Towards Cross-Platform Cloud Computing*

Magdalena Slawinska, Jaroslaw Slawinski, and Vaidy Sunderam

Department of Mathematics and Computer Science,
Emory University, Atlanta, Georgia, USA
{magg,jaross}@mathcs.emory.edu, vss@emory.edu

Abstract. Cloud computing is becoming increasingly popular and prevalent in many domains. However, there is high variability in the programming models, access methods, and operational aspects of different clouds, diminishing the viability of cloud computing as a true utility. Our ADAPAS project attempts to analyze the commonalities and differences between cloud offerings with a view to determining the extent to which they may be unified. We propose the concept of dynamic adapters supported by runtime systems for environment preconditioning, that help facilitate cross platform deployment of cloud applications. This vision paper outlines the issues involved, and presents preliminary ideas for enhancing the executability of applications on different cloud platforms.

1 Introduction

The past few years have witnessed rapidly growing interest in cloud computing. However, while instant, affordable, access to computing and data resources at dynamically variable levels is valuable, *usability* of clouds for legacy applications and software written in different programming models is low. Moreover, *heterogeneity* in cloud computing platforms often implies inflexibility in the ability to switch between providers, e.g., executing the same application on Microsoft's Windows Azure [7], Google App Engine [18], Amazon EC2 [21], Open Cirrus [5], etc is a formidable challenge.

In this paper we describe our proposed approach to enhance executability of applications on a variety of platforms, in particular, on multiple current and emerging clouds. We outline our project, titled *Adaptive Application Platform Executive System* (ADAPAS) that proposes abstractions for the *dynamic adaptability* of computing paradigms to specific resources. The goals of our project are: (1) *increasing flexibility* in user's choices with respect to target cloud platforms, and (2) *facilitating development* of cloud-aware applications by unifying interfaces and provisioning runtime support.

We believe that at least for certain classes of applications and target platforms, defining a unified capability interface, and subsequent application-to-platform mapping is possible, and that such flexibility will be of value. We note that the

* This work is partially supported by US National Science Foundation grants CNS-0720761 and OCI-1124418.

M. Alexander et al. (Eds.): Euro-Par 2011 Workshops, Part I, LNCS 7155, pp. 5–14, 2012.

eventual goal of *completely portable executability* may prove intractable, or at least be susceptible to some performance degradation. Nevertheless, these efforts will help evolve cloud computing frameworks, identify opportunities for unification, and may result in minimizing the porting effort across cloud platforms.

2 Background and Context

The rapid growth of cloud computing and *aaS resources has been accompanied by increased heterogeneity in platform types and resource access methods. The vision of "computing as another utility" is subsequently both expanded and diminished – the former as users scale out applications beyond on-premises resources and the latter as they try to reconcile programming paradigms with specific cloud facilities. Several projects have attempted to address these issues in different ways.

In order to scale an application out, one method is to provide homogenization at the access level and application paradigm level. There are several efforts to approach access homogenization by formal and informal standardization efforts such as Simple Cloud API [4], EUCALYPTUS [15], Nimbus [11], AppScale [8]. In order to support homogenization at the paradigm level, (1) resource providers offer specialized cloud services such as Amazon Elastic MapReduce [1], Tashi [13], or (2) users may adapt an resource to a required specialization level via conditioning, i.e., installing relevant middleware layers, e.g., ElasticWolf [19], Unibus [20], Nimbus [11].

Another aspect of application deployment is providing missing dependencies. This may be achieved by providing environments with encapsulated application's dependencies, e.g., statically linked executables, creating application bundles with all dependencies (PortableApps [2]), or preparing specialized images with preinstalled software dependencies (rPath, rBuild [3]). There are also projects that, instead of the homogenization approach, propose new application frameworks that provide their own programming models (e.g., CometCloud [12], Aneka [22]). In our project, we propose a novel approach based on (1) the *virtualized execution model* and (2) providing application's dependencies via an *adapter* layer.

