Towards Computing as a Utility via Adaptive Middleware

Jarek Slawinski, Tiziano Passerini, Vaidy Sunderam

Department of Mathematics & Computer Science
Atlanta, GA, USA
Background

- Historical context
  - “Computing may some day be organized as a public utility” – John McCarthy, 1961
  - Remote job entry, timesharing: 1960’s and 70’s
  - Distributed resource aggregation/sharing: 1980’s and 90’s
  - Grid computing: 1990’s and 00’s
  - Cloud Computing: 2000’s and 10’s

- What is the biggest challenge?
  - Mismatch between application & target platform
    - Programming model, dependencies, versions, other mismatches

- What do other utilities do?
  - (1) Standards; (2) Adapters
Resource heterogeneity

<table>
<thead>
<tr>
<th>Access</th>
<th>ssh/interactive, web services, queue systems, authentication, authorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deployment mechanisms</td>
<td>Stage in, stage out, discovery, configuration, build, install</td>
</tr>
<tr>
<td>Capabilities</td>
<td>Any mix of computational, storage, network, etc. properties, libraries, interfaces</td>
</tr>
<tr>
<td>Operating environment</td>
<td>Privilege levels, system software, support</td>
</tr>
<tr>
<td>Machine architecture</td>
<td>Processors, block devices, fabric</td>
</tr>
</tbody>
</table>

- Homogenization/standardization:
  - Does one size fit all?
  - Unification conceals valuable unique features
**ADAPT** approach

- Adaptive Application and Platform Transformation
  - Framework to
  - Enhance the usability of cyber-infrastructure platforms for
  - Legacy science and engineering applications

- Ultimate goal: use any resource for an application
  - Framework matches applications to resource back-ends
  - Deploying adapters/adaptations as needed

- No manual modification of the application (porting)
  - Name substitutions (source code)
  - Library replacements (execution environment)
  - Outsource capabilities (run-time)
Dynamic adaptation vision

- **Situation-specific adapters** that match a particular application(s) with a particular target machine(s)

- **Classes of applications:**
  - Parallel
  - Script-based
  - Workflows
  - Composite
Range and levels of ADAPTation

- Basic model goal: application requirements match resource capabilities

- Adapters provision missing capabilities for an application
  - Modulate app requirements
  - Assemble new capabilities
- Different scopes of adaptation

![Diagram showing the range and levels of ADAPTation with categories for application requirements and resources, and various types of adapters like wrappers, library substitutions, platform transformations, tasks outsourcing, and null adaptation.](image-url)
Situation-specific adapters

- One application may have several adapters/adaptations tailored for specific target platforms
Adaptations

- **Application-side**
  - Source level (simple name) substitutions
  - FUSE – for wrappers and code modifications
  - Static linking with modified/specialized/tailored libraries
  - Library injection techniques (LD_PRELOAD, LD_LIBRARY_PATH, ld.so.conf, *.local)
  - System call interception/virtualization

- **Platform-side (Unibus project)**
  - Defines *Capability Model*
  - Soft Conditioning – installs software/libraries as needed
  - Successive Conditioning – composition
Unibus environment conditioning

- Using successive conditioning, Unibus may modify the specialization level of the target platform.
**ADAPT Framework**

1. Determine application requirements
2. Enumerate potential target resource candidates
3. Provide needed adapters/adaptations to match app to chosen target
4. Set up environment (e.g., staging, configuration)
5. Monitor runtime
Use case 1 – “similar” platforms

- LifeV is a Computational Fluid Dynamics library (C++/Trilinos)
  - FEM – Finite Element Method, Hemodynamics

   ![Reconstruction from CT](image1)
   ![Mesh generation](image2)
   ![Solution](image3)

- MPI simulation code: block decomposition, matrix-vector ops, nearest neighbor comms
- Deploy on 4 similar targets, ?effort, ?performance and cost
Four target computing platforms

- **Experiment:** (a) ease of deployment; (b) optimal platform?

- **puma:** "home" on-premises 128-core cluster
- **ellipse:** university 1024-core cluster
- **lagrange:** HPC cluster in Cilea Supercomputer Center
- **ec2:** Amazon’s IaaS instances from their “cluster compute” type offering

