A Service Composition Framework for Market-Oriented High Performance Computing Cloud

Tran Vu Pham
Ho Chi Minh City University of Technology, Vietnam
t.v.pham@cse.hcmut.edu.vn

Hani Jamjoom
IBM T.J. Watson Research Center
jamjoom@us.ibm.com

Kirk Jordan
IBM T.J. Watson Research Center
kjordan@us.ibm.com

Zon-Yin Shae
IBM T.J. Watson Research Center
zshae@us.ibm.com

ABSTRACT

Despite the success High Performance Computing (HPC) across a number of application domains, the adoption of HPC resources and applications is still limited, primarily due to its high capital cost, system complexity, application availability, and service delivery model. Recently, several research efforts have shown that the emerging Cloud Computing service model can improve on-demand access to HPC capacity as utility. This paper introduces a framework for on-demand composing and deploying available HPC applications as services on HPC clouds. The composition is enabled by an ontology that describes dependencies and relationships among HPC software and resources.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*

1. INTRODUCTION

Today, there is a wide-spectrum of highly successful applications running on top of High Performance Computing (HPC) platforms. Unfortunately, they have been driven by a relatively limited number of research and commercial institutions. While some improvements have been observed with Grid Computing initiatives, large-scale adoption continues to be hindered. There are a number of reasons for this slow adoption, including the high capital cost, system complexity, and the lack of comprehensive resource management and service models.

At the same time, cloud computing [5,8,11], which is driven by industry interest, is emerging as a disruptive paradigm that is giving users easy access to large numbers of software applications, platforms and virtualized computational resources. By enabling a cloud model in HPC, it creates opportunities for new applications that can take advantage of the underlying computing capability, and accelerates the adoption of scientific applications from research community to commercial space. Early experiences showed that HPC clusters for running scientific applications can be feasibly provisioned by commercial cloud computing providers [9, 10, 13, 16]. These studies point to a trade-off in performance and that cloud computing is suitable for a limited range of scientific applications. There have also been efforts to develop compute clouds specifically for scientific applications [13, 17]. The use of virtualization together with virtual appliances [15] in scientific clouds enables on-demand provisioning of homogeneous resources with little effort for software configuration [6, 12].

Motivated by the above challenge, we have developed an HPC cloud market paradigm. Our model expands on previous market models (e.g [7,8]) that focus on how to trade computational resources to include applications, even if they are still in early stages of development. The market encompasses various HPC stakeholders, including research scientists, application developers, independent service providers, and HPC cloud providers. This creates opportunities for participants with different roles to interact on different stages of application lifecycles. To enable the above HPC cloud market paradigm, we have developed a composition framework for on-demand conversion of applications into services on HPC clouds. The core of the composition framework is an ontology, which describes the available applications, various system components involved and their interdependencies. The rest of this paper will discuss the proposed market paradigm and the composition framework in more detail.

2. HPC MARKET PARADIGM

An HPC market paradigm based on cloud computing model is adopted to allow the consumption of HPC computational resources as utility, making it available to a wider community at reasonable costs. The orchestration within the market is demonstrated below through a scenario from oil and gas industry. There, large companies usually work in collaboration with research institutions to develop new applications/models and also have their own internal HPC systems for evaluation of the new applications/models without re-

HPDC'10, June 20–25, 2010, Chicago, Illinois, USA.

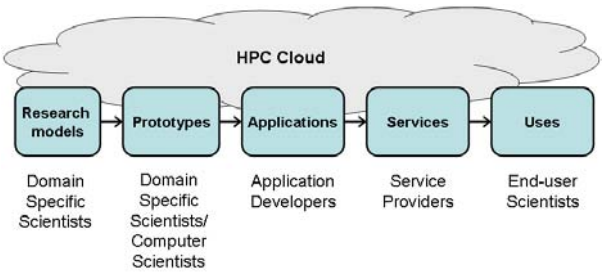

Figure 1: Participants with different roles contribute different values to the technology value chain in the HPC market.

