GenWrapper: A Generic Wrapper for Running Legacy Applications on Desktop Grids

Attila Csaba Marosi
atisu@sztaki.hu

Joint EGEE and EDGeS Summer School on Grid Application Support
29 June 2009 - 07 July 2009
Budapest, Hungary
Outline

• Volunteer Computing and Desktop Grid Computing
  • BOINC – Berkeley Open Infrastructure for Network Computing
    – native applications
    – the BOINC Wrapper
• GenWrapper
  – Motivation
  – Details
• Applications and Projects utilizing GenWrapper
• Conclusion
Volunteer Computing and Desktop Grid Computing

- **Volunteer Computing** usually refers to aggregates formed by non-dedicated (volunteer) desktop nodes
  - *public resource computing, public-based desktop grids, public desktop grids*
  - volatile nature of connected resources
  - the resource donating entity (“**donor**”) needs to trust the entity (“**project**”) gathering the resources

- **Desktop Grid Computing** is using private resources available at institutions and companies
  - *institutional desktop grids, enterprise desktop grids, local desktop grids*
  - administrators have total control over the resources
  - donors may not be aware of donating (idle) CPU time
  - dedicated resources, **different security requirements**
BOINC

- **donors install the BOINC Core Client and attach to a Project**
  - the client downloads an Application and sets of input data ("Work units")
  - the Application processes the input data
  - the client uploads the output ("Completed result")

- **BOINC is suited for Master-Worker style applications**
  - no communication is possible between the "work units"
The native BOINC application

- any application needs special preparation
  - needs to be recompiled and linked with the BOINC library
  - has to call `boinc_init()` at the beginning and `boinc_finish()` before exit
    - the Core Client and the Application uses shared memory for communication
  - for each file to be opened needs to be resolved via `boinc_resolve()`
  - there is a separate working directory ("slot") and storage dir ("project")
The BOINC Wrapper

- **the wrapper is a native application, but:**
  - handles communication with the Core Client (suspends, resumes, starts and kills the application; reports fraction and CPU time used)
- **can be used to port Legacy Applications**
  - no need to change the original code
  - legacy applications are run as sub-processes
- each application may have input, output files, environment and command line
- checkpoints after each finished application (task)
- uses an XML style configuration file for task description
Motivation for a Generic Wrapper

Why did we need it?

- The features of the BOINC Wrapper are not enough
  - patching config files on client machines
  - input files need preparation
  - generating extra messages (log, debug)
  - independent jobs in a single WU (batching)
  - unknown number of output files
  - legacy applications may start processes themselves
  - support for DC-API and BOINC API

- Wanted to be prepared for unknown requirements might be raised by future applications

- We did not want to extend the BOINC Wrapper to make it an XML-based programming language, we choose to use an existing language -> Bourne shell
A Generic Wrapper

• How did we do it?
  – we took GitBox a Windows only port of BusyBox …
  – a single binary providing POSIX shell interpreter and essential UNIX commands (sed, grep, tar, echo, etc)
  – was used earlier by the git version control system on Windows (abandoned now)
  – … and ported it back to Linux and Mac OS X (while still runs on Windows)
  – the name remained GitBox, but has little common with the original…
  – extended it to…
    – use the BOINC API and to provide the API for POSIX shell scripting (boinc resolve_filename, boinc fraction, boinc fraction_percent)
    – have more commands available (like unzip, awk, etc) and fixed some…
    – handle communication with the Core Client: report CPU time; suspend, resume and kill processes started (not trivial!); CPU throttling, etc.

• How does it work?
BOINC Core Client

Launcher
- `boinc_init()`
- `unzip application`
- `generate starter`
- `execute GitBox` `boinc_finish()`

BOINC Client Library

Starter script
- `source profile script`
- `exec work unit script`

Application Bundle (Zip)

GitBox

Profile script
- `Legacy Application 1.`
- `...`
- `Legacy Application N.`

Application

Work unit script
- `perform arbitrary action`
- `exec Legacy Application 1.`
- `boinc_fraction_done 0.5` `perform arbitrary action` `exec Legacy Application N.`
- `boinc_fraction_done 1` `perform arbitrary action` `zip outputs`

Input 1. ... Input N.

