2011 IEEE International Conference on High Performance Computing and Communications

Extending executability of applications on varied target platforms

Julien Bourgeois*, Vaidy Sunderam*, Jaroslaw Slawinski*, Bogdan Cornea†
*Emory University, 201 Dowman Drive, Atlanta, Ga. 30322, USA
Email: {julien.bourgeois, vss, jslawin}@emory.edu
†University of Franche-Comte, LIFC, 1 Cours Leprince-Ringuet, Montbeliard, 25201, France
Email: bogdan.cornea@univ-fcomte.fr

***Abstract*—High-performance applications are often developed for a specific class of target platforms and executing them on the increasing variety of Cloud and grid environments requires substantial adjustments and reconciliation. The aim of our framework, called ADAPT (Adaptive Application and Platform Translation), is to allow adaptation of applications for execution on various computational resources. The overall objective is to enhance usability of cyber-infrastructure platforms by providing automated adaptations of applications and conditioning of target environments so that greater cross-utilization is achieved. This paper presents a proof-of-concept experiment of a possible use of ADAPT, *viz.* executing a C/MPI application originally written for clusters on the Microsoft Azure infrastructure. This MPI application-to-cloud adjustment is based on automatic identification of the application programming paradigm followed by applying an application transformation to enable execution on an alternative target.**

***Keywords*-HPC, adaptation, cloud computing, MPI, Azure**

I. Introduction

The ability to access information processing, storage, and transformation capabilities over the Internet is expanding the repertoire of possible computing platforms in every application domain. With the increasing maturation of grids and especially clouds, the vision of *computing as a utility* is starting to become a reality. However, the *usability* of such cyber-infrastructure platforms for efficiently executing applications often proves to be a challenge, particularly in science and engineering. These applications are often able to utilize resources that span local and campus facilities, those accessed via virtual organizations, and on-demand Cloud offerings for data and compute-intensive processing. But in order to do so, they often need adjustments and reconciliations specific to each target platform – that pose considerable logistical obstacles to usage of the best resource in a given instance.

We therefore suggest that there is a need for methodologies and frameworks that allow adaptation of applications for execution on the most appropriate resource. We propose one such framework called ADAPT (Adaptive Application and Platform Translation), whose objective is to enhance usability of cyber-infrastructure platforms. The key concepts being investigated experimentally in ADAPT are: (1) a virtual execution platform that presents abstractions of application requirements and resource capabilities; (2) determination of adjustments that are required for the application-platform combination for a given run; (3) assembly of adapters that provide different levels of matching ranging from simple substitutions to alternative libraries and outsourced execution; and (4) provisioning environmental infrastructure on the target platform to ensure that all execution-dependent needs are staged and deployed. This approach is likely viable for several common classes of applications, and for several cyber-infrastructure platforms including common Infrastructure as a Service (IaaS) and some Platform as a Service (PaaS) clouds.

This paper presents a proof-of-concept exercise of the possibilities of ADAPT through an example, exemplifying identification of program characteristics and source-to-source program translation. The idea is the following. An application is capable of executing on a limited number of target platforms. By identifying program characteristics, ADAPT will be able to extend the potential execution platforms without requesting information from the user. For example, the proposed test case uses a C/MPI master-worker application that is usually executed on a cluster but that could also be executed in a cloud environment, e.g. in an Azure environment. However, to do this, it is necessary to first detect that the application is a C/MPI master-worker application and then to adapt this application to the target architecture. Identification of programming paradigms will be able to extract from the application its characteristics, i.e. master-worker paradigm with MPI, so that it can be determined if adaptation to execute on candidate targets, e.g. PaaS clouds is possible. A further step is to perform a source-to-source translation of the application so that it can be executed directly on the target platform. These steps will be greatly aided by other tools; in our paper we describe the use of the ROSE compiler framework to parse the application source code, to analyze it and to modify it with source-to-source translation.

978-0-7695-4538-7/11 $26.00 © 2011 IEEE
DOI 10.1109/HPCC.2011.41

II. Related work

In order to enable portable application execution, one approach is to provide homogenization at the access level and application paradigm level. In the grid space, methods and techniques put forth by the Globus [1] project are canonical examples of standardization. In the cloud domain, there are several efforts aimed at accessing homogenization through formal and informal standardization efforts such as Simple Cloud API [2], Open Cloud Manifesto [3], EUCALYPTUS [4], Nimbus [5], and AppScale [6].