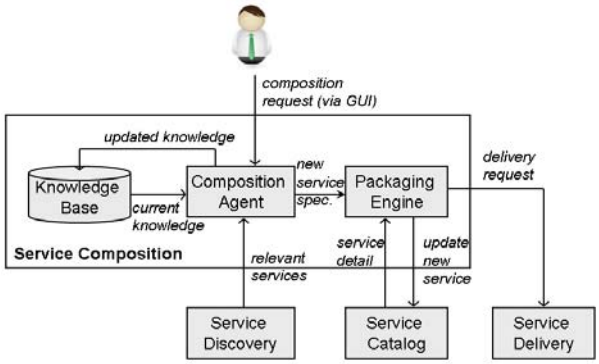

Figure 2: Cloud service composition architecture

leasing their private datasets to other parties. For small and medium companies, evaluation of new applications/models is more challenging as these companies often have limited access to HPC systems. Although they may have access to new applications/models from oil/gas research community, they do not have their own facility to test the applications/models with their datasets.

With the HPC market paradigm, the problems of small and medium companies can be solved in the following ways: (1) they can get access to HPC computational resources; (2) they can use the applications/models, even if they are under development, with their own datasets; (3) they are still in control of their valuable datasets. This model also creates opportunities for other parties involved. For researchers, their research applications/models are more visible to industry. Hence, the chance of their adoption is increased. They can also use HPC development platforms provided in the cloud for developing applications/models. This is difficult in the pre-cloud era, where HPC computational resources are still expensive to acquire and run. In addition, researchers will be able to receive early feedback from industry. For HPC computational resource providers, their resources can sustain higher utilization, thus be more competitive in the market. There may be independent service vendors, whose interest is on providing value-added services. They assemble different pieces (e.g. applications, models, operating systems, etc.) into readily accessible services and resale them to others in need. The chain in which different pieces are put together to create new value is referred to as a *technology value chain*, which is summarized in Figure 1.

3. SERVICE COMPOSITION FRAMEWORK

The service composition has been explored by many researches for Grid services and Web services, in which service interfaces are well defined using Web Service Description Language. However, Cloud Computing services are not necessary in form of Grid or Web services. Particularly in the system being introduced, the services could be software applications, platforms or computing infrastructures. Service composition is the process of creating a new service by assembling different software components (e.g. applications, APIs, libraries and operating systems) and hardware infrastructures. The software components themselves can be (part of) other available services.

3.1 Cloud Service Composition Architecture

The overall architecture of the Service Composition subsystem is described in Figure 2. The composition process is initiated by the user. The request could be for turning an application into a service or for creating a service platform, which consists of a set of applications, APIs and an operating system. A simple request could only be the application name. The service composition engine will use its knowledge to find the remaining components to assemble the service. A complex request may consist of many other service attributes such as requirements on hardware capability, service quality, and so on.

The composition request is handled by the Composition Agent. The agent parses the request and uses knowledge provided by the Knowledge Base to iteratively resolve all dependencies. It interacts with Service Discovery to retrieve information about existing relevant services. The composition will only be successful if all the required components (required APIs, compatible operating systems and hardware infrastructures) exist. The result of a successful composition delivered by the Composition Agent is a new service specification which details the required components of the services and the dependencies among them for deployment. This specification is used to update the Knowledge Base for later use. It is then forwarded to the Packaging Engine for packing all the software components into a software package. Newly created software package, together with the service specification, is then registered in the Service Catalog and forwarded to Service Delivery for provisioning the service.

3.2 Knowledge Base

The Knowledge Base provides the knowledge of the existing services, software applications, hardware and their interrelationships for the Composition Agent to reason about during compositions. An ontology has been developed using Web Ontology Language (OWL) [2] for capturing the knowledge needed. The classes of the ontology that can be used to describe components directly involved in composition processes and their hierarchy are shown in Figure 3. In addition to the classes of entities and their hierarchy, the current version of the ontology also provides mechanisms for capturing the various types of relationship exist among the entities, including:

