PROCEEDINGS OF SCIENCE

AppPot: bridging the Grid and Cloud worlds

Riccardo Murri
GC3: Grid Computing Competence Center
University of Zurich
E-mail: riccardo.murri@gmail.com

Sergio Maffioletti*
GC3: Grid Computing Competence Center
University of Zurich
E-mail: sergio.maffioletti@gc3.uzh.ch

This paper presents AppPot, a system for creating Linux software appliances. AppPot appliances can be run as a regular batch or grid job and executed in user space, and require no virtualization support in the infrastructure.

The main design goal of AppPot is to bring the benefits of a virtualization-based IaaS cloud to existing batch-oriented computing infrastructures. In particular, AppPot addresses the application deployment and configuration on large heterogeneous computing infrastructures: users are able to prepare their own customized virtual appliance to provide a safe execution environment for their applications. These appliances can then be executed on virtually any computing infrastructure, being it a private or public cloud, as well as any batch-queueing compute cluster.

We give an overview of AppPot and its features, the technology that makes it possible, and briefly report on experiences running it in production use within the Swiss national grid infrastructure SMSCG.

EGI Community Forum 2012 / EMI Second Technical Conference
26-30 March, 2012
Munich, Germany

*Speaker.

PoS(EGICF12-EMITC2)004

1. Introduction

Application deployment and configuration on large heterogeneous systems is a complex infrastructure issue that requires coordination among system administrators, end users, and grid-level operation teams. This is further complicated when it comes to scientific applications that are, in several important cases, not included in the mainstream Linux distributions.

Virtualized infrastructures and software appliances provide an effective solution to these problems. However, they require a specific infrastructure and a usage model that is markedly different form the batch-oriented processing that is still the backbone of scientific computing.

This paper presents a system (called "AppPot") to bring the benefits of virtualization-based Infrastructure-as-a-Service (IaaS) clouds to existing batch-oriented computing infrastructures.

AppPot comes in the form of a set of POSIX shell scripts that are installed into a GNU/Linux system image, and modify its start-up sequence to allow controlling task execution via the kernel boot command-line. Another utility is provided to invoke an AppPot system image from the shell command line, and request it to execute a given command. Together these features allow the usage of AppPot Virtual Machine (VM) appliances in batch-oriented systems, including grids and local clusters.

The paper is structured as follows: in Section 2, we describe AppPot, its features, and its architecture; in Section 3 we review the goals that guided AppPot development and the problems that we set forth to solve; in Section 4 we report on some real-world experiences and how they relate to the issues outlined in Section 3. Finally, Section 5 recaps the main points of the paper.

2. What is AppPot?

AppPot is the name we give to a software product consisting of an "appliance" (a VM) and a set of support scripts that are installed within the appliance or are used to run it, both interactively and non-interactively.

An AppPot appliance consists basically of just an *AppPot disk image*, which is a complete GNU/Linux system installed in a partition image file (in "raw" format). The AppPot disk image is a regular VM disk image file; it can be used -and has been tested- with the KVM [4], UML [1, 11], and VirtualBox [12] virtualization systems.

When run under the User-Mode Linux (UML) virtualization system, AppPot appliances can be run as regular non-interactive jobs; this requires a few ancillary files:

1. a shell script `apppot-start` used to run a command-line program within the AppPot appliance;
2. an UML Linux kernel;
3. the auxiliary programs `slirp` [8, 5] and `empty` [2] that enable optional features of AppPot (networking, I/O streams redirection).

All these files can, all or in part, be installed system-wide so that many users can benefit from a shared installation.

2.1 Using AppPot

There are three main usage modes of AppPot, detailed below.

Interactive local execution. Users receive an AppPot system image, containing a working installation of Debian GNU/Linux.[1] Users have full access rights to the AppPot system image thus they can modify it by installing new software, libraries, reference data, etc. For instance a user can install her own version of a computational code or a reference dataset that will be used during the computational analysis.

AppPot is then started directly on the users' own computer, using one of the supported VM systems.