In order to support homogenization at the paradigm level, either (1) resource providers offer specialized cloud services such as Amazon Elastic MapReduce [7], Tashi [8], or (2) users may adapt a resource to a required specialization level via conditioning, i.e., installing relevant middleware layers, e.g., ElasticWolf [9], Unibus [10], Nimbus [5]. There are also projects which, instead of the homogenization approach, propose new application frameworks that provide their own programming models (e.g., CometCloud [11] and Aneka [12]). Our current ADAPT project adopts a somewhat different philosophy, and uses a blend of virtualization and middleware adapters to assist with cross-target execution. This approach permits specific features of resources to be fully exploited, while permitting sufficient flexibility in matching applications to the best suited resource for a given run.

The complementary aspect of facilitating application execution is to ensure that the selected target contains all the needed dependencies, libraries, and runtime systems. In IaaS clouds such as Amazon EC2 [13] or Rackspace [14] one could argue that virtualization addresses these issues i.e. an image containing all the needs of the application could be constructed. However, this approach is not ideal when some requirements vary from run to run (e.g. the use of beta vs. production libraries), when switching between providers. These drawbacks apply also to projects that statically link executables, creating application bundles with all needed libraries (PortableApps [15]), or those that prepare specialized images with preinstalled software dependencies (rPath, rBuild [16]). Furthermore, such approaches are inapplicable in shared resource or grid environments that have to be prepared manually.

In this project we leverage our past and ongoing work on provisioning application requirements through environment conditioning and software-assisted staging of executables, libraries, and other dependencies. We believe that our research on a virtualized execution platform realized via middleware adapters, and preparing target resources through conditioning will help reduce logistical and operational burdens on both providers and users.

Extracting characteristics from an application uses several aspects of compilation and can use the same methods as code clone detection tools. Finding clones or similar characteristics is simpler when the source code is available but it is even possible to find code clone from binary executables [17] which broadens the possibilities of analysis. However, the major code clone detection tools rely on source code analysis. They can be roughly classified in four families [18]: textual [19], lexical [20], syntactic [21], [22], and semantic [23] according to the method used for detecting clones. According to [18], the syntactic approaches, and more precisely, tree-based matching approaches that use a parser to convert source programs into parse trees or abstract syntax trees (ASTs), obtain better scores in their ranking. Our method could be compared with this scheme as we are also converting source in intermediate representation like AST.

However, although these tools show good results for fine-grained detection, our concern in paradigm-mapping is more on coarse-grain detection, which means that we want to find conceptual similarities rather identical line codes. For example, to detect if a program is based on a master-slave paradigm, it is possible to use code clone detection techniques between the target program and one or several generic master-slave programs. But since the commonalities between these codes are the communication structures rather than their codes (which can be completely different depending on the application), the similarities will be too difficult to detect by a code clone detection tool. Another solution would be to use a higher-level concept [24] but these methods, based on semantic analysis, do not provide support for detecting communications patterns either.

III. ADAPT

The key premise of ADAPT is that several classes of applications can be executed on multiple types of computational back-ends with the assistance of a flexible and adaptive middleware environment. The conceptual goals of ADAPT are to enable an application to use: (1) multiple resources and resource classes for a given run; and/or (2) different types of resources for different executions. The ADAPT framework dynamically provides *matching adapters* and performs target platform *environment conditioning* as needed to enable this flexible use of multifaceted resources. For example, MPI applications such as Gromacs [25] or NPB [26] typically run on clusters managed by a batch scheduler, but can be also executed on workstation networks or IaaS clouds [9], [27]. In the latter situations, ADAPT assists with the required provisioning needed to prepare the target environment, but may also supply adapters e.g. command-name replacements.

Moving to somewhat more sophisticated scenarios, script-based applications written assuming a certain execution environment (specific libraries, data access methods, OS version) can be adapted for other targets through the use of wrappers, software packages offering equivalent functionality, and other similar transformations. “Adapters” to enable such matchings can be assembled from repositories or even dynamically generated and deployed automatically, thereby facilitating execution on a compatible target platform.

In an even more complex scenarios, it is possible to imagine paradigm transformation, e.g., executing an iterative MapReduce application as an interacting set of MPI processes [28]. At the extreme end of the scale, the entire functionality of an application can be realized by an equivalent “outsourced” substitute that executes on an available (or best suited) platform, or is simply delivered via a Software as a Service (SaaS) cloud. The ADAPT framework will analyze the needed adaptations, matching application needs to resource capabilities, and evolve a suite of adapters to enable cross-platform execution.

