CLOUD SERVICES FOR AUTOMATION OF SCIENTIFIC AND ENGINEERING COMPUTATIONS

Sukhoroslov O.V.[1], Putilina E.V.[1]
Institute for Information Transmission Problems of the Russian Academy of Sciences, Moscow, Russia [1]

Abstract: *The report discusses common problems associated with the automation of scientific and engineering computations, as well as promising approaches to solving these problems, based on cloud computing models. The use of service-oriented approach in scientific and engineering computing can improve the research productivity by enabling publication and reuse of computing applications, as well as creation of cloud services for automation of computation processes. An implementation of this approach is presented in the form of Everest cloud platform which supports publication, execution and composition of computing applications in a distributed environment.*
KEYWORDS*: RESEARCH AUTOMATION, WEB SERVICES, CLOUD COMPUTING, DISTRIBUTED COMPUTING, APPLICATION COMPOSITION, WORKFLOW, PARAMETER SWEEP APPLICATIONS*

1. Introduction

The modern scientific and engineering research require the use of computational software and high-performance computing resources. There are several common activities associated with such research, such as running computing applications on HPC resources, integration of multiple computing resources, sharing of computing applications, combined use of multiple applications and running parameter sweep experiments. Due to the inherent complexity of computing software and infrastructures, as well as the lack of required IT expertise among the researchers and engineers, all these actions require a significant amount of automation in order to be widely applied in practice.

The use of service-oriented approach in scientific and engineering computing can improve the research productivity by enabling publication and reuse of computing applications, as well as creation of cloud services for automation of computation processes [1]. An implementation of this approach is presented in the form of Everest cloud platform [2,3] which supports publication, execution and composition of computing applications in a distributed environment.

Everest follows the Platform as a Service model by providing all its functionality via remote web and programming interfaces. A single instance of the platform can be accessed by many users in order to create, run and share applications with each other without the need to install additional software on their machines. Any application added to Everest is automatically published both as a user-facing web form and a web service. Unlike other solutions, Everest runs applications on external computing resources connected by users, implements flexible binding of resources to applications and provides an open programming interface.

The paper is structured as follows. Section 2 briefly describes the architecture of Everest, while Sections 3-7 discuss the approaches used in Everest to addresses the mentioned problems related to the automation of computing activities. Section 8 concludes and discusses future work.

2. Everest Architecture

A high-level architecture of Everest is presented in Figure 1. The server-side part of the platform is composed of three main layers: REST API, Applications layer and Compute layer. The client-side part includes web user interface (Web UI) and client libraries.

REST API is the platform's application programming interface implemented as a RESTful web service [4]. It includes operations for accessing and managing applications, jobs, resources and other platform entities. REST API serves as a single entry point for all clients, including Web UI and client libraries.

Applications layer corresponds to a hosting environment for applications created by users. Applications are the core entities in Everest that represent reusable computational units that follow a well-defined model [5]. Each application created by user is automatically exposed as a RESTful web service via the platform's REST API. This enables remote access to the application both via Web UI and client libraries. An application owner can manage the list of users that are allowed to run the application.