3 Proposed Abstractions

As presented in Figure 1, ADAPAS aims to enhance the *executability* of applications to allow their execution on different back-ends, without the need for explicit porting effort. Initially, we intend to concentrate on scientific applications at the application side, and PaaS and IaaS cloud resources with primary focus on PaaS, specifically the Azure platform and GAE, as back-ends. Our approach, however, is intended to be applicable to any other classes of resources including grids and multiple cloud offerings. In the following paragraphs, we present the initial design of the ADAPAS framework.

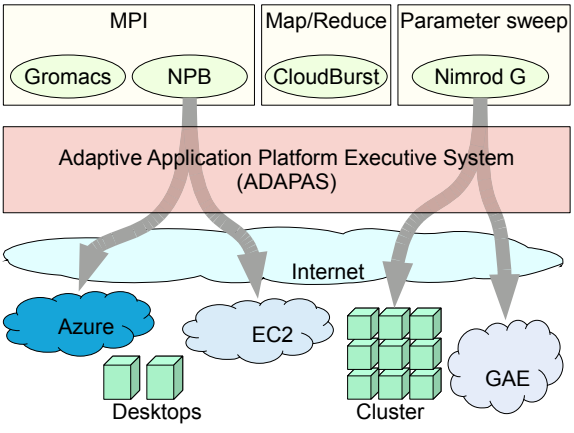

Fig. 1. The Adaptive Application Platform Executive System (ADAPAS) idea

3.1 ADAPAS Overview

Figure 2 depicts a system overview of ADAPAS. The core component of the system is a *Virtual Execution Platform* (VEP). The VEP contains a context layer that comprises *contexts* corresponding to execution environments for a given application, e.g., the MPI context corresponds to MPI applications. In order to execute an application, the VEP provides *adapters* on the application side, and *environment conditioners* on the resource side to provide the required matching.

3.2 Application Model

We assume that an application consists of one or more *programs* which are basic execution units. For instance, an MPI application may consist of one program that is instantiated in many copies. The user may have access to the application's source codes or/and precompiled binary files. The application may have *dependencies* such as binary compatible specific software packages, preinstalled runtimes, dynamically linked libraries, third-party services, etc. In the proposed approach, applications should not require modifications at the source code level in order to execute on specific computational back-ends. Initially, we focus on scientific applications based on established programming paradigms (e.g. SPMD message passing codes, parameter sweep applications, partitioned global address space applications, and workflows) as well as selected libraries and frameworks (e.g., MPICH2 [10], Co-Array Fortran [16], and HADOOP Map/Reduce [23]).

3.3 Resource Model

We model computational resources as *resource chunks* and *computing chunks*, as shown in Figure 3. Resource chunks are the smallest resource allocation units

