GBAC: Virtualization based execution environment for Desktop Grids

Attila Csaba Marosi, József Kovács, Péter Kacsuk
atitisu@sztaki.hu, smith@sztaki.hu, kacsuk@sztaki.hu

MTA SZTAKI
Hungary

http://edgi-project.eu
Start date: 2010-06-01
Duration: 24 months

EDGI is supported by the FP7 Capacities Programme under contract nr RI-261556

Cracow Grid Workshop 2011 – November 7th, 2011, Cracow
Outline

- Introduction
- Problem statement and motivations
  - Available solutions
- Comparison of API’s and tools for application development
- GBAC: Generic BOINC Application Client
- Conclusions and future works
Introduction

- Volunteer Computing (and Desktop Grids) aggregate large number of resources
  - Volatile, non-dedicated, heterogeneity of resources

- **BOINC** – Berkeley Open Infrastructure for Network Computing – is the most popular
  - Individual/ isolated “projects” gather resources to solve their compute-intensive problem

- [http://boincstats.com](http://boincstats.com)
  - 72 different BOINC projects
  - 6,625,750 hosts
  - 2,289,655 users
• Only master-worker / parameter sweep / bag of tasks type applications are suited

• Existing applications need to be “boincified” (ported) – C/ C++/ Fortran/ Python API
  – File access functions need to be changed
  – Recompile and link with BOINC libraries
  – Application binaries for different platforms (e.g., Windows, Linux) should be provided
  – Only application level check-pointing is supported

• Applications must be deployed (“registered”) at the BOINC project
Problem statement – 1. Diverse resources

- BOINC projects aggregate diverse type of resources
  - E.g., Windows or Linux based, 32bit or 64 bit OS
- Scientific applications are usually Unix based, while majority of the resources are Windows based
  - Not always possible to port applications
- Applications should be written to run on all OS versions

- [http://boincstats.com](http://boincstats.com)
  - 6,625,750 hosts total
  - 4,710,478 Windows hosts
  - 1,607,016 Linux hosts
  - 299,096 Mac OS X hosts
Problem statement –
2. Check-pointing

- Volatile, non-dedicated resources
  - Application can be interrupted any time

- Check-pointing is supported only at the application level

- Check-pointing has to be implemented for each application separately
  - Not always possible
  - **Alternative**: units of works ("workunits") should be optimally sized
Problem statement – cont’d

3. Legacy applications
   - Legacy applications: source code not available – re-compilation or re-linking not possible

4. Application deployment (“registration”)
   - Only at the server deployed applications can be used – usually only after a complex validation procedure
   - Applications with complex requirements (E.g., external libraries, specific OS version) are hard to port and deploy

5. Trust: isolation and resource limits
   - Volunteer’s computer runs an executable sent by the project
   - BOINC Client enforces resource limits
Available tools for application development

• Native API
  – BOINC provides a C /C++/ Fortran/ Python API for application development

• “Meta” API
  – A single API for multiple DCI’s
  – Provide API’s both for client and master sides
  – Hides specifics of the middleware
  – E.g., DC-API

• Wrapper technologies
  – For legacy applications
  – E.g., BOINC Wrapper, GenWrapper
Comparison of APIs and technologies

<table>
<thead>
<tr>
<th></th>
<th>BOINC API</th>
<th>DC-API (&quot;Meta API&quot;)</th>
<th>BOINC Wrapper</th>
<th>GenWrapper</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming language</td>
<td>C/C++ /Fortran /Python</td>
<td>C/C++ (/Java, experimental /Python)</td>
<td>No programming, only a job description file</td>
<td>POSIX shell scripting</td>
</tr>
<tr>
<td>Suitable for legacy applications</td>
<td>✓ (difficult implementation)</td>
<td>✓ (difficult implementation)</td>
<td>✓ (with limitations)</td>
<td>✓</td>
</tr>
<tr>
<td>Suitable for non-legacy applications</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Isolation and enforce resource limit</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓ (with limitations)</td>
</tr>
<tr>
<td>Homogenous execution environment (single application version)</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>System / Application level Check-pointing support</td>
<td>✗/✓</td>
<td>✗/✓</td>
<td>✗/✓</td>
<td>✗/✓</td>
</tr>
</tbody>
</table>
GBAC – Generic BOINC Application Client

- Virtualization based - VirtualBox
  - System level check-pointing - snapshots
  - Homogenous execution environment – Virtual Appliances
  - Increased security - Isolation, resource limits enforced automatically, fault tolerant sandbox

- Wrapper – not just for legacy applications
  - Contains all BOINC specific parts, handles communication with BOINC

- Contains a Linux based VA – homogenous execution environment
  - Legacy applications are executed inside a running VM instance
GBAC: Overview

1. Submission Interface and job management (3G Bridge)

DG Server (BOINC)
- Job Database
- Applications
  - GBAC Wrapper
    - Binary for Platform A
    - Binary for Platform B
    - Virtual Appliance