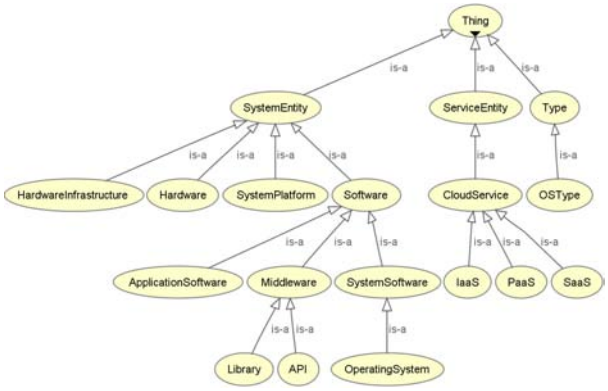

Figure 3: Hierarchy of terms in the ontology

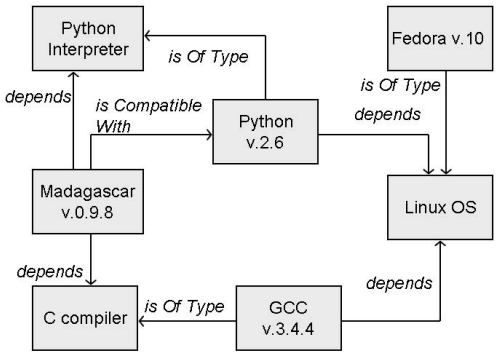

Figure 4: Using ontology to model Madagascar and its dependencies

- *Dependency:* refers to the type of relationship between two components in which one component must exist for the other to function.
- *Compatibility:* if a component is compatible with another component, they will be able to work together in one system setting.
- *Conflict:* if the two components are said to be conflict with each other, they will not function when they are put together in one system. Conflict relationship is symmetric, which means if A conflicts with B then B also conflicts with A.
- *Whole-part:* the type of relationships between two components, in which one component is a part of the other. Whole-part relationship is transitive.
- *Type:* grouping of instances that have the same set of functionality. Instances of the same type should not co-exist within a system. A type can be further described by subtypes to better reflect the dependencies.

The relationships are expressed in OWL as object properties. The ontology also has mechanisms for capturing hardware requirements for software to operate. For example, to achieve optimal performance, a software application may need a hardware infrastructure with certain number of processors, clock speed, amount of memory, etc.

Figure 4 shows an example of using the ontology to model Madagascar [1], which is an open-source software package for multidimensional data analysis and reproducible computational experiments (often used in seismic research). In this model, three different types are introduced: Python Interpreter, C Compiler and Linux OS. Others are specific software instances.

This model implies that if there is another C compiler, say GCC v.3.4.3, it then can be used with the Madagascar v.0.9.8, but only either of GCC v.3.4.4 or GCC v.3.4.3 can be used in one system setting. However, it is not the same for Python Interpreter, where there is a declaration of compatibility. Compatibility relationship is used to specify the instances of a particular type that are compatible with another instance. For example, figure 4 says that among the instances of type Python Interpreter only Python v.2.6 can be used with the Madagascar package.

3.3 Reasoning

The reasoning process is carried out by the Composition Agent when receiving a user request for a service. The user is expected to know only what he/she wants from the service. The Composition Agent needs to work out how to get the service for the user. The knowledge captured in models stored in the Knowledge Base is used to address the user's need. Firstly, the agent extracts from the related models all dependencies related to the user's request. Then, the agent following the dependency graph to select all the required system components. Secondly, once the set of all the system components have been identified, the agent produces a set of hardware requirements that can satisfy all the identified software components, based on the specifications of the software components in the Knowledge Base. Then, it searches against the Knowledge Base for suitable hardware infrastructures. The infrastructures are then matched against available services.

The current prototype was developed using Java. OWL-API [3] and Pellet [4] were used as tools for ontology processing and reasoning, respectively. During the composition, temporary solutions are stored in directed acyclic graph data structures. A solution graph is initialized with the requested component(s). Starting from these intial components, the reasoner will follows the dependencies declared in the Knowledge Base to identify the necessary components for deployment of the requested components. Once identified, components at the dependency targets are added to the solution graph one by one, as long as all the associated constraints are satisfied. The temporary solution graph becomes the final solution when all the dependencies are successfully resolved.

4. RELATED WORK

A number of market models have been introduced in HPC. In these market models, HPC computational resources are the main objects. In our paradigm, the market is extended

for exchanging of compute models, applications and composed cloud services. This extension provides opportunities for participants beyond HPC resource providers, to include application developers, modelers, and independent service vendors. Their collaborations can create extra values in the market.

