogeneous Cluster

274.8 TFlops with 3200 nodes -> ~91.6% of peak

Nodes used: 2157 Sun nodes and 1043 Bull nodes.

History

Components

- Communication Library
- Process Management
- MPI

ParaStation @ JSC

Infiniband

- Hardware Components
- Topology

MPI scaling

Running your program

MPI Scaling

IB design based on reliable connections (rc mode)

- Needs private memory for each connection
- Might have significant impact for huge jobs

ParaStation MPI is not special in this respect

- Each IB connection requires 0.55 MB of memory (with default settings)
- No real problem for small jobs

Largest jobs currently on JuRoPA have 8192 processes

- Thus, 8192 connections for each process
- ~ 4 GB of memory occupied just by MPI
- Each core has just 3 GB of memory (24 GB/node)

MPI Scaling – solution I

Most MPI programs do not need every connection

- Nearest neighbor communication
- Scatter/Gather and Allreduce based on binary trees
- Typically just a few dozen connections when having hundreds of processes

ParaStation MPI supports this with “on demand connections”

- `export PSP_ONDEMAND=1`
- was used for the Linpack runs (np > 24000)
- `mpiexec –ondemand`

Backdraw

- Late all-to-all communication might fail due to short memory

Default on JuRoPA is not to use “on demand connections”

MPI Scaling – solution II

Sometimes all-to-all communication is required

Try to reduce memory-usage by reducing number of buffers

- ParaStation MPI uses pre-allocated buffers
  - *Copy data into local send-buffers*
  - *RDMA into remote receive buffers*
  - *Copy into final destination*
- Default is to pre-allocate 16 send and receive buffers
- Each buffer has 16 kB

Steering via environment variables

- PSP_OPENIB_SENDQ_SIZE
- PSP_OPENIB_RECVQ_SIZE

MPI Scaling – solution II

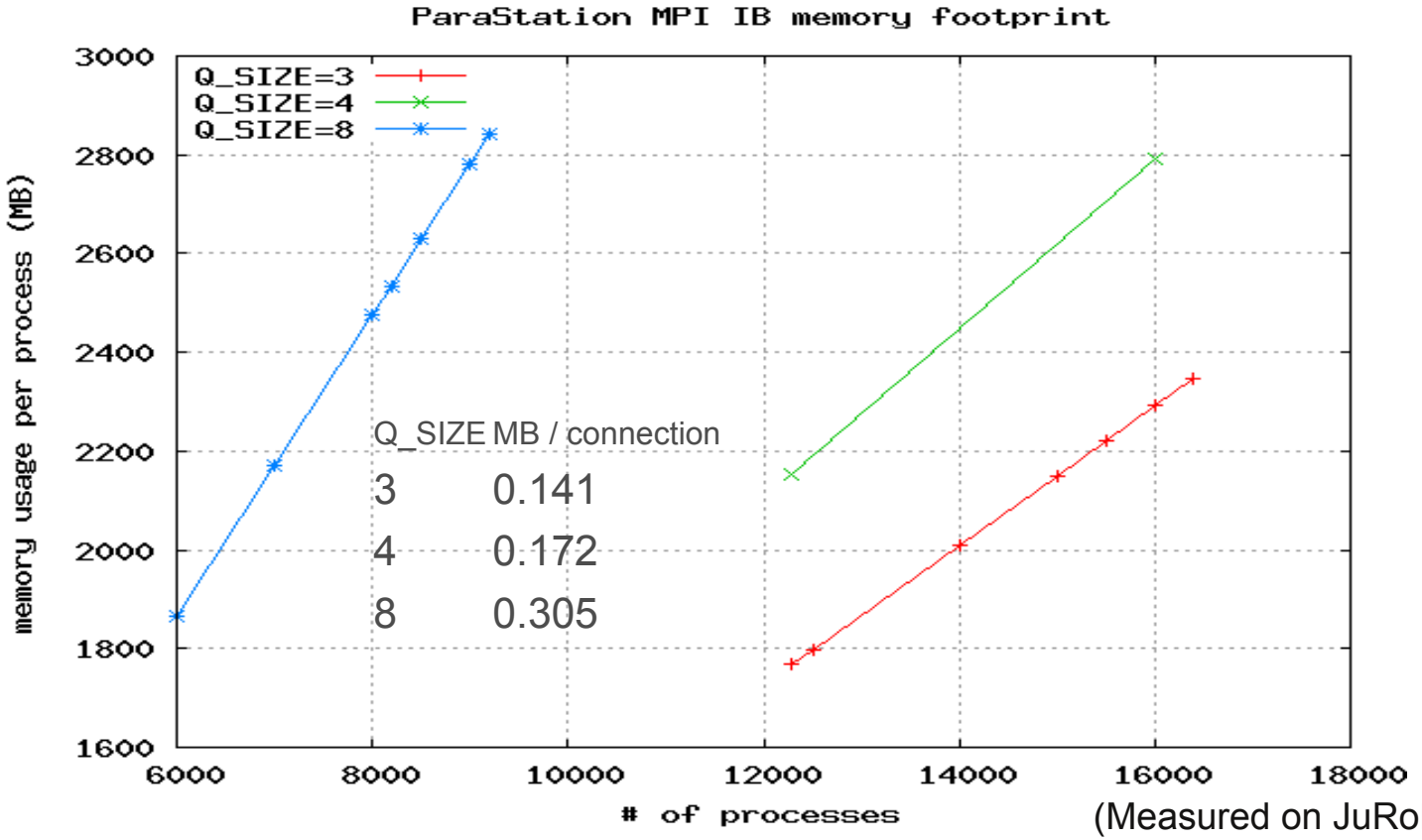
Example:

```
export PSP_OPENIB_SENDQ_SIZE=4
export PSP_OPENIB_RECVQ_SIZE=4
export PSP_ONDEMAND=0
```

allocates 4 * 2 fixed size (16 kB) buffers, resulting in ~170 kB per connection
For np=10000 memory consumption was measured:
~1800 MB/process = ~14 GB/node