The goal of the ADAPT project is pragmatic, and aims to enable application execution without excluding resource types to the maximum extent possible, and relieve *both users and resource providers* from burdens of application porting as well as resource provisioning and coordination. To achieve this ADAPT intends to offer complementary mechanisms for (1) provisioning of system software on the resource side and (2) mapping of program needs on the application side. This two-pronged approach has the potential to improve executability of unmodified applications on raw (unconditioned) resources, even if the resource presents a non-obvious, non-standard execution back-end for this application. To achieve this, ADAPT will support *dynamic* and *automatic pre-conditioning* of resources together with provisioning *adapters* for applications. Examples for the former include automatically meeting software dependencies such as libraries or run-time systems needed to execute an application on user-selected computational back-ends. Stubs or macros for parameter matching, library interceptors, and callback based translations are examples of the latter.

A. Operational Use Case

The example shown in figure 1 is an MPI application that uses data from a relational database [29]. The user interacts with ADAPT and specifies: an application profile, references to target resources and the execution context, e.g., MPICH, while ADAPT through the identification of program characteristics, automatically defines the main characteristics of the application. Scenario A shows a process that includes substituting

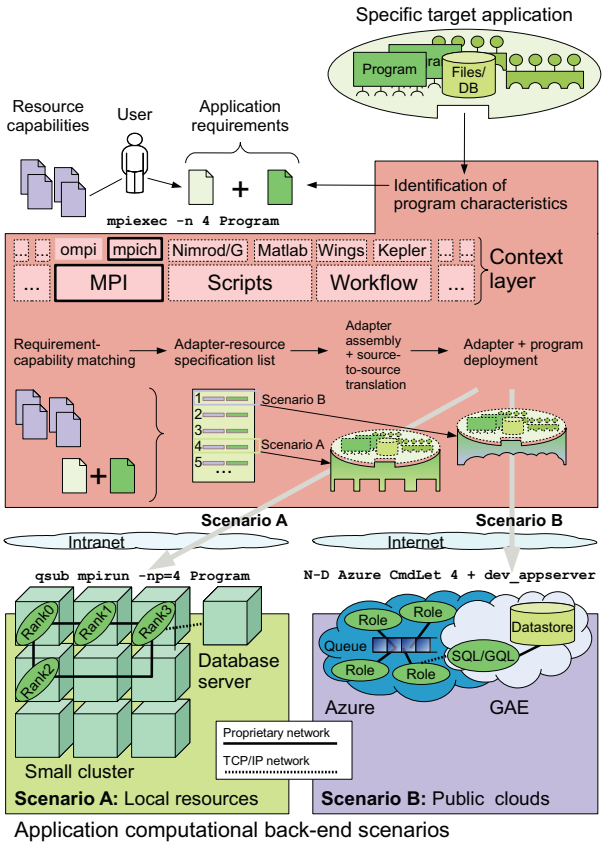

Figure 1: Operational use case

MPICH commands (e.g., `mpiexec -n 32 app_name param1 param2`) by queue submission equivalents and supplying database server parameters. For the parallel component of the application, ADAPT (1) determines application requirements; (2) reports missing capabilities (e.g., numerical libraries, a SQL server), and (3) generates a situation-specific adapter. To execute the application, ADAPT (4) establishes a session connection to database servers, (5) stages the database files, and (6) submits the MPI application with its adapter. Scenario B (inspired by Windows Azure Native Code and P/Invoke features [30]) is more involved and may seem far-fetched – but we outline a plausible, if somewhat contrived, implementation as an extreme example. In this scenario, ADAPT (1) performs source-to-source translation on the application in order to to express communications in terms of native Azure mechanisms (Queues, Blobs) as suggested in [31], [32]. For the database component, ADAPT (2) stages a GEA program that works on the GEA Datastore and processes SQL queries using GQL [33] and (3) configures the redirection of database invocations to the GEA program. To execute the application, ADAPT (4) populates GEA Datastore, and, finally, (5) launches Azure roles

manifesting MPI processes. In either scenario, during runtime, users may issue MPICH-like commands such as `mpdtrace` or `mpdallexit` regardless of the specific MPI implementation used; ADAPT adapters perform any needed translations.

IV. Identification of program characteristics

The example shown in figure 1 shows a more detailed view of ADAPT with the identification of program characteristics. The idea is to be able to build profiles that describe the requirements of the different cyber-infrastructure platforms, to extract from the source code of an application its characteristics and then to match them with the requirements of the candidate execution platforms. If there is a match, the next step will be to perform a source-to-source translation of the application so that it can be executed directly on the target platform.