Batch job on a cluster resource. In a typical cluster setup, the AppPot appliance disk image file together with a UML kernel are made available to the batch cluster execution node; the `apppot-start` command is invoked by the batch job to run a command non-interactively within the AppPot appliance.

Here we leverage the fact that the UML virtualization system can run in unprivileged mode without any support from the cluster administrator: the disk image file and the UML kernel are the only files which are needed to start a non-interactive UML VM.

Grid job AppPot can also be executed as a grid job on a distributed infrastructure. In this case, the AppPot appliance disk image file, execution script and reference data need to be transferred to the destination node before the execution. This is normally achieved by specifying those input files as part of the grid job description file. Everything else works as in the "batch job on a cluster" scenario.

The virtualized execution model within UML guarantees perfect sandboxing: the user cannot gain no more privileges than the executing user already has.

3. Goals and use cases

The following scenarios are meant as an illustration of AppPot's intended use cases. Throughout the paper, we shall describe how the requirements from these use cases translate into design decisions for AppPot, and how well the goals have been met.

3.1 Deployment of complex applications

Some software packages (notably, many scientific codes) require complex installation procedures. Two commonly-encountered issues are:

1. The application depends on software that is not readily available on the host operating system. This is often the case with many applications developed in-house by research groups, which can only be compiled against a specific version of a support library. A more widely-known example is given by the visualization software *XCrysDen* [6]: it depends on the

[1] The AppPot support software tries to be cross-platform and just rely on the POSIX shell features, so it could run on other Linux flavors as well. No tests have been performed on other Linux distributions, however, as Debian GNU/Linux currently fulfills all of our needs.

Meschach [9] library of linear algebra routines, which is not included in any Linux distribution and rather difficult to compile on one's own as its website is old and unmaintained.

2. The application has a complex or non-standard compilation procedure, and the documentation is scarce. This is, for instance, the case with the GAMESS-US [7, 3] application: the application must be compiled by running an interactive C-shell script that asks a few questions about the system setup and then pre-processes and compiles the source files in a sequence. If the system configuration does not fall among the variants listed in the supplied configuration script, the system administrator has to inspect the script sources, figure out what the values for the internal configuration variables should be, and modify the compilation procedure accordingly.

In a grid infrastructure, there is an additional problem of scale: *all* systems administrators must be conversant with the installation procedures of *every* software piece, and every application must be compatible with all the computing systems available in the infrastructure.

3.2 Running self-developed code

A large fraction of research groups are developing their own software applications; oftentimes for computational experiments that are ephemeral or limited in scope to a local group or niche community.

Leveraging the User-Mode Linux [1] virtualization system, AppPot appliances can run on grid and local clusters as regular batch system jobs, without the need for sysadmin support or root access. This solves both the aforementioned problems:

- AppPot software appliances are a way to implement generic application deployment on a computational grid, and especially to enable users to provide their own software to the computing cluster: a complete AppPot appliance consists of three files, that can be copied to the execution cluster with any available mechanism, including the "stage in" facilities provided by most grid and cluster batch systems.
- Users can use an AppPot VM on their computer for coding, and then run the same VM as a Grid jobs or in a Cloud IaaS infrastructure for larger tests and production runs.

4. Real-world usage

Let's see now how the issues illustrated in Section 3 can be addressed using AppPot.

4.1 Deployment of complex applications

AppPot allows the execution of the appliance through UML; in this way, the appliance contains the complex application as well as its full dependencies and, at the same time, it spares the grid site administrators from executing and certifying the application deployment: all that is needed is to make a certain AppPot VM image and accompanying UML kernel available.[2]

[2] VM-based appliance are a well-known solution to the complex application deployment problem. In this respect, the only novelty introduced by AppPot is the use of UML as the underlying virtualization technology, which makes deployment as easy as copying a few files.

4.2 Running development code

In case the application has a frequent update cycle, to re-deploy the software appliance every time is not a viable option. AppPot provides a snapshot/changes mechanism for this: users can create a "changes" file that encodes the differences of the locally-modified appliance with a "base" one: the "base" disk image is pre-deployed at grid sites, while users keep produce the "changes" files from their own, locally-modified VM. During AppPot boot, the `apppot-init` script will re-create the modified appliance from the base one, by merging in the changes.

