Non-Intrusive Migration of MPI Processes in OS-bypass Networks

Simon Pickartz\textsuperscript{1}, Carsten Clauss\textsuperscript{2}, Stefan Lankes\textsuperscript{1}, Stephan Krempel\textsuperscript{2}, Thomas Moschny\textsuperscript{2}, and Antonello Monti\textsuperscript{1}

\textsuperscript{1}RWTH Aachen University, Aachen, Germany
\textsuperscript{2}ParTec Cluster Competence Center GmbH, Munich, Germany
Why do we need Migration?

Resiliency

- Increasing hard- and software failures with growing cluster sizes
- Evacuation of faulty nodes instead of whole job aborts

Load balancing

- Applications’ scalability is usually limited by a single resource
- Co-scheduling can help to improve the overall cluster utilization
  - Revocation of an exclusive node assignment
  - Necessity for dynamic schedules
What about Checkpoint / Restart?

- Can be regarded as heavyweight counterpart of migration
- All processes of a job are affected
- Unnecessary synchronization overhead
- local vs. global consistency
  - Node evacuation only affects processes running on the particular node
  - Load balancing by moving only a subset of processes
Agenda

- Goals

- Background
  - The pscom Library
  - OS-bypass Networks

- Migration of MPI Processes

- Evaluation
  - Overhead
  - Migration Time

- Conclusion
Goals

1. Avoidance of any runtime overhead
2. Minimization of the additional migration costs
3. Application transparency
4. Platform / hardware independence
Goals

1. Avoidance of any runtime overhead
   → No influence on the application’s performance without migrations
2. Minimization of the additional migration costs
3. Application transparency
4. Platform / hardware independence
Goals

1. Avoidance of any runtime overhead
   → No influence on the application’s performance without migrations
2. Minimization of the additional migration costs
   → Minimal influence on the migration performance itself
3. Application transparency
4. Platform / hardware independence
Goals

1. Avoidance of any runtime overhead  
   → No influence on the application’s performance without migrations
2. Minimization of the additional migration costs  
   → Minimal influence on the migration performance itself
3. Application transparency  
   → Migrations without adaption to the application’s code
4. Platform / hardware independence
Goals

1. Avoidance of any runtime overhead
   → No influence on the application’s performance without migrations

2. Minimization of the additional migration costs
   → Minimal influence on the migration performance itself

3. Application transparency
   → Migrations without adaption to the application’s code

4. Platform / hardware independence
   → No tailored solution to one interconnect
The pscom Library

- Communication layer of ParaStation MPI
- MPICH derivate with full MPI-3 support
- Plugins for different interconnects
  - Chosen based on a priority / fall-back scheme
  - Point-to-point channels
- Internal message queueing facility
- On-demand connection establishment
OS-bypass Networks

- Direct access to the hardware from the user application
- Connection state information managed within the hardware
- Employment in virtualized environments via
  - PCIe pass-through
  - Single-root I/O virtualization
- **Location-dependent** resources exacerbate migrations
Application Transparent Migration

- Shutdown / Reconnect instead of Checkpoint/Restart
  - Local consistency is sufficient
  - Distinguish between migratable and non-migratable connections

- Requirements for the protocol
  - Point-to-point communication Channels
  - Reliable channels

- Employed in virtualization context
  1. *Tear-down of all non-migratable connections*
  2. Detach the HCA via ACPI hot-plug
  3. Migrate the VM to the target host
  4. Attach the new HCA on the target host to the VM
  5. *Re-establish the connection on-demand*
Application Transparent Migration (contd.)

- Add migration states to pscom
- Trigger cycle by external request
- Only affects non-migratable connections
Application Transparent Migration (contd.)

Migration framework

- suspend request
- write suspend
- suspend response
- migrate
- resume request

Process to be migrated

- shutdown request
- shutdown response
- read suspend
- reset to on-demand
- read resume
- write resume

Peer

- read suspend
- write suspend
- reset to on-demand
- read resume
- write resume
What about “Ungracious” Applications?

Lazy / Cooperative

- Set flag in callback function
- Delay Shutdown / Reconnect until process enters pscom
  → Asynchronous Shutdown / Reconnect

**Pro**

- No further communication channel
- Minimal overhead during migration

**Contra**

- Migrated process has to enter pscom
- Peer processes have to enter pscom
  → Delays in long computation phases
What about “Ungracious” Applications? (contd.)

Instantaneous / Threaded

- Trigger Shutdown / Reconnect protocol within callback
- Start progress thread on receipt of remote migration request
- Pro
  - Instantaneous migration
  - No dependencies to the application’s communication behavior
- Contra
  - Little overhead during migration (i.e., caused by progress thread)
  - Out-of-band channel with multi-cast required
Test Environment

- 4-node Cluster
  - 2 Sandy-bridge Systems
  - 2 Ivy-bridge Systems

- InfiniBand FDR Mellanox Fabric
  - Up to 56 GiB/s
  - Support for SR-IOV

- Software stack
  - CentOS 7.1
  - Mellanox OFED stack (v 3.0-1.0.1)
  - QEMU / KVM 2.3.0
Runtime Overhead

Throughput

Throughput in MiB/s

Size in Byte

0 128 1 Ki 8 Ki 65 Ki 512 Ki 4 Mi

Throughput

- Standard
- S/R
- S/R (threaded)
Latency

- No influence on the critical path
- In case of permanently enabled thread function
  - ≡ Period 1 ms
  - ≡ More contention on internal locks

<table>
<thead>
<tr>
<th></th>
<th>Disabled</th>
<th>S/R</th>
<th>S/R (threaded)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Norm</td>
</tr>
<tr>
<td>Ø</td>
<td>1.10</td>
<td>1.10</td>
<td>1.10</td>
</tr>
<tr>
<td>σ</td>
<td>0.11</td>
<td>0.11</td>
<td>0.11</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Perm</td>
</tr>
<tr>
<td>Ø</td>
<td></td>
<td></td>
<td>1.17</td>
</tr>
<tr>
<td>σ</td>
<td></td>
<td></td>
<td>0.83</td>
</tr>
</tbody>
</table>
Scalability

- Migration Time in s vs. # Connections
- Time to S / R per Connection

Legend:
- Suspend
- Detach
- Migration
- Attach
- Resume
Migrating mpiBLAST

Migration Time

Migration Time in s

Migration Time

Suspend

Detach

Migration

Attach

Resume

# MPI Processes

Migration Time in s

0

1

2

3

4

5

6

8

16

32

64
Conclusion

- Protocol for application transparent migration
  - Establishes local consistency
  - Executed on a per-connection basis
- Working prototype
  - No runtime overhead
  - Minimal migration cost
  - Successfully evaluated for InfiniBand
- Future work
  - User container-based migration
  - Try mechanism with load balancer to prove benefits at a larger scale
  - Comparison to Checkpoint / Restart mechanisms
- Open source: https://github.com/fast-project/pscom
Thank you for your kind attention!

Simon Pickartz et al. – spickartz@eonerc.rwth-aachen.de

Institute for Automation of Complex Power Systems
E.ON Energy Research Center, RWTH Aachen University
Mathieustraße 10
52074 Aachen

www.eonerc.rwth-aachen.de
Migrating mpiBLAST

Migration Time

Runtime using 16 Procs.

Suspend  Detach  Migration  Attach  Resume