A. Characteristics specification

We consider two dimensions characterizing application deployment. The first scope relates to the phases the application programs undergo: developing, build, stage-in, execution, and runtime. The analyses of the deliverables from these phases are the source of the *application profile* characteristics that describe the application requirements and allow for the application-platform adaptation. Application assembly constitutes the other characteristics range – in few cases scientific applications consist of a single program to execute. Typically, a scientific application combines several executables (available as source codes and/or binary packages, different target platform and programming paradigms), their delivered dependencies (customized or specific libraries and third parties' software packages), and data (formats, access, source, configurations, etc). The analysis of relations among application components enhances the application profile with a holistic view on the application execution and behavior.

The examination of the programs' source code delivers meta-information about the application. The simplest outcome exposed by the tool would be requirements resulting from the API invocations' analysis (libraries, external software). However the source code inspection has potential to discover and classify semantic-full dependencies such as communication patterns (e.g., loosely/tight coupled, processes arrangements), classes of algorithms, typical data structures, motives of the resources use (e.g., using the hard-drive just for swapping). Such meta-data is crucial to plan the adaptation coupling the application with the prospective targets.

The next phase, build, provides the further refinement. This stage frequently employs well established build frameworks, such as GNU Autotools, makefiles, or shell scripts [34]. The limited syntax of the build systems makes the meta-data extraction particularly straightforward. This information may be auxiliary for the source code processing and provide the additional conditions: flavors and versions of the required libraries, specific/ default programs parameters, build suit settings (e.g., compilers and optimization flags that are safe to use). Moreover, the build analysis gives insight into the required capabilities of the target resources (e.g., Autotools macros, SCons scanners) and system software dependencies (the type of OS and crucial toolkits).

Even the stage-in, execution, and runtime phases may offer exploitable meta-information useful for the adaptation. If the application includes scripts facilitating these stages, then ADAPT analysis tools might detect the required input data and platform properties (configuration files, the methods the data are distributed, number and quality of computational resources, assumptions regarding the file system, etc). Moreover, the tool may determine the workflow of the intermediate data and establish the stream channels between conditioned application executables (or services). Even though the actual execution scripts do not exist, the ADAPT analyzing tool would reason about the specific execution based on the sample or historical runs, if provided.

In summary, that dimension may provide: (1) API requirements (libraries, frameworks, programming languages, versions), (2) semantic relations (communication patterns, algorithms and data structures), (3) quantitative information (number of processes/programs, size of data), (4) conditions for the resource capabilities (shared file system, runtime monitoring).

The matching of those properties against the available resource platforms will (1) narrow the possible targets, (2) allow preparing the adaptation scenarios, and (3) make possible the preliminary performance evaluation of the application-platform pairs.

The second dimension stems from our application model. The application unites several components: application programs, delivered dependencies, and data. This range describes such combinations that constitute the applications. Along this dimension, ADAPT analyzing tool may determine the properties regarding the format of the files and reason about the required capabilities that the hosting infrastructure should manifest in order to sustain the application execution. The application-target adaptation may reorganize the application constituents altering provided dependencies (e.g., library exchange, software substitutes for not available packages) and data (e.g., conversions, exchanging sources) or outsourcing elements impossible to deliver on the platform. As the result, examination of the application components widens scopes of adaptation as

we have prospect to test various adaptation techniques.

If the application provides the software dependencies, we should identify them to the possible degree as they contribute to the total application requirements. The valid identification may allow replacing theses dependencies with the natively provided by the target platforms. As the result, this would improve the application performance as well as target utilization attained for the application-target adaptation. Similarly, the input and intermediate data formats also are the objects to be adapted. In this matter, we need to provide the database systems and required protocols recognition. Moreover, the ADAPT analyzing tool may estimate the required data system capabilities such as size of the space, performance quality, volume of data to transmit, methods of the data stage-in, access and authentication mechanisms.

B. Extracting application characteristics

In order to extract the characteristics and to perform source-to-source translation of an input application, ADAPT consists of a custom translator which is built using the *ROSE compiler framework* [35].

ROSE is a compiler infrastructure supplying very complex methods for source-to-source code analysis and transformation. Among its numerous features, we emphasize those relevant to the functionality of ADAPT i.e. the support for analyzing applications written in C, C++, Fortran, OpenMP, or UPC, as well as the intermediate representations which are essential to static analysis and transformation of input codes.

At a more detailed view over ROSE major features, we notice why it is the compiler framework that is most suited for building tools for static code analysis and transformation.