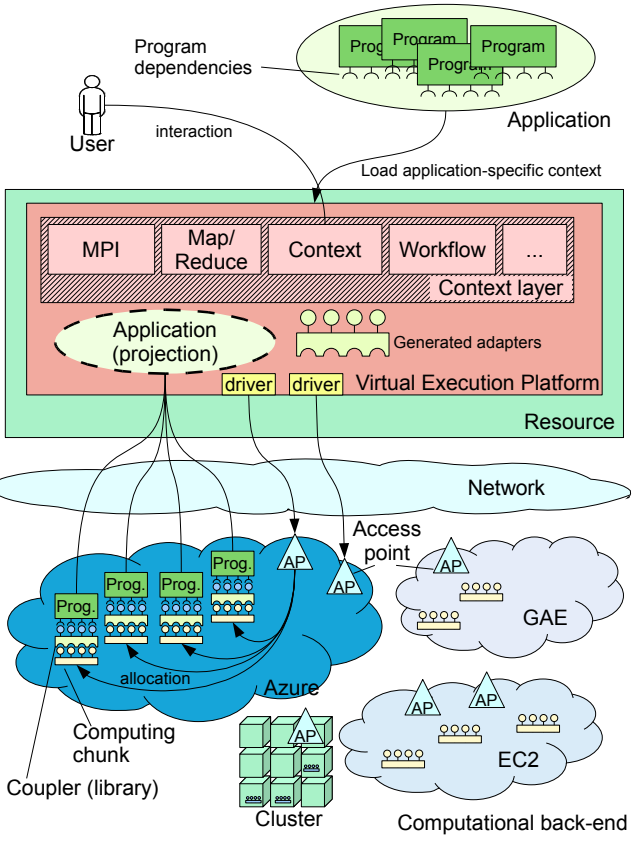

Fig. 2. The ADAPAS system overview

that can be *allocated* to perform the user's computational task on a given computational resource, e.g., cluster nodes on a local cluster, Amazon EC2 quad-core instances, a pluglet container in H2O [14]. A computing chunk is the smallest logical processing unit on the provided resource chunk that is capable of hosting an execution unit. For instance, an executing program, manifested as a process is a computing chunk in a typical timeshared Unix system. On Windows Azure, a computing chunk is a role (worker or web) since a role has the capability to run a program. Programs can be executed only if their dependencies can be expressed in terms of computing chunk capabilities, e.g., the x86 instruction set, a mathematic library, a connection to a database, supported services, etc.

3.4 Application Execution Model

ADAPAS provides a toolkit, called a *Virtual Execution Platform* (VEP), to manage application execution. VEP provides a unified execution model, specialized on-demand by dedicated and pluggable "execution contexts." Contexts support

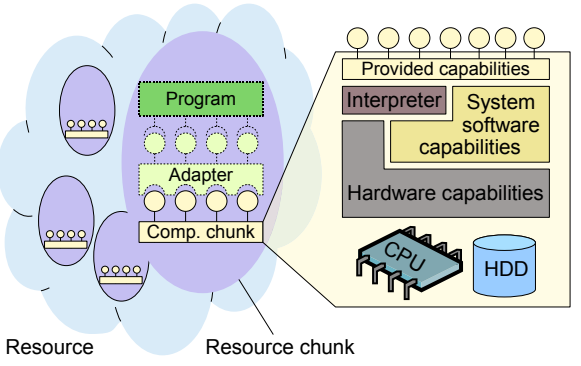

Fig. 3. Resources, resource chunks, and computing chunks

the user with a familiar, specific application-oriented execution environment (e.g., MPI, Map/Reduce, parameter sweep applications, workflow). Contexts will be loaded for a single application run and unloaded when the execution finishes.

The VEP is responsible for all execution stages: (1) obtaining an application (source codes, or binaries) from the client site or a repository, (2) program compilation against target-specific back-end resources (if application source codes are available), (3) resource chunk allocation to enable instantiation of computing chunks, (4) preparing and adaptation of the appropriate libraries required by the program to execute on a particular computing chunk, (5) staging-in the program with all adapted dependencies to the computing chunks, (6) coordinating all programs in order to execute the application, (7) presenting the application status to the application's current context, and (8) releasing resource chunks at the end of execution.

4 ADAPAS Design Approach

Since the goal of ADAPAS is to enhance executability of any unmodified application on any resource, the VEP needs to provide a unifying abstraction, capable of expressing computing capabilities for any computational back-end. To address this, we propose a Virtually Unified Capabilities (VUC) interface that will define common capabilities needed to execute a program. Second, resource capabilities need to meet program dependencies. We approach this through Dependency-Capability objects (DC objects), adapter middleware (adapters), and the VEP engine. The VEP engine will (1) identify program dependencies and discover resource capabilities, and (2) match requested dependencies with offered capabilities via the adapter and DC objects. Third, there is a need for mechanisms to provide missing program dependencies on a computing chunk to enable program execution. This is addressed by the VEP engine and DC objects with *callbacks*.

