Virtualization for Desktop Grid Clients

Marosi Attila Csaba
atisu@sztaki.hu

BOINC Workshop’09, Barcelona, Spain, 23/10/2009
“Using Virtual Machines in Desktop Grid Clients for Application Sandboxing”

- Joint work with Gilles Fedak (INRIA) and Oleg Lodygensky (IN2P3)
- Carried out in the frame of a CoreGRID Researcher Exchange Programme
- Idea came after the 3rd BOINC Workshop (BOF on virtual machines
- Work was done in November-December 2007
  - 5 weeks total at INRIA, Orsay, France
- A technical report was published with the results in June 2008
- Was a long time ago, but I think it might be still interesting…
Goals 1/2

• “Provide a checkbox in the BOINC Manager which enables the execution of any application inside a Virtual Machine (sandbox).”
  • Usable by any deployed application
  • Should not require to install any additional libraries
  • Should be integrated with the client
  • Should not interfere with the daily work of the user
• “Should be a general solution that can be integrated with different DG middlewares.”
  • In our case at least with XtremWeb (INRIA, IN2P3) and BOINC (SZTAKI)
“The solution should be primary aimed at Volunteer Computing projects.”

- Applications with little or no external dependencies (when possible)
  - To avoid large VM images
- CPU intensive applications
- Most likely Windows hosts, but should run also on Linux and Mac OS X
- These characteristics can be also true for some commercial applications…
  - E.g. the applications of CancerGrid
Benefits - motivation

- Simplified application development
  - A binary for a single platform (preferably Linux) is enough
  - Applications with many dependencies can be run
- Legacy applications
  - Applications without source code can be run on BOINC
- System-level checkpoint
  - VMs can be suspended, checkpointed, resumed
  - No need to implement it at the application level
- Enforce resource limits
- Isolation
Considered virtualization tools

- Bochs
  - Emulator implemented in C++
- QEMU
  - Processor emulator
- KQEMU
  - Extension for QEMU to improve performance
- VMWare Player
- VirtualBox
Requirements for virtualization tools 1/4

• Transparency for the system
  • Should work “out of the box” with already deployed Desktop Grids
  • Should not constrain any restrictions to applications when using the VM
    • Checkpoint and resume, suspend and continue, measure and report the used CPU time and fraction done

• Transparency for the user
  • No special knowledge or preparation should be required for deployment
  • Should not interfere with the daily routine of the volunteer
Requirements for virtualization tools 2/4

- **Isolation**
  - Applications running in the VM should not have any possibility for outside contact
    - Network access, accessing the files on the host, etc.
- **Backdoor**
  - Should be a method for accessing files inside the VM
  - Still no access to outside world for the guest
  - e.g. QEMU allows to forward a port from the guest to a socket at the host (without networking at the guest)
- **Cross-platform**
  - Should run on Windows, Linux, Mac OS X
Requirements for virtualization tools 3/4

- **Instantiation**
  - More than one VM could be running at a time
  - Duplicate images for each VM should be avoided
    - *Using overlay images*
- **Failure-tolerant (“bullet-proof”)**
  - No malicious application or task may render the VM unusable for future tasks
    - Creating and reverting to snapshots
    - *Using overlay images*
- **Performance**
  - Performance penalty for using the VM should be low
Requirements for virtualization tools 4/4

- **Background ("headless execution")**
  - Should not present windows, pop-ups or a graphical display, should run in the background
- **Licensing**
  - Should be open source e.g. GPL, LGPL, BSD, Apache, etc.
## Comparison

<table>
<thead>
<tr>
<th>virtualization method</th>
<th>Bochs</th>
<th>QEMU</th>
<th>QEMU + KQEMU</th>
<th>VMware Player</th>
<th>VirtualBox</th>
</tr>
</thead>
<tbody>
<tr>
<td>transparency for the user</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. deployment</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>b. slowdown</td>
<td>✓</td>
<td>✓</td>
<td>✓ (KQEMU)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>transparency for the system</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. checkpoint and resume</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>b. suspend and continue</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>c. remote control</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>d. backdoor</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>isolation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>cross-platform</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>performance</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>instantiation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>background</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>licensing</td>
<td>LGPL</td>
<td>GPL</td>
<td>GPL</td>
<td>Proprietary</td>
<td>Proprietary/ GPL</td>
</tr>
</tbody>
</table>
## Comparison

<table>
<thead>
<tr>
<th>feature</th>
<th>Bochs</th>
<th>QEMU</th>
<th>QEMU + KQEMU</th>
<th>VMware Player</th>
<th>VirtualBox</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtualization method</td>
<td>emulation</td>
<td>emulation</td>
<td>virtualization</td>
<td>virtualization</td>
<td>virtualization</td>
</tr>
<tr>
<td>1. transparency for the user</td>
<td>✓</td>
<td>✓</td>
<td>× (KQEMU)</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>a. deployment</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>b. slowdown</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2. transparency for the system</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>a. checkpoint and resume</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>b. suspend and continue</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>c. remote control</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>d. backdoor</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>3. isolation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>4. cross-platform</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>5. performance</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>6. instantiation</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>7. background</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>8. licensing</td>
<td>LGPL</td>
<td>GPL</td>
<td>GPL</td>
<td>Proprietary</td>
<td>Proprietary/ GPL</td>
</tr>
</tbody>
</table>
General ideas

* VM images are big – create them on the spot
  * Distribute a base image, and inject the input files on the client
  * Use overlay images for fault tolerance
* Define and use atomic commands for VM control and task execution
  * *libvirt* was considered, too complex, functionality was missing
  * Use an existing protocol, e.g. *http* or *ssh*
    * *http* already has PUT, GET to store and retrieve files
Architecture – with BOINC