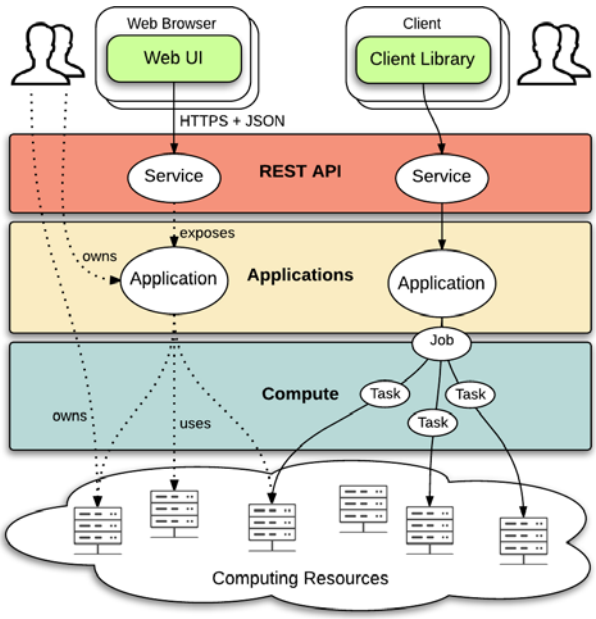

Fig. 1 *Everest Architecture.*

Everest doesn't provide its own computing infrastructure to run applications, nor does it provide access to some fixed external infrastructure like grid. Instead Everest enables users to attach to it any external computing resources and to run applications on arbitrary sets of these resources.

Compute layer manages execution of applications on remote computing resources. When an application is invoked via REST API it generates a job consisting of one or more computational tasks. Compute layer manages execution of these jobs on remote resources and performs all routine actions related to staging of task input files, submitting a task, monitoring a task state and downloading task results. Compute layer also monitors the state of resources attached to the platform and uses this information during job scheduling.

Web UI provides a convenient graphical interface for interaction with the platform. It is implemented as a JavaScript application that can run in any modern web browser. Web UI provides access to all functionality of the platform. It is built directly on top of the REST API, i.e., it uses the same interface as all other platform clients.

Client libraries simplify programmatic access to Everest via REST API and enable users to easily write programs that access applications and combine them in arbitrary workflows. At the moment, a client library for Python programming language is implemented.

3. *Running Applications on HPC Resources*

The first problem faced by researchers trying to use HPC resources for performing computations is the lack of convenient interfaces for submitting their jobs. The most common way of interacting with HPC cluster is via command-line shell using batch system commands, which is too low-level and difficult for researchers without Linux background. Other major obstacles are building and configuring applications on such resources. This seriously impacts research productivity and limits the scope of researchers using HPC resources today.

A common approach for solving this problem is by building convenient high-level user interfaces for HPC resources that run preinstalled and configured applications. There are two types of such interfaces: *generic*, which allow submission of any kind of computing jobs and applications, and *domain-specific*, which support running a specific application or class of applications. The domain-specific interfaces are the most convenient for researchers because they allow specifying computing job in terms of the problem being solved. There are two approaches for implementing high-level user interfaces: using *web-based interfaces* (e.g. via web portals [6]) or integrating with established *problem-solving environments* (e.g. MATLAB). Both approaches are convenient for unskilled researchers, because the first approach doesn't require installation of additional software on user's machine, while the second one uses familiar environment.