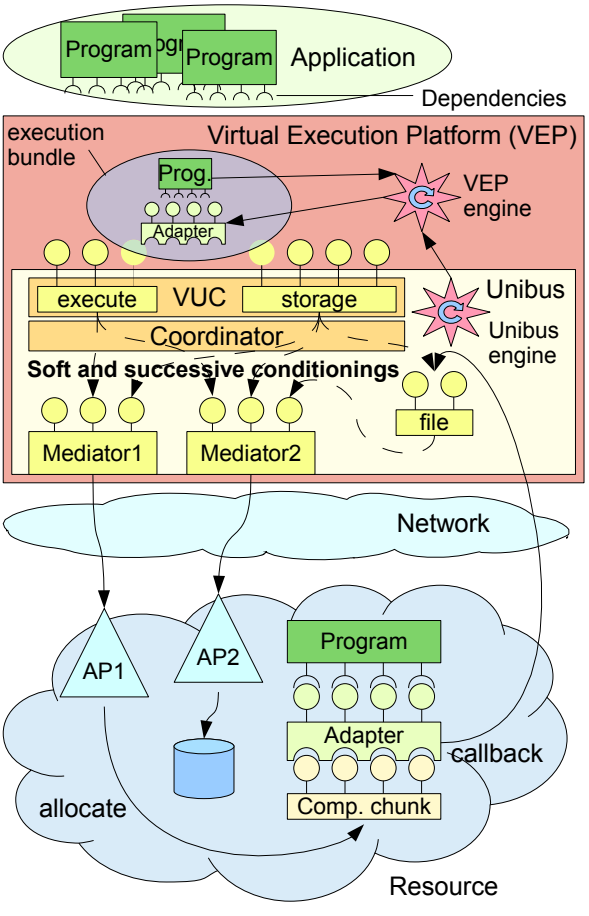

Fig. 4. Virtually Unified Resource Interface

Fourth, is the matter of providing a familiar interface to the user. We address this by proposing VEP-pluggable contexts that are specific for execution environments (e.g., MPI, Hadoop). These aspects are explored below in more detail.

4.1 Virtually Unified Capabilities

VUC capabilities are divided into two interfaces: *execution* and *storage*. The execution interface provides methods related to program execution: (1) sending an *execution bundle* to an assigned computing chunk, (2) starting the program, and (3) monitoring program execution. The execution bundle (bundle) is a complete set of files that are *necessary and sufficient* to execute the program: binary files and program dependencies. Storage interface methods are related to data management: staging-in and retrieving data.

As shown in Figure 4, VUC interfaces are built upon the *coordination layer* (*coordinator*). The coordinator is responsible for four coordination tasks: (1) resource chunk allocation, (2) assigning computing chunks to resource chunks, (3) releasing resource chunks, and (4) managing data transfer routes. For instance, in order to utilize computational power in a more efficient manner, the coordinator may assign four computing chunks to a quad-core CPU resource chunk. To improve network bandwidth, the coordinator may create a resource-local cache to reduce data transfer over the Internet. The coordinator may also aggregate different resources in order to (1) enhance offered capabilities (e.g., combining computing and storage resources to obtain computing *and* storage interfaces), or (2) provide specific features (e.g., fault-tolerant mechanisms, addressing security issues).

4.2 Dependency-Capability Objects

In order to enable the execution of a program, all program dependencies needs to be met on a target computing chunk. In a few cases the only dependency is a binary-compatible program executable. Usually, the program requires a middleware layer that allows the program to run on selected resources. Typically, a resource can be specialized to a required specialization level by soft conditioning (installing missing software dependencies). For a new class of resources such as *aaS, a resource specialization process needs to be augmented; we intend to do this via *dependency-capability objects* (DC objects).