This model has been successfully used in the Swiss Multi-Science Computational Grid (SMSCG) infrastructure to support running in-house development code that however depends on a deep stack of pre-requisites (e.g., Python code built upon several layers of statistical and imaging libraries).

4.3 Dynamical expansion of clusters

An AppPot VM appliance can be prepared, that contains a copy of local cluster execution node Operating System (OS); it should be customized by the site admin to connect back to the home site using a Virtual Private Network (VPN). This appliance can then be executed on an accessible external computing infrastructure (e.g., the EGI infrastructure or any flavor of public clouds), which results in a new node being added to the local batch system. With the aid of some control/monitor software, this expansion can be made *dynamic*: one or more "compute node" appliance are started each time the local cluster is overloaded resources. This is similar to the "glide-in" mechanism implemented in the Condor batch execution system [10].

This approach has been successfully used in the Swiss VM-MAD project to dynamically expand a local Sun Grid Engine cluster to support the execution of swarms of proteomics jobs. The Virtual Machines Management and Advanced Deployment (VM-MAD) project is entering its final stages at the time of this writing, so full details will appear elsewhere.

5. Conclusions

AppPot is currently used in production within the Swiss National Grid Infrastructure SMSCG, supporting several use cases like those presented in this paper. We are collecting feedback on the effectiveness of AppPot in large-scale grid computations; we would like to stress that such effectiveness is not just a function of system performance, but should also include consideration of how it makes large-scale computing more accessible (on the users' side) and manageable (on the systems administrators side).

A. List of acronyms

KVM	Kernel-based Virtual Machine
IaaS	Infrastructure-as-a-Service
OS	Operating System
POSIX	Portable Operating System Interface

SMSCG	Swiss Multi-Science Computational Grid
UML	User-Mode Linux
VM	Virtual Machine
VM-MAD	Virtual Machines Management and Advanced Deployment (project ETHZ.7 funded by the SWITCH AAA track)
VPN	Virtual Private Network

References

[1] Jeff Dike. *User Mode Linux*. Prentice Hall, April 2006.

[2] empty - run processes and applications under pseudo-terminal (PTY) sessions. `http://empty.sf.net/`, July 2012.

[3] M. S. Gordon and M. W. Schmidt. Advances in electronic structure theory: GAMESS a decade later. In C. E. Dykstra, G. Frenking, K. S. Kim, and G. E. Scuseria, editors, *Theory and Applications of Computational Chemistry: the first forty years*, pages 1167–1189, Amsterdam, 2005. Elsevier.

[4] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. KVM: the Linux virtual machine monitor. In *Proceedings of the Linux Symposium*, volume 1, pages 225–230, 2007.

[5] J. Knoble. Almost Internet with SLiRP and PPP. *Linux Journal*, 1996(24es):2, 1996.

[6] A. Kokalj. Computer graphics and graphical user interfaces as tools in simulations of matter at the atomic scale. *Comp. Mater. Sci.*, 28:155–168, 2003.

[7] M. W. Schmidt, K. K. Baldridge, J. A. Boatz, S. T. Elbert, M. S. Gordon, J. H. Jensen, S. Koseki, N. Matsunaga, K. A. Nguyen, S. Su, T. L. Windus, M. Dupuis, and J. A. Montgomery. General Atomic and Molecular Electronic Structure System. *J. Comput. Chem.*, 14:1347–1363, 1993.

[8] SLIRP (Wikipedia entry). `http://en.wikipedia.org/wiki/Slirp`. Cited 28 February 2012.

[9] D. Stewart. Meschach: Matrix computations in C. `http://homepage.cs.uiowa.edu/~dstewart/meschach/meschach.html`, 1994.

[10] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The Condor experience. *Concurrency and Computation: Practice and Experience*, 17(2-4):323–356, 2005.

[11] User-Mode Linux website. `http://user-mode-linux.sourceforge.net`.

[12] Oracle vm virtualbox. `http://www.virtualbox.org`, August 2012.