Memory footprint (all 2 all)

With 16k processes you loose more than 75% of your memory for communication buffers and connection handling!
(JuRoPA 24GB/node ~ 3GB/process)



(Measured on JuRoPA by N.Eicker)

History

Components

- Communication Library
- Process Management
- MPI

ParaStation @ JSC

Infiniband

- Hardware Components
- Topology

MPI scaling

Running your program

Running your program

compilation

- Using the wrapper mpicc, mpicxx, mpif90, ...
- Compiler selection:
  - *Module avail*
  - *Module list*
  - *See overview by JSC*

Running your program

mpiexec

- which options for mpiexec?
- which environmental variables?

Running your program

Options:

- mpiexec -np <N> # procs to start
- Normally, no other options necessary

Running your program

Checking Startup

```
machine:/some/dir>export PSP_DEBUG=3
machine:/some/dir>mpiexec -np 2 ./hello
<PSP 8536:set PSP_DEBUG = 3>
<PSP 8536:# Version(PSCOM): Mar 26 2009>
<PSP 8536:default PSP_SO_SNDBUF = 32768>
<PSP 8536:default PSP_SO_RCVBUF = 32768>
… (some more environment variables)
<PSP 8536:default PSP_TCP = 1>
<PSP 8536:Register arch tcp with priority 01.02>
<PSP 8536:default PSP_SHM = 1>
<PSP 8536:Register arch shm with priority 01.90>
<PSP 8536:default PSP_P4S = 1>
<PSP 8536:Register arch p4s with priority 01.10>
<PSP 8536:default PSP_OPENIB = 1>
<PSP 8536:Using   /opt/parastation/lib64/libpscom4openib.so>
<PSP 8536:Register arch openib with priority 01.20>
<PSP 8536:default PSP_MVAPI = 1>
…
<PSP 8536:ACCEPT  (134.94.172.25,8538,0x605398,r0000001)
to (134.94.172.25,8536,0x605398,r0000000) via shm>
```

Running your program

Stopping your job

```
kill <one pid of your MPI job>
PSIlogger: Child with rank 1 exited on signal 15: Terminated
PSIlogger: Child with rank 0 exited on signal 15: Terminated
```

If the job aborted:

Check the first lines of your output

Available memory exceeded ?

Running your program

My program does not start ...

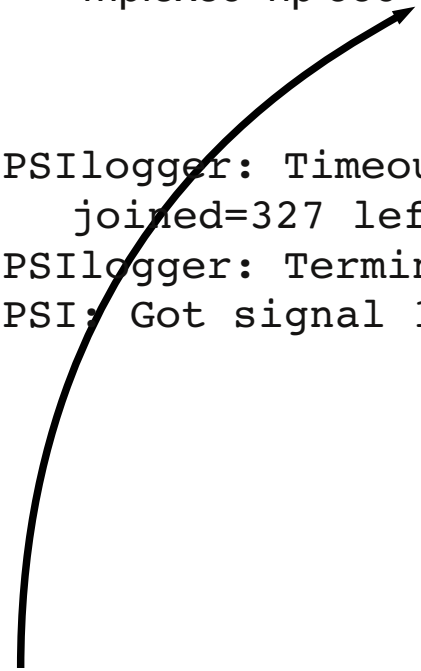
- PMI Barrier problems...
- MPI/ParaStation problems...
- IB problems...
- Use PSP_DEBUG=3 to see what's going on.

Running your program

PMI Barrier problem:

Symptom:

mpiexec -np 336 -x ./binary -options...



```
PSIlogger: Timeout: Not all clients joined the first pmi barrier:
   joined=327 left=9 round=1
PSIlogger: Terminating the job.
PSI: Got signal 15 from 0x0bb8235f[3000:9055]
```

-x exports the whole environment, too much stress

Running your program

PMI Barrier problem: how to avoid it

- Don't use -x, use --export and a , separated list
  --export=MYVAR,YOURVAR
- Try increase the PMI barrier timeout:
  export PMI_BARRIER_TMOUT=<secs>, -1 for infinite wait, default is 60 seconds
- Try increase the rounds of PMI barriers
  export PMI_BARRIER_ROUND=<#>, default is 1 round

Running your program

Other tuning parameters:

- Check out *man ps_environment*
- Beware: some for debugging, testing …
- Example:
  PSP_ OPENIB_PATH_MTU: Controls IB MTU: 1 = 256 Bytes, 2 = 512 Bytes, 3 = 1024 Bytes (default)
- See above:
  - PSP_ONDEMAND

    Works best with nearest neighbor communications, scatter/gather, allreduce
  - PSP_OPENIB_SENDQ_SIZE
    PSP_OPENIB_RECVQ_SIZE

    Queue size modification is only viable option for all-to-all communication

Thank you!