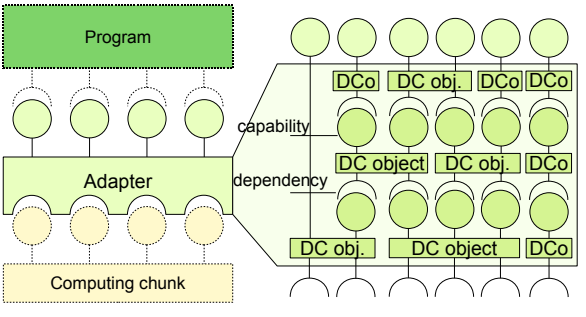

Fig. 5. DC objects create the adapter middleware

DC objects can be characterized as *software functions* or *transformations* that utilize underlying capabilities in order to provide new, more abstract capabilities that will satisfy program dependencies. Every DC object also has its own well defined dependencies as well as capabilities it offers, as presented in Figure 5. A DC object can invoke chunk capabilities, or other DC object capabilities.

We envision that DC objects may be *stateful* and *semantically-aware*. For instance, a DC object implementing the file interface may need to store handles to files as well as the file content. A semantically-aware DC object will be able

to parse the passed parameters and accordingly specialize the invocation. For instance, if a program attempts to connect to a specific port, a DC object implementing the "connect" operation may infer that, in fact, the program requires a connection to a database. The connect operation may involve a series of operations such as re-routing the connection, connecting to a database, or translating queries (if the query is incompatible with the queried database), etc.

4.3 Adapters

We use adapters to address the problem of missing dependencies on a particular computing chunk for a specific application. The adapter, dynamically created or downloaded from a repository by the VEP, serves as a situation-specific middleware layer between application requirements, and a concrete computing chunk. In order to start an application program, the VEP will dynamically create the relevant adapter, attach it to the program bundle and deploy the bundle on a computing chunk.

We implement adapters as libraries containing DC objects that will provide functionally equivalent replacements invoked by original programs. Programs can be distributed as dynamically linked executable files or source codes. In the latter case, equivalent function replacement can be directly incorporated to the programs via static linking at the build stage. From a practical point of view, there are a few options to investigate on how to apply an adapter to a program. Static linking is the easiest method to "inject" dependencies into the program executables. Providing dynamic libraries may be possible by the DLL injection technique (LD_PRELOAD, LD_LIBRARY_PATH, ld.so.conf, *.local, etc). When appropriate, interception of system calls can be employed (e.g., UMview [9], ptrace [17], process monitors).

4.4 Contexts

Contexts are user interfaces to applications executed by the ADAPAS. They will support particular execution environments rather than a particular program. Technically, contexts may be implemented as VEP plugins that are automatically loaded for a particular application run. An example scenario of a context usage is presented below:

```
user@comp> vep target=azure app=npb.tar.gz ctx=mpi
VEP 1.0
mpi context loaded
vep:mpi_ctx>>> mpiexec -np 16 MG.B.16 > results.out
```

In the scenario above, the user starts the VEP and specifies, as input parameters, a target platform, an application to run, and an execution context. Next, the VEP loads the application to be executed, and also the specified execution context. This enables the user to interact with the application execution environment in a familiar manner. In the presented scenario, the context provides the

mpiexec command that executes the MG benchmark [6]. The context command cooperates with the VEP in order to transparently compile and start a program and deliver output results. ADAPAS plans to provide a several standard contexts such as MPI, Map/Reduce, etc.

5 Summary

Cloud computing has been under a growing spotlight as a viable solution for providing a flexible, on-demand computing infrastructure. We propose an adaptive virtual platform to enable "completely portable executability" across different applications programming paradigms and resources, specifically cloud computational back-ends. We approach this by proposing the Virtual Execution Platform that will provide mechanisms to automatically and dynamically adapt programming paradigms to resources or resources to programming paradigms. We also propose the Virtually Unified Capabilities interface that will facilitate development of cloud-aware applications and interoperability across platforms.