To some extent, our service composition framework is analogous to Web Service compositions. The distinctions are that in our approach, the objects of the composition are software components and the composition is the process of assembling the components together, constrained by their relationships. In contrast, Web Services composition focuses on how to assemble services together, starting with a given set of inputs and conditions, to produce the expected set of outputs, constrained by the matching of service inputs, outputs and their operating conditions.

The problem with software dependency addressed in this paper is close to the one in software management. There have been solutions such as RPM for the Linux operating systems and, a more general approach, Solution Deployment Descriptor (SDD) by OASIS [14]. In our solution, the dependencies—relationships in general—are described at a coarse level, assuming that the users who contribute the software components may have only partial knowledge of their whole target deployment environments. This high level way of describing relationships allows quick resolution of the dependencies.

5. CONCLUSION

In this paper, we have introduced a market-oriented paradigm for HPC cloud deployments. A major distinction between the proposed market paradigm and previous proposals is the introduction of new resource dimensions, which we believe improves the collaboration between different parties. Additionally, we have detailed a framework for composing different software components, available on the open market, into services that can be deployed in the HPC cloud. We believe that our proposed approach of providing access to HPC resources and applications as on-demand cloud services, enabled by the automatic composition, is important for accelerating the usage and commercialization of HPC resources.

Undoubtedly, there is a number of challenges with the proposed market model that we do not address in this paper. While we expect many of the technical challenges to be overcome with today's technologies, a robust licensing and costing model needs to be established to ensure wide-scale adoption of HPC clouds.

6. REFERENCES

[1] Madagascar. http://www.reproducibility.org/wiki/Main_Page; accessed December 04, 2009.

[2] OWL - Web Ontology Language. http://www.w3.org/2004/OWL/; Accessed December 10, 2009.

[3] Owl api. http://owlapi.sourceforge.net/; Accessed December 10, 2009.

[4] Pellet: Owl 2 reasoner for java. http://clarkparsia.com/pellet/; Accessed December 10, 2009.

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. *Above the Clouds: A Berkeley View of Cloud Computing*. UC Berkeley, Feb 2009.

[6] R. Bradshaw, N. Desai, T. Freeman, and K. Keahey. A scalable approach to deploying and managing appliances. In *TeraGrid 2007*, Madison, WI, USA, June 2007.

[7] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1507–1542, 2002.

[8] R. Buyya, C. S. Yeo, and S. Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *Proc. The 10th IEEE International Conference on High Performance Computing and Communications (HPCC-08)*, Dalian, China, 2008.

[9] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The cost of doing science on the cloud: the montage example. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.

[10] C. Evangelinos and C. Hill. Cloud computing for parallel scientific hpc applications: Feasibility of running coupled atmosphere-ocean climate models on amazon's EC2. In *Cloud Computing and its Applications (CCA-08)*, Chicago, IL, USA, Oct 2008.

[11] B. Hayes. Cloud computing. *Commun. ACM*, 51(7):9–11, 2008.

[12] K. Keahey and T. Freeman. Contextualization: Providing one-click virtual clusters. In *eScience 2008*, Indianapolis, IN, USA, Dec 2008.

[13] K. Keahey and T. Freeman. Science clouds: Early experiences in cloud computing for scientific applications. In *Cloud Computing and Its Applications 2008 (CCA-08)*, Chicago, IL, USA, Oct 2008.

[14] OASIS Solution Deployment Descriptor (SDD) TC. Solution deployment descriptor specification 1.0. Technical report, September 2008. Available at: http://docs.oasis-open.org/sdd/v1.0/os/sdd-spec-v1.0-os.pdf.

[15] C. Sapuntzakis, D. Brumley, R. Chandra, N. Zeldovich, J. Chow, M. S. Lam, and M. Rosenblum. Virtual appliances for deploying and maintaining software. In *LISA '03: Proceedings of the 17th USENIX conference on System administration*, pages 181–194, Berkeley, CA, USA, 2003. USENIX Association.

[16] T. Sterling and D. Stark. A high-performance computing forecast: Partly cloudy. *Computing in Science and Engineering*, 11(4):42–49, 2009.

[17] L. Wang, J. Tao, M. Kunze, A. C. Castellanos, D. Kramer, and W. Karl. Scientific cloud computing: Early definition and experience. In *10th IEEE International Conference on High Performance Computing and Communications*, pages 825–830, Dalian, China, 2008.