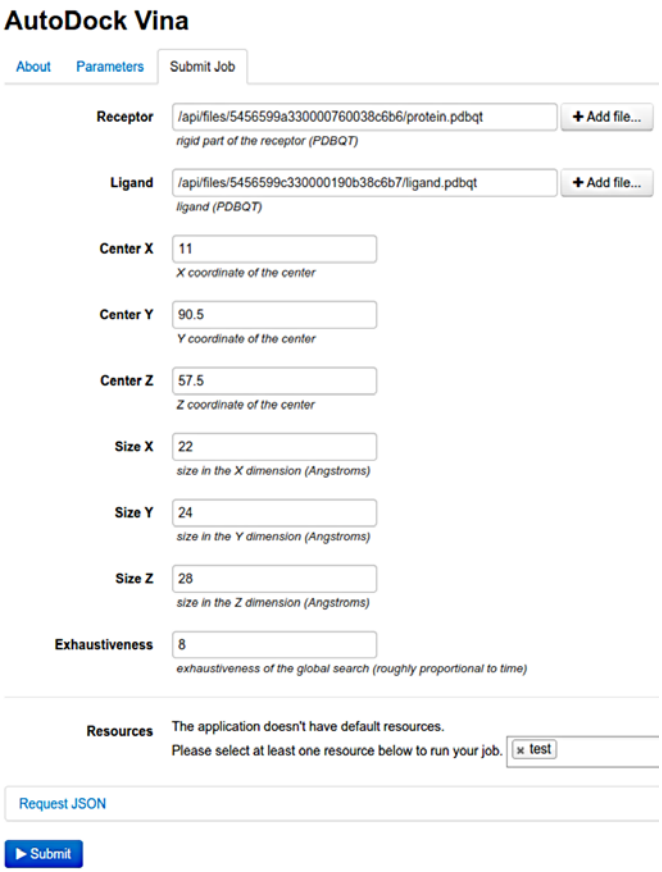

***Fig. 2** A web form for running Autodock Vina generated by Everest.*

Everest implements convenient access to computing applications via domain-specific web interfaces. For each published application Everest automatically generates a web form for specifying input values and running the application. An example of such interface for AutoDock Vina, a molecular docking application, is shown on Figure 2. While existing computational portals implement similar interfaces, typically only portal administrators can add new applications or resources to the system. In contrast, Everest enables users to add new applications by themselves and manage them via convenient web interface. This approach helps to engage researchers in sharing applications with each other (see the next section).

4. *Sharing of Applications and Resources*

Sharing of computing applications is crucial for any collaborative work among researchers, such as research projects. The reuse of existing applications built by fellow researchers helps to avoid costly reimplementation. The traditional approaches of sharing applications include sending application binary via email, installing it on some shared computing resource, or sharing a code repository. All these approaches are somewhat inconvenient for either users or application developers. For example, users have to download and rebuild applications each time it changes, or developers have to manually perform such updates.

Everest uses Software as a Service model to facilitate application sharing and solve the mentioned problems. A user can add an application to Everest and specify users and groups that are allowed to run this application. This instantly makes the application available to all these users without requiring any installation. This also streamlines application updates because all users access a single application instance managed by the developer and don't have to perform updates themselves.

The application sharing brings also an important issue of resource sharing. Since any application requires a computing resource to run, the application owner should either provide a resource to run the application by allowed users or let the users utilize their resources for running the application. Everest implements flexible binding of resources to applications supporting different use cases discussed below.

Static resource binding means that an application owner configures a static set of resources that should be used by Everest to run the application. In this case the owner implicitly allows application users to run the application on these resources. This approach is used by many scientific web services that are tied to a fixed computing infrastructure. Static binding is convenient for application users since they don't deal with resources directly. Such approach is also desirable if an application has specific hardware requirements or commercial value. However, in this case the application owner has to supply computing resources in addition to the application itself. This is a serious barrier for individual scientists wanting to easily share their applications in the form of services with as many colleagues as possible.

Dynamic resource binding means that an application user manually selects resources for running her job. This approach eliminates the mentioned barrier by enabling users to run the application on their resources. In this case the application owner need not to supply any computing resources. For the application users dynamic binding implies the need to attach their computing resources to Everest. However, this operation should be done only once for each resource.

Implementation of dynamic binding is usually problematic for self-hosted applications since there is a lack of trust between application owner and users for direct delegation of resource credentials. Since Everest both hosts applications and interacts with resources it can play a role of a trusted third party ensuring that an application owner can not gain direct access to a user's resource.

Everest implements both static and dynamic resource binding models. It is possible to use both models in an application by providing default static resources and enabling users to dynamically override it with their resources.

5. *Integration of Computing Resources*

Modern scientific and engineering computations require the use of high-performance computing resources. Nowadays the researchers have at their disposal a variety of such resources, including standalone servers, computing clusters, grid infrastructures and clouds. The aggregation of computing potential of these resources could both greatly enhance the research productivity and increase

the resource utilization. However, there is a lack of convenient tools that allow users to seamlessly combine the available resources into a single computing environment. Grid infrastructures [7] target only computing clusters and don't provide access to all resources, while distributed computing toolkits [8] require installation and configuration on a user's machine.

Everest uses Platform as a Service model to facilitate integration of computing resources and solve the mentioned problems. A user can attach to Everest its resources and let the platform to run applications across arbitrary sets of resources. From this point of view Everest can be seen as a multitenant metascheduling service.