<table>
<thead>
<tr>
<th></th>
<th>puma</th>
<th>ellipse</th>
<th>lagrange</th>
<th>ec2</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu arch.</td>
<td>Opteron 2/2</td>
<td>Opteron 2/2</td>
<td>Xeon 2/6</td>
<td>Xeon 2/8</td>
</tr>
<tr>
<td># cpu/cores</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RAM/core</td>
<td>1GB</td>
<td>1GB</td>
<td>1.3GB</td>
<td>3.8GB</td>
</tr>
<tr>
<td>network</td>
<td>1GbE</td>
<td>1GbE</td>
<td>IB 4X DDR</td>
<td>10GbE</td>
</tr>
<tr>
<td>storage</td>
<td>OK</td>
<td>insufficient disk quota</td>
<td>OK</td>
<td>insufficient image mod</td>
</tr>
<tr>
<td>access</td>
<td>user space</td>
<td>user space</td>
<td>user space</td>
<td>root</td>
</tr>
<tr>
<td>support</td>
<td>full</td>
<td>very limited</td>
<td>limited</td>
<td>none</td>
</tr>
<tr>
<td>build env.</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yum</td>
</tr>
<tr>
<td>compiler</td>
<td>GCC 4.3.4</td>
<td>GCC 4.1.2</td>
<td>GCC 4.1.2</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yum</td>
</tr>
<tr>
<td>dependencies</td>
<td>all</td>
<td>none</td>
<td>blas, lapack src. install.</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>src. install.</td>
<td>src. install.</td>
</tr>
<tr>
<td>MPI</td>
<td>Open MPI</td>
<td>none</td>
<td>Open MPI</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yum</td>
</tr>
<tr>
<td>parallel jobs</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>execution</td>
<td>PBS</td>
<td>SGE</td>
<td>PBS</td>
<td>shell</td>
</tr>
</tbody>
</table>
## Soft-conditioning

<table>
<thead>
<tr>
<th></th>
<th>Puma (home)</th>
<th>Ellipse</th>
<th>Lagrange</th>
<th>EC2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simul.App</td>
<td></td>
<td></td>
<td></td>
<td>User space installation</td>
</tr>
<tr>
<td>LifeV</td>
<td></td>
<td></td>
<td></td>
<td>User space installation</td>
</tr>
</tbody>
</table>
| Trilinos         | Srcs exist but recomp. Each run |          |          | Determine existence of: libraries, include files *Followed by*  
|                  |             |         |          | Common user space installation process  
|                  |             |         |          | E.g. wget tar.gz or bin (ACML), ./configure, make  
|                  |             |         |          | prepare cmake conf.; invoke: cmake, make, make install  
| ParMETIS         |             |         |          | User space installation on demand |
| SuiteSparse      | User space installation – previously installed |          |          | User space installation |
| BLAS/Lapack      |             |         |          | User space install | Previously inst. | yum install + config. |
| Boost >1.40      |             |         |          | User space install |
| HDF5             | User space installation – previously installed |          |          | System install available |
| MPI              |             |         |          |                         |
| C++/Fort.        | System provides all requirements |          |          | yum install |
| GNU Make         |             |         |          | System install available |
| Autotools        |             |         |          |                         |
| Cmake > 2.8      | User space inst. avail |          |          | User space installation |
Results - performance

- Iteration = one step of simulated time, e.g., 0.01 sec
- RD kernel
- NS kernel

Graph showing performance of different kernels with varying numbers of MPI processes.
Cost of RD test

- Cost of a single time step of the simulation
- `ec2`: Amazon sells instances not cores, so price for 1 and 9 processes is the same (time increases slightly)
- `ec2` mix:
  - Resources in two flavors:
    1. Fully-paid instances
    2. Spot request instances (auction-based price for underutilized resources)

The cost aware `ec2` mix algorithm:
- Instantiates as many spots as possible, remaining instances are full-price (we did not observe any performance degradation for such a mix)

Our department cluster is cheapest; however, the simulation is slow.
Results – cost factors

- Switch platforms to optimize
  - time
  - money

An on-premise cluster is not necessarily the cheapest resource...

- Users may change targets for different problem sizes
Use case 2 – a “weird” experiment

- MPI applications on MR environments
  - Run un-modified MPI codes on Hadoop (Amazon Elastic MR)
  - Why?
    - IaaS ↔ PaaS space has intriguing flexibility
    - Other potential benefits?
- Related projects
  - MPI-MR library (Sandia), Twister (Indiana), MR on MPI (IU/UTK), etc

- Restrict to collective communications
  - Key idea: `emit(mpi_rank, msg content)`
  - Preserve and restore process state across MPI_coll_call implemented as MR shuffle
Job flow

ep-program execution

```c
#include "mpi.h"
...
int main(...){
...
int MPI_Bcast(...)
Write messages
Self-checkpoint & exit
}
...
int MPI_Reduce(...)
Write messages
Self-checkpoint & exit
}
Resume & msgs
Restart from checkpoint
Read messages
}
M chỉnh nț讏Finalize();
return 0;
}
```

Initial data batches

- Start
- Checkpt. & msgs
- Reduce
- Resume & msgs
- Map
- Write messages
- Self-checkpoint & exit

Execution environment, state of the process

Exit

Message passing
Summary

Adaptation is an alternative (or enhancement) to
- Application refactoring, standardization, homogenization
- ADAPT project defines a methodology and framework
- Computing as a utility
  - Very appealing notion
  - Adapters are commonplace in other utilities