- ROSE development history begins with Sage III and its predecessor Sage++ compiler preprocessor tool kit [36];
- ROSE analysis and transformation are achieved using intermediate representations (IR) which accurately store all information from the input code until the end of all necessary investigations and transformations;
- The ROSE framework has built-in methods for performing traversal, analysis and transformation of the IR, while checking for any violations of the programming language used for the input code;
- After all transformations to an IR are made, ROSE can write the modified IR into a new source code using the same programming language as the input one.

In the following are presented the main reasons why ROSE was chosen as foundation for ADAPT application characterization module.

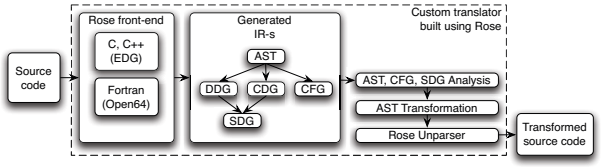

Figure 2: Custom translator built using Rose. The input source code is parsed by the front-end compiler; the IR are available for analysis and transformation; the new code is unparsed and ready for being built and ran.

- ADAPT aims at extracting the characteristics of scientific and engineering application. Nowadays, most compute- or communication-intensive applications meant to be executed on cloud resources are likely to be written in a programming language supported by ROSE, i.e. C, C++ or Fortran;
- ROSE front-end contains mechanisms that build very accurate IR of the input code;
- ROSE front-end builds IR-s such as the abstract syntax tree or the system dependence graph as well as program analyses (see Fig. 2) that ease access to IR-s for traversing and modifying them if needed.

The IR-s available through the custom built translator using ROSE are useful for a series of analyses that ADAPT imposes for determining characteristics and for performing transformations of the input code. Depending on the complexity of the input, the static analysis varies from using the abstract syntax tree up to the control, data or system dependence graphs. In what follows, we present the importance of each IR to ADAPT application characterization and transformation process.

The *Abstract Syntax Tree (AST)* is the simplest yet very accurate IR that ROSE builds from an input code. The AST is easy to traverse and just as easy to be modified. ADAPT uses the AST built by ROSE to identify key elements such as statements, basic blocks and communication calls. Regarding communication, ADAPT is making use of traversal methods in search of key instructions such as MPI_Init, MPI_Send or MPI_Recv. By identifying an MPI_Init, ADAPT knows that it is processing an MPI application. The next step in the automatic analysis is to find all communication calls that are giving an insight on message exchange pattern, e.g. to check whether the code is a master-slave application. Through AST traversal, ADAPT searches for MPI_Send, MPI_Recv, etc. and extracts their source and destination from the argument list. The basic rule of master-slave application (see section V, formula 1) is tested until either the check fails or the program end is reached. After the analysis and the transformations finish, the modified AST is written to a new source file; this output is called the transformed code with respect to the original (input) code. Finally, the transformed

code consists of the input code with all transformations performed on the AST. As long as the statements present in the source code do not depend on other variables, the AST suffices for performing static analysis on the input program, otherwise ADAPT must analyze the dependence graphs in order to make an accurate decision.

Data Dependence Graph (DDG) provides information on the dependences between data used in the code to be analyzed; *Control Dependence Graph (CDG)* represents the control dependences between the vertices of an AST. *System Dependence Graph (SDG)* [37] is the super-graph combining both data and control dependences in one representation. If the static analysis is unable to yield the expected results only by examining AST then SDG becomes the correct representation to analyze. Currently, we use the analysis of SDG for extracting application characteristics in a completely static manner.

C. Source-to-source automatic transformation to fit the Azure cloud environment

The proof-of-concept presented in this paper involves automatic transformations of the input code in order to adapt it to a suitable deployment environment, as a result of the analysis described in section IV-B. Since we took the example of C/MPI master-slave application, in the following we present how the source-to-source transformation occurs.

The custom translator built using ROSE and included in ADAPT, statically analyzes the input application. Upon this analysis, a decision is made: if the code does not follow the master-slave paradigm, the analysis ends without any further transformations. Otherwise, ADAPT initiates and proceed to the transformation phase. These decisions are implemented as part of the core engine of Adapt regarding the possible scenarios (see figure 1). At the next stage, ADAPT analyzes the code's AST and modifies the communication-related aspects, i.e. obtaining ranks and exchanging messages among other processes. The result is the transformed code that no longer depends on MPI libraries but requires an intermediate Azure-compatible middleware provided by ADAPT. This wrapper simplifies the code transformation so the code can be directly executed on Azure. From the MPI calls we create comments with the essential parameters such as the source, the destination and the message to be transmitted. The wrapper that we define is written in C# and addresses C, C++, and Fortran codes that need to run on Windows Azure. The code resulting from the static analysis is included in our Windows Azure master-worker template without any other modifications. The Windows Azure master-worker project consists of basic Web Role - Worker Role applications that include the computation parts compiled in the DLL. This DLL is, in fact, the transformed code obtained upon static analysis, which was compiled as part of an Azure application. When the Azure project is launched, the communication is supported natively by Windows Azure Web Roles and Worker Roles.