Currently the preferred method for attaching a resource to Everest is based on using a developed program called *agent*. The agent runs on the resource and acts as a mediator between it and Everest. This approach has a number of advantages in comparison to plain SSH access such as supporting resources without inbound connectivity (behind a firewall or NAT) and ability to control actions performed by Everest on the resource. However, such approach requires manual deployment of the agent on each resource. To mitigate this disadvantage the developed agent has minimal software requirements (Python 2.6+) and is easy to deploy by an unprivileged user.

Besides standalone servers and clusters supported via the described agent, Everest also implements integration with the European Grid Infrastructure (EGI). A user can attach as a new resource a specific virtual organization within EGI by uploading a valid proxy certificate. This certificate is used by Everest to submit jobs to EGI on behalf of the user. The interaction with the grid is implemented via EMI User Interface (UI) which provides a standard set of commands for accessing EGI.

Figure 3 show an example of combined use of computing cluster and EGI grid via Everest for running a parameter sweep application from geophysics domain consisted of 670 tasks. Green and blue lines correspond to a number of tasks running on the cluster and the grid respectively over the course of the experiment.

Fig. 3 *Number of running tasks over the experiment using resources of local cluster and grid infrastructure.*

6. *Automation and Application Composition*

Running Everest applications via Web UI is easy and convenient, but it has some limitations. For example, if a user wants to run an application many times with different inputs, it is inconvenient to submit many jobs manually via web form. In other case, if a user wants to produce some result by using multiple applications, she has to manually copy data between several jobs. Finally, Web UI is not suitable if one wants to run an Everest application from his program or some other external application.

To support all these cases, from automation of repetitive tasks to application composition and integration, Everest implements a REST API. It can be used to access Everest applications from any programming language that can speak HTTP protocol and parse JSON format. However REST API is too low level for most of users, so it is convenient to have ready-to-use client libraries built on top of it. For this purpose a client library for Python programming language called Python API was implemented.

Figure 4 contains an example of program using Python API. It implements a simple diamond-shaped workflow (depicted in the top right corner of the picture) that consists of running four different applications – A, B, C and D.

```
import everest

session = everest.Session(
  'https://everest.distcomp.org', token = '...'
)

appA = everest.App('52b1d2d13b...', session)
appB = everest.App('...', session)
appC = everest.App('...', session)
appD = everest.App('...', session)

jobA = appA.run({'a': '...'})
jobB = appB.run({'b': jobA.output('out1')})
jobC = appC.run({'c': jobA.output('out2')})
jobD = appD.run({'d1': jobB.output('out'), 'd2': jobC.output('out')})

print(jobD.result())

session.close()
```

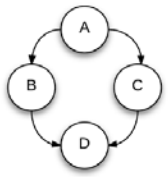

Fig. 4 *An example of wokflow described using Everest Python API.*

The nonblocking semantics of Python API, similar to the dataflow programming paradigm [9], has a number of advantages. It makes it simple to describe computational pipelines without requiring a user to implement boilerplate code dealing with waiting for jobs and passing data between them. This approach also implicitly supports parallel execution of independent jobs such as *jobB* and *jobC* in the presented example.

7. *Running Parameter Sweep Experiments*

Parameter sweep applications (PSA) represent an important class of computational applications that require a large amount of computing resources in order to run a large number of similar computations across different combinations of parameter values. These applications are becoming extremely important in science and engineering. While PSAs can be extremely time-consuming and require enormous amount of processor time, the individual tasks are independent and can be run in parallel. Therefore, this class of applications is naturally suited for distributed computing. The potential speedup that can be achieved by running PSA across distributed computing resources is significant. However, the heterogeneous and complex nature of such environments requires the use of high-level tools that automate task submission, scheduling, data movement and failure recovery.

In order to facilitate running PSAs on distributed computing resources, a generic web service called Parameter Sweep was developed [10]. It is implemented as an Everest application following the abstract model and using mechanisms described above. The architecture of service implementation and its interaction with Everest platform is presented in Figure 5.