Computing Resource
- DG Client
- VirtualBox

2. DG Client

3. VirtualBox

Cracow Grid Workshop 2011 – November 7th, 2011, Cracow
The EDGeS 3G Bridge

Job Handler Interface

Input

Job Database

Queue Manager

Queue 1
Queue 2
... Queue n

Queue Handler Interface

Output

Grid Handler Interface

BOINC Plug-in

EGEE Plug-in

Condor Plug-in

IaaS Plug-in

EGEE BOINC WebService

...
1. A job is sent to 3G Bridge
2. 3G Bridge has a list of algorithms (applications) registered
   - If the job references a non-existing application (algorithm) a fallback algorithm is used
   - The fallback in this case is the “gbac” algorithm
3. The job is assigned to the given queue and the gbac algorithm
4. The job is submitted to BOINC as a work unit for the “gbac” application
   - The algorithm name in 3G Bridge matches the application name in BOINC
### GBAC: the “wrapper”

<table>
<thead>
<tr>
<th>Application</th>
<th>1. GBAC Wrapper is the main executable of GBAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>wrapper binary</td>
<td>- It handles communication with the BOINC client and the Virtual Machine</td>
</tr>
<tr>
<td>Configuration file</td>
<td>2. Wrapper is configured via the vbox_job.xml file</td>
</tr>
<tr>
<td>Virtual Appliance (Linux 32 bit)</td>
<td>- Operating System name</td>
</tr>
<tr>
<td></td>
<td>- Memory size</td>
</tr>
<tr>
<td></td>
<td>- Network access</td>
</tr>
<tr>
<td></td>
<td>- <em>Host shared directory access</em>: used to transfer files between host and guest</td>
</tr>
<tr>
<td></td>
<td>- Virtual Appliance to use</td>
</tr>
<tr>
<td>Input files</td>
<td>3. Wrapper will start the virtual machine using the supplied Virtual Appliance</td>
</tr>
<tr>
<td>Legacy application binaries (Linux 32 bit)</td>
<td>- <strong>The Virtual Appliance is part of the application and downloaded only once</strong></td>
</tr>
<tr>
<td>Legacy application input files</td>
<td>4. The input files are accessed from the VM by a guest-host shared folder</td>
</tr>
<tr>
<td></td>
<td>- from security perspective this is not the best solution</td>
</tr>
<tr>
<td>Output files</td>
<td>5. Output files are generated in the same directory and picked up by the client after the execution finishes</td>
</tr>
<tr>
<td>Legacy application output files</td>
<td></td>
</tr>
</tbody>
</table>
GBAC: Virtual Appliance

- A single VA: 32bit Linux (Debian)
- Immutable image
  - Single VA for all tasks and applications
  - Each task is executed in its own instance
  - Disk I/O is written in a overlay image which is discarded after use

\begin{center}
\begin{tikzpicture}
  \node (va) [draw] {Virtual Machine 1};
  \node (task) [above of=va] {Task A};
  \node (disk) [below of=va] {Overlay Instance Disk Image};
  \node (base) [below of=disk] {Immutable Base Image};

  \node (vm2) [draw, right of=va] {Virtual Machine 2};
  \node (task2) [above of=vm2] {Task A};
  \node (disk2) [below of=vm2] {Overlay Instance Disk Image};

  \draw [->] (task) -- (va);
  \draw [->] (va) -- (disk);
  \draw [->] (disk) -- (base);
  \draw [->] (task2) -- (vm2);
  \draw [->] (vm2) -- (disk2);
\end{tikzpicture}
\end{center}
## Comparison of APIs and technologies - including GBAC

<table>
<thead>
<tr>
<th></th>
<th>BOINC API (&quot;Meta API&quot;)</th>
<th>DC-API (&quot;Meta API&quot;)</th>
<th>BOINC Wrapper</th>
<th>GenWrapper</th>
<th>GBAC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Programming language</strong></td>
<td>C/C++ /Fortran /Python</td>
<td>C/C++ (/Java, experimental /Python)</td>
<td>No programming, only a job description file</td>
<td>POSIX shell scripting</td>
<td>None required</td>
</tr>
<tr>
<td><strong>Suitable for legacy applications</strong></td>
<td>✓ (difficult implementation)</td>
<td>✓ (difficult implementation)</td>
<td>✓ (with limitations)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Suitable for non-legacy applications</strong></td>
<td>✓</td>
<td>✓</td>
<td>❌</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Isolation and enforce resource limit</strong></td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Homogenous execution environment (single application version)</strong></td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>✓ (with limitations)</td>
<td>✓</td>
</tr>
<tr>
<td><strong>System / Application level Check-pointing support</strong></td>
<td>❌/✓</td>
<td>❌/✓</td>
<td>❌/✓</td>
<td>❌/✓</td>
<td>✓/✓</td>
</tr>
</tbody>
</table>

---

*Cracow Grid Workshop 2011 – November 7th, 2011, Cracow*
Conclusions and future work

- Porting applications to DG’s is sometimes not an easy task
  - E.g., legacy applications, supporting diverse resources

- There are tools to ease this burden
  - There is no “silver bullet”

- GBAC offers through the use of virtualization solution for most of the current challenges
  - Still not a silver bullet

- Future work includes
  - Improved transfer of legacy application files
  - Improved management of Virtual Appliances (more overlays, more base images)
Thank you!

http://edgi-project.eu/