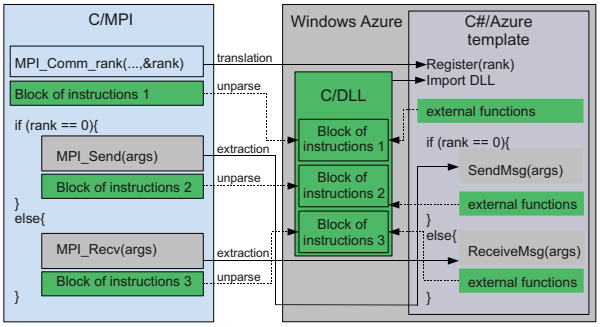

Figure 3: Transforming a C/MPI code into a Windows Azure cloud application template. The sequential code is copied to the C/C#/Azure application; the MPI communication are defined in our MPI-Azure wrapper and called from within the C/C#/Azure code.

V. Test-case

The objective of this experiment is to test one characteristic detection on a very simple case. It does not validate the whole approach but shows that this proof-of-concept can work. The experiment is the following. A program source code is sent to ADAPT. The ROSE front-end has to identify a master-slave paradigm using MPI and, if the detection is successful, to transform the program into an Azure application. This ROSE front-end implements the following method for detecting this paradigm. Let $G_k(V,E)$ be k directed graphs with $V(G_k)$ representing the tasks and $E(G_k)$ the communications of a program P. $G_k(V,E)$ represents therefore the communication scheme of task number k. Then, P follows a master-worker paradigm if and only if:

$$\forall k \in [1; ntasks], \nexists\, v_j / v_i v_j \in E(G_i)\, with\, j \neq 0 \qquad (1)$$

In other words, the above-presented formula imposes that communication must comply with a *star* topology, with the master being the center of the star.

We chose three simple C/MPI applications [38], [39], [40] out of which two are using the master-slave paradigm and the third one is not.

Each application in turn is passed at ADAPT input. The analysis and the decision are made in a fully static manner as described in section IV-B. The main operations (see Fig. 3) which take place inside the translator are:

- creating the AST of the input codes *mpi1.c*, *mpi2.c* and *mpi3.c* respectively;
- identifying the variable representing the *rank*;
- searching for the conditional statement which decides whether a *rank* is the *master* or the *worker*;

- scanning the worker-related instructions in search of MPI function calls;
- when an MPI function call is found, extract its arguments and check the source, if it's an MPI_Recv, or the destination, if it's an MPI_Send;
- make a decision based on formula (1), i.e. see if any of the slave processes are programmed to exchange messages with anyone else than the master process;

A negative decision may be made prematurely if collective communications are found in the slaves code.

Table I shows the decisions made by ADAPT and presents the time cost for analyzing application characteristics and taking a decision based on the custom built translator using ROSE.

If the input code is indeed a master-slave paradigm, then transformation occurs. Fig. 3 shows the idea used in ADAPT for the phase following the correct identification of a master-slave code type. The most important aspect of the code transformation is splitting the input into three parts:

1) Assigning a rank to current process. The MPI syntax for obtaining the rank is translated into a call to the Register member defined in our C#/Azure template;

2) Identifying MPI communication calls, extracting their arguments and inserting them into the Azure application as calls to members defined in our C#/Azure template. Relevant MPI_Send arguments are used to call the SendMsg member of our template. In the same manner, messages will be received using arguments extracted from MPI_Recv. These arguments enable calls to the ReceiveMsg member in the C#/Azure template;

3) Unparsing the instruction blocks to standard sequential C code. The AST nodes representing the MPI calls identified in the previously presented step (see the AST excerpt before transformation [41]) are now being removed from the tree (see the AST excerpt after the transformation [41]). The resulting code is communication-free and is then inserted into the Azure application as a C/DLL code. This DLL is interfaced to the Windows Azure master-slave cloud application through our C#/Azure template. The template makes use of Windows Azure Native library which supports native C and C++ code execution in a C# environment.