Output Output

Work unit

GenWrapper
Sample GenWrapper script

```bash
1. IN=`boinc resolve_filename in`
2. OUT=`boinc resolve_filename out`
3. NUM=`cat ${IN}`
4. PERCENT_PER_ITER=$((100000 / NUM))
5. for i in `seq $NUM`; do
6.   PERCENT_COMPLETE=$((PERCENT_PER_ITER * i / 1000))
7.   boinc fraction_done_percent ${PERCENT_COMPLETE}
8.   echo -e "I am ${PERCENT_COMPLETE}% complete." >> ${OUT}
9.   sleep 1;
10. done
```

- no need to call `boinc_init()` or `boinc_finish()`
  - exit status of the script is the exit status of the work unit
- the script should implement checkpointing, and checkpoint itself when fits
- every input and output file needs to be resolved
- no background jobs yet (Windows lacks `fork()`)
  - but legacy applications may create new processes themselves
Some applications and projects using GenWrapper

- **CancerGrid Project** - *Grid Aided Computer System For Rapid Anti-Cancer Drug Design*
- **EDGeS Project** – *Enabling Desktop Grids for e-Science*
- **3D Video Rendering Service using Blender** @ UoW
- **Protein Molecule Simulation using AutoDock** @ University of Westminster
- **Patient Readmission Application** - statistical model developed in *R* @ UoW
- **Discrete event simulator using Discrete Event Modelling on Simula (Demos)** @ Norwegian University of Science and Technology (NTNU)
- **EMMIL** – *E-commerce model to integrate logistics* @ International Business School Inst. of Information Systems and Logistics, Budapest, Hungary
The CancerGrid project

- EU Framework Program 6 (FP-6, 2006-2009)
- Title: *Grid Aided Computer System For Rapid Anti-Cancer Drug Design*
- Project period
  - January 1, 2007 – December 31, 2009
- Goals:
  - Developing *focused libraries* with a high content of anti-cancer leads, building *models* for predicting various molecule properties
  - Developing a *computer system* based on grid technology, which helps to accelerate and automate the *in silico design* of libraries for drug discovery processes
The CancerGrid applications - Wide variety of applications in a workflow

Applications: **cmol3d, mopac, mdc, fmt, fma, etc.**

- Fortran, C, C++
- processing/ memory requirements
- multi-binary applications (already contain some wrappers), libraries
- **legacy binaries for Linux and Windows**
- config file preparation before execution
- pure logging/ debugging information
- variable run-time
  - one to one Work unit mapping not always efficient
  - **batching**
- variable number of output files
- need to be executed in a specific order
- workflows have been created
  - *molecule descriptor calculator, model building and property prediction*
The CancerGrid applications – descriptor calculator

- the most computation intensive workflow is the descriptor calculator
  - 4 jobs for molecular calculations
  - 2 jobs for file format conversion
  - 3 jobs for database manipulation

- main parameters of the workflow (from computing perspective)
  - $N$: number of two dimensional input molecules
  - $M$: number of confirmers (variants of a molecule)

- molecular calculation jobs are executed once for each input or once for each confirmers
  - typical value for $N : 30,000$; for $M : 100$
  - 3,000,000 instances, total $\sim 10,000,000$ jobs

- the granularity of the workflow is fine grained
  - running time of one instance is a few minutes
  - not suitable for conversion to BOINC work units one to one
The CancerGrid architecture

- a Job Database, Job Queues and Queue Manager extension has been introduced at the BOINC server.
- once a queue contains appropriate number of jobs a work unit is created using DC-API
- a shell script is created to manage the execution of the batch
  - assembled from head, body and tail fragments
  - body part is repeated for each job in the batch
  - may contain macros like `%{name}`
  - executed by GenWrapper
3D Video Rendering Service using *Blender*

University of Westminster, London, UK

- Open source rendering software
- Blender renders frames sequentially
  - Set of frames is sent to workers and a master creates the complete video from the pieces
- Part of an on-line distributed rendering service

http://wgrass.wmin.ac.uk/index.php/Desktop_Grid:Rendering
Conclusions

- GenWrapper offers a generic solution for wrapping and executing an arbitrary set of applications on BOINC (BOINC API or DC-API) and XtremWeb
  - POSIX like scripting language
  - Not a silver bullet!
    - security considerations, legacy applications with lot of external dependencies
- great flexibility and powerful tool for porting legacy applications
- based on a modified version of GitBox (~BusyBox)
  - open source (GPL/ LGPL)
- runs on Windows, Linux, Mac OS X
- small size
  - Launcher and GitBox are ~400KByte each (will be integrated into a single binary)
- many applications and projects are using it
Thank You!

- GenWrapper
  - [http://sanjuro.lpds.sztaki.hu/genwrapper](http://sanjuro.lpds.sztaki.hu/genwrapper)

- Enabling Desktop Grids for e-Science
  - [http://www.edges-grid.eu](http://www.edges-grid.eu)

- CancerGrid
  - [http://www.cancergrid.eu](http://www.cancergrid.eu)

desktopgrid@lpds.sztaki.hu