References

1. Amazon Elastic MapReduce (2011), `http://aws.amazon.com/elasticmapreduce/`
2. PortableApps.com web page (2011), `http://portableapps.com/`
3. rPath Documentation (2011), `http://docs.rpath.com/`
4. The Simple Cloud API (2011), `http://www.simplecloudapi.org/`
5. Avetisyan, A.I., Campbell, R., Gupta, I., Heath, M.T., Ko, S.Y., Ganger, G.R., Kozuch, M.A., O'Hallaron, D., Kunze, M., Kwan, T.T., Lai, K., Lyons, M., Milojicic, D.S., Lee, H.Y., Soh, Y.C., Ming, N.K., Luke, J.-Y., Namgoong, H.: Open Cirrus: A Global Cloud Computing Testbed. Computer 43, 35–43 (2010)
6. Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Frederickson, P., Lasinski, T., Schreiber, R., et al.: The NAS Parallel Benchmarks. International Journal of HPC Apps 5(3), 63 (1991)
7. Chappell, D.: Introducing Windows Azure. DavidChappell & Associates (December 2009), Sponsored by Microsoft Corporation
8. Chohan, N., Bunch, C., Pang, S., Krintz, C., Mostafa, N., Soman, S., Wolski, R.: AppScale Design and Implementation. Technical report, UCSB Technical Report Number 2009 (2009)
9. Gardenghi, L., Goldweber, M., Davoli, R.: View-OS: A New Unifying Approach Against the Global View Assumption. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008, Part I. LNCS, vol. 5101, pp. 287–296. Springer, Heidelberg (2008)
10. Gropp, W.D.: MPICH2: A New Start for MPI Implementations. In: Kranzlmüller, D., Kacsuk, P., Dongarra, J., Volkert, J. (eds.) PVM/MPI 2002. LNCS, vol. 2474, p. 7. Springer, Heidelberg (2002)
11. Keahey, K., Figueiredo, R., Fortes, J., Freeman, T., Tsugawa, M.: Science Clouds: Early Experiences in Cloud Computing for Scientific Applications. In: Cloud Computing and Its Application (CCA 2008) (October 2008)
12. Kim, H., el Khamra, Y., Jha, S., Parashar, M.: An autonomic approach to integrated hpc grid and cloud usage. In: Fifth IEEE International Conference on e-Science 2009, pp. 366–373. IEEE (2009)

13. Kozuch, M., Ryan, M., Gass, R., Schlosser, S., O'Hallaron, D., Cipar, J., Krevat, E., López, J., Stroucken, M., Ganger, G.: Tashi: location-aware cluster management. In: Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds, pp. 43–48. ACM (2009)
14. Kurzyniec, D., Sunderam, V.: Combining FT-MPI with H2O: Fault-tolerant MPI across administrative boundaries. In: Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium, pp. 120a–120a (2005)
15. Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D.: The Eucalyptus Open-source Cloud-computing System. In: 9th IEEE International Symposium on Cluster Computing and the Grid, Shanghai, China (2009)
16. Reid, J., Numrich, R.W.: Co-arrays in the next Fortran Standard. Sci. Program 15(1), 9–26 (2007)
17. Rochkind, M.J.: Advanced UNIX programming. Prentice-Hall, Inc., Upper Saddle River (1985)
18. Severance, C.: Using Google App Engine. O'Reilly Media (May 2009)
19. Skomoroch, P.: MPI Cluster Programming with Python and Amazon EC2. In: PyCon 2008, Chicago (2008)
20. Slawinski, J., Slawinska, M., Sunderam, V.: The Unibus Approach to Provisioning Software Applications on Diverse Computing Resources. In: International Conference On High Performance Computing, 3rd International Workshop on Service Oriented Computing (December 2009)
21. Varia, J.: Cloud architectures. Technical report, Amazon Web Services, White Paper (2008)
22. Vecchiola, C., Pandey, S., Buyya, R.: High-performance cloud computing: A view of scientific applications. In: 10th International Symposium on Pervasive Systems, Algorithms, and Networks, pp. 4–16. IEEE (2009)
23. White, T.: Hadoop: The Definitive Guide. O'Reilly Media (May 2009)