Upon execution of the newly obtained Windows Azure application, the result is hypothetically assumed to be no different than the initial MPI code. We have therefore showed that the adaptation of a C/MPI application for an execution on top of the Azure cloud service is possible.

VI. Conclusion

In this paper, we present a novel concept for enhancing usability of cyber-infrastructure platforms through an adaptation process. The implementation of this concept called ADAPT may allow developers and users of scientific applications to broaden their possibilities for target execution platforms. The objective of ADAPT is to be as transparent as possible for the user who can focus on selecting the most appropriate execution platform rather than dealing with the burden of adaptation. Early experiments show that it is possible to analyze and to transform a C/MPI application (written assuming a cluster target) into a C/C#/Azure program (for the Microsoft Azure cloud) without requiring any user intervention. These initial demonstrations need further evaluations and more experiments should be conducted to confirm the extent to which they are generalizable. ADAPT takes advantage of an existing source-to-source translator built using ROSE, which helps in code analysis as well as in code transformation. Next steps in our project will include: (1) extending the possible execution platforms by including Hadoop or Google App Engine, (2) analyzing more complicated master/worker programs where the interaction between the master and the workers is more sophisticated, (3) extending the scheme to include other possible communication patterns, such as a ring or regular mesh structure which can be transformed using templates on target cloud platforms.

Table I: Application characterization automatically made by ADAPT. $^{(1)}$Time cost to make decision;

Input code	Master - slave	$t_{decision}$ $^{(1)}$ [ms]
mpi1.c	Yes	0.243
mpi2.c	Yes	0.233
mpi3.c	No	0.258

Acknowledgement

This work is partially funded by the French National Agency for Research (ANR-07-CIS7-011-01 [42]) and US National Science Foundation grant CNS-0720761.

References

[1] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *International Journal of High Performance Computing Applications*, vol. 11, no. 2, p. 115, 1997.

[2] "The Simple Cloud API," http://www.simplecloudapi.org/, 2011.

[3] "Open Cloud Manifesto," http://www.opencloudmanifesto.org/, 2011.

[4] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus Open-source Cloud-computing System," in *9th IEEE Int. Symposium on Cluster Computing and the Grid*, 2009.

[5] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa, "Science Clouds: Early Experiences in Cloud Computing for Scientific Applications," in *CCA'08: Cloud Computing and Its Application*, 2008.

[6] N. Chohan, C. Bunch, S. Pang, C. Krintz, N. Mostafa, S. Soman, and R. Wolski, "AppScale Design and Implementation," UCSB Technical Report No.2009, Tech. Rep., 2009.

[7] "Amazon Elastic MapReduce," http://aws.amazon.com/elasticmapreduce/, 2011.

[8] M. Kozuch, M. Ryan, R. Gass, S. Schlosser, D. O'Hallaron, J. Cipar, E. Krevat, J. López, M. Stroucken, and G. Ganger, "Tashi: location-aware cluster management," in *1-st Workshop on Automated Control for Datacenters and Clouds*. ACM, 2009, pp. 43–48.

[9] P. Skomoroch, "MPI Cluster Programming with Python and Amazon EC2," in *PyCon*, 2008.

[10] J. Slawinski, M. Slawinska, and V. Sunderam, "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*, 2009.

[11] H. Kim, Y. el Khamra, S. Jha, and M. Parashar, "An autonomic approach to integrated hpc grid and cloud usage," in *e-Science'09: Fifth IEEE International Conference on e-Science*. IEEE, 2009, pp. 366–373.

[12] C. Vecchiola, S. Pandey, and R. Buyya, "High-performance cloud computing: A view of scientific applications," in *10th International Symposium on Pervasive Systems, Algorithms, and Networks*. IEEE, 2009, pp. 4–16.

[13] "Amazon Elastic Compute Cloud (Amazon EC2)," http://aws.amazon.com/ec2/, 2011.

[14] "Rackspace hosting," http://www.rackspace.com, 2011.

[15] "PortableApps.com web page," http://portableapps.com/, 2011.

[16] "rPath Documentation," http://docs.rpath.com/, 2011.

[17] A. Sæbjørnsen, J. Willcock, T. Panas, D. Quinlan, and Z. Su, "Detecting code clones in binary executables," in *ISSTA'09: Proceedings of the 18th International Symposium on Software Testing and Analysis*. ACM, 2009, pp. 117–128.

[18] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *Science of Computer Programming*, vol. 74, pp. 470–495, 2009.