BOINC core client

VM API

VM Manager

VM Instance Image

VM Instance (Guest)

Monitor

VM Base Image

Disk I/O

Communication Daemon (HTTP)

Message Handler

Data Handler

Execution Environment

Application
Input Files
Output Files
stdin/ stdout/ stderr/ argv
Architecture – with BOINC

**VM Manager**
- Start
- Create

**VM Instance Image**
- Disk I/O

**VM Instance (Guest)**
- VM create
- VM load
- VM suspend
- VM resume
- VM start
- VM stop
- VM checkpoint
- VM continue
- VM destroy

**Communication Daemon (HTTP)**
- TASK start
- TASK stop
- TASK ping
- TASK get
- TASK put
- TASK delete
- TASK msg_to_host
- TASK msg_to_guest
- TASK ls
- TASK mkdir

**Execution Environment**
- Application
- Input Files
- Output Files
- stdin/ stdout/ stderr/ argv

**VM API**
- BOINC core client

**CancerGrid**
Architecture - VM API

- “High-level” C/C++ API to control task execution and VMs
- `vm_sb_*` functions for task execution
  - start, stop, put, get, ping, delete, msg_to_host, msg_to_guest, ls, mkdir
- *Communication daemon* on the guest side
- `vm_*` functions for VM control
  - create, load, suspend, resume, start, stop, checkpoint, continue, destroy, get_status
- *VM Manager* provides these functions
• “Low-level” component for managing VM images
• Performs operations requested by the VM API
  • create, load, suspend, resume, start, stop, checkpoint, continue, destroy, get_status
• **VM Base Images** store default Linux OS and components that are required to run by the guest OS
• Communications Daemon, *Handler, Execution Environment
• Overlay images are created for **VM instances** – all disk I/O goes here
  • Thrown away after task finishes
• Metadata stored in a SQLite database
Architecture - VM Manager 2/2

- Controls the VM via the monitor of QEMU (concept from libvirt)
  - monitor is bound to a socket
  - works like a terminal
    - send a command
    - if we get a prompt success
- VM Base image
  - Debian Linux
  - Compressed QCOW2 format
  - ~350MB
- Instance image ~50-150MB
Architecture – Communication Daemon

• Handles task specific commands
  • Embedded HTTP server, receives messages from host
    • `start`, `stop`, `put`, `get`, `ping`, `delete`, `msg_to_host`, `msg_to_guest`, `ls`, `mkdir`
  • implemented over HTTP PUT/ GET/ POST

• QEMU allows to forward a TCP port in the guest to a TCP port on the host – modified QEMU to map to a socket on the guest when available

• All communication is initiated by the host, guest only responds
Architecture – Message Handler, Data Handler, Execution Environment

• Execution Environment
  • Starts application in a work directory - can be removed and recreated at the end of each task
  • Environment variables set
  • Command line parameters

• Data Handler, Message Handler
  • Implement functionalities for the available commands
How to Implement BOINC functionalities?

- **Checkpoint/ Resume**
  - Provided by the VM (QEMU)
  - Multiple checkpoints can be stored in a single overlay image

- **Suspend/ Continue**
  - Provided by the VM (QEMU)

- **Measure CPU time**
  - QEMU instance is a single process, we can measure its used CPU time directly (**is not implemented**)

- **Report fraction done**
  - Using `msg_to_host` (**is not implemented**)

- **Enforce resource limits (CPU, disk)**
  - Provided by the VM (QEMU)
Performance – Intrusiveness 1/2

- We wanted to know
  - How big is the CPU overhead of the virtualization (QEMU)
  - How does lowering the priority of the VM instance process affects performance and responsibility of the host system
- We run a test – execute a work unit in the VM while performing daily routine-work on the host
  - Was editing a PowerPoint presentation
  - Each part of the test was run 20 times
  - Application “BinSYS” from SZTAKI Desktop Grid
  - Host: Pentium IV 2.53GHz CPU, 1GB RAM, Windows XP
  - Guest: 160MB RAM, Debian Linux
### Performance – Intrusiveness 2/2

<table>
<thead>
<tr>
<th>Type</th>
<th>Slowest</th>
<th>Fastest</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native Linux</td>
<td>711.06 sec</td>
<td>708.17 sec</td>
<td>710.20 sec</td>
</tr>
<tr>
<td>Windows host, Linux guest, QEMU normal priority, with KQEMU</td>
<td>747.56 sec</td>
<td>744.21 sec</td>
<td>745.12 sec</td>
</tr>
<tr>
<td>Windows host, Linux guest, QEMU below normal priority, with KQEMU</td>
<td>759.76 sec</td>
<td>757.60 sec</td>
<td>758.71 sec</td>
</tr>
</tbody>
</table>

- **“Normal priority”** – Noticeable slowdown in the host, especially when disk I/O
- **“Below normal priority”** – No slowdown
- Without the KQEMU component, the execution was extremely slow
Status and future work

- “Prototype” - It works, but…
  - Parts of the High-level VM API are missing
  - Integration with the BOINC Client is missing
- Part of a proposal for an EU funded project due to start in 2010
- Technical report available at http://boinc.berkeley.edu/trac/wiki/VmApps
Usage in the CancerGrid project

- Workflows are executed
  - Consist of legacy applications using GenWrapper (BOINC)
  - CPU intensive applications
- Consortium of academic and industrial partners
  - Consortium members donate CPU time
  - Office computers running Windows
- Increased security would be more than welcomed by Administrators…
If you need more detailed (technical) information, email to desktopgrid@lpds.sztaki.hu or visit www.desktopgrid.hu

Thank you for your attention!

Questions?

Acknowledgement:
CancerGrid EU FP6 project (FP6-2005-LIFESCTHTALTH-7)
http://www.cancergrid.eu