[19] J. H. Johnson, "Identifying redundancy in source code using fingerprints," in *CASCON'93: Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: software engineering*, vol. 1. IBM Press, 1993, pp. 171–183.

[20] B. Baker, "On finding duplication and near-duplication in large software systems," in *Proceedings of 2nd Working Conference on Reverse Engineering*, 1995, pp. 86 –95.

[21] J. Mayrand, C. Leblanc, and E. Merlo, "Experiment on the automatic detection of function clones in a software system using metrics," in *Proceedings of the International Conference on Software Maintenance*, 1996, pp. 244 –253.

[22] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees," *IEEE International Conference on Software Maintenance*, p. 368, 1998.

[23] R. Komondoor and S. Horwitz, "Using slicing to identify duplication in source code," in *SAS'01: Proceedings of the 8th International Symposium on Static Analysis*. Springer-Verlag, 2001, pp. 40–56.

[24] A. Marcus and J. I. Maletic, "Identification of high-level concept clones in source code," in *ASE'01: 16th IEEE international conference on Automated software engineering*. IEEE Computer Society, 2001, pp. 107–.

[25] D. Van Der Spoel, E. Lindahl, B. Hess, G. Groenhof, A. E. Mark, and H. J. Berendsen, "GROMACS: fast, flexible, and free." *Journal of computational chemistry*, vol. 26, no. 16, pp. 1701–1718, 2005.

[26] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, P. Frederickson, T. Lasinski, R. Schreiber *et al.*, "The NAS Parallel Benchmarks," *International Journal of HPC Apps*, vol. 5, no. 3, p. 63, 1991.

[27] J. Slawinski, M. Slawinska, and V. Sunderam, "Unibus-managed Execution of Scientific Applications on Aggregated Clouds," in *CCGrid'10: 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE, 2010, pp. 518–521.

[28] T. Hoefler, A. Lumsdaine, and J. Dongarra, "Towards Efficient MapReduce Using MPI," *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pp. 240–249, 2009.

[29] J. Gray, D. Liu, M. Nieto-Santisteban, A. Szalay, D. DeWitt, and G. Heber, "Scientific data management in the coming decade," *ACM SIGMOD Record*, vol. 34, no. 4, pp. 34–41, 2005.

[30] T. Redkar, *Windows Azure Platform*, 2010.

[31] "FAQ About Azure for Research," http://research.microsoft.com/en-us/projects/azure/faq.aspx, 2011.

[32] "How to move MPI-style apps into Azure?" http://social.msdn.microsoft.com/Forums/en/windowsazure/thread/bb5a82ed-9411-4a31-a1f3-12f0a0111f25, 2010.

[33] "GQL Reference," http://code.google.com/appengine/docs/python/datastore/gqlreference.html, 2011.

[34] M. Slawinska, J. Slawinski, and V. Sunderam, "Enhancing Build-Portability for Scientific Applications Across Heterogeneous Platforms," 2009, pp. 1–8.

[35] M. Schordan and D. Quinlan, "A source-to-source architecture for user-defined optimizations," in *Modular Programming Languages*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2003, vol. 2789, pp. 214–223.

[36] F. Bodin, P. Beckman, D. Gannon, J. Gotwals, S. Narayana, S. Srinivas, and B. Winnicka, "Sage++: An object-oriented toolkit and class library for building Fortran and C++ restructuring tools," in *OON-SKI'94: The 2nd annual object-oriented numerics conference*, 1994, pp. 122–136.

[37] P. E. Livadas and S. Croll, "System dependence graphs based on parse trees and their use in software maintenance," *Information Sciences*, vol. 76, no. 3-4, pp. 197–232, 1994.

[38] T. Way. Mpi test code 1. [Online]. Available: http://www.csc.villanova.edu/~tway/courses/csc8400/s2007/handouts/mpi.ex1.c

[39] Mpi test code 2. [Online]. Available: http://www.mcs.anl.gov/research/projects/mpi/tutorial/mpiexmpl/src2/io/C/solution.html

[40] Mpi test code 3. [Online]. Available: http://www.itdefense.org/open/education/csc/parallel/zong_spring2010/mpi/mpi_demo1.cpp

[41] J. Bourgeois, V. Sunderam, J. Slawinski, and B. Cornea. Online resources on ADAPT. [Online]. Available: http://lifc.univ-fcomte.fr/~bcornea/recherche#extending

[42] ANR CIP project web page. [Online]. Available: http://spiderman-2.laas.fr/CIS-CIP