At the core of the developed service is a declarative format for describing a parameter sweep experiment. In order to run an experiment, a user should prepare and submit its description in the form of a plain text file called plan file. This file contains parameter definitions and other directives that together define rules for generation of parameter sweep tasks and processing of their results by the service. This approach aims to solve a common problem faced by researchers trying to use general-purpose computing tools and environments for running PSAs. Namely, a user have to implement custom programs for generating individual tasks comprising PSA and processing their results. The use of declarative description enables users to minimize or completely avoid such programming work, thus increasing the productivity and accessibility of the developed service.

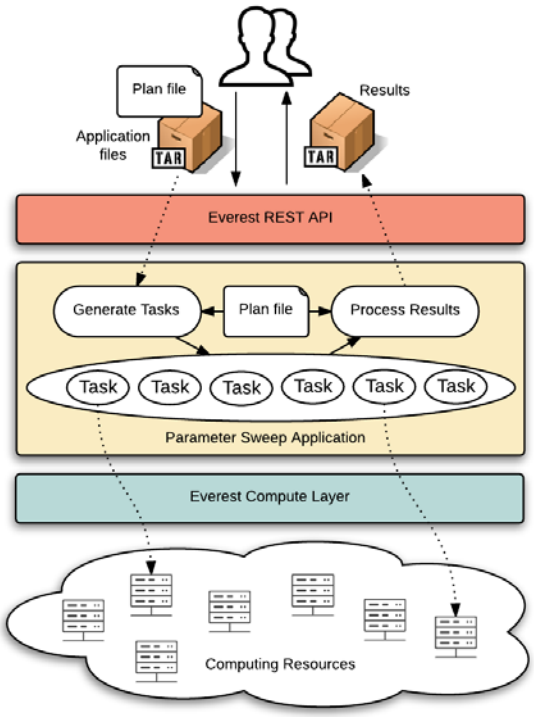

Fig. 5 *Architecture of Parameter Sweep services.*

```
parameter n from 1 to 100 step 1

input_files @run.sh vina
write_score.py protein.pdbqt
input_files ligand${n}.pdbqt
config.txt

command ./run.sh

output_files ligand${n}_out.pdbqt
log.txt @score

criterion min $affinity
```

Fig. 6 *An example of plan file with PSA description.*

An example of plan file is presented in Figure 6. The presented plan file corresponds to a virtual screening experiment using Autodock Vina, a well-known program for molecular docking. This experiment performs 1000 docking runs with different ligand molecules against the same protein molecule and selects results with a minimum affinity (binding energy) value.

The Parameter Sweep service has two inputs. The first input represents the plan file described above. The second optional input represents an archive with application executables, scripts and other input files referred in the plan file. Upon job submission the user should also explicitly specify resources to be used for running the experiment. These resources should be attached to Everest as described previously.

Upon submission of a new job via REST API the service parses the submitted plan file and generates job tasks representing parameter sweep experiment. In order to do this, it takes cartesian product of parameters specified in the plan file subject to specified constraint directives. Then the service passes the generated tasks to the Compute layer of Everest which performs scheduling and execution of tasks on specified resources.

Upon completion of individual tasks the service extracts task results and performs additional filtering in accordance with post-processing directives specified in the plan file. After all tasks are completed the service produces a single output which refers to an archive containing the results of the filtered tasks.

8. Conclusion

We have discussed common problems associated with the automation of scientific and engineering computations and presented approaches for solving these problems based on cloud computing models and implemented in the Everest platform. We have demonstrated that the use of service-oriented approach in scientific and engineering computing can improve the research productivity by enabling publication and reuse of computing applications, as well as creation of cloud services for automation of computation processes.

Future work will address the remaining challenges, such as implementation of advanced scheduling across multiple resources, integration with other types of computing resources and supporting secure and portable applications. We also plan to extend the types of applications that users can publish on Everest by providing native support for parallel, many-task and data-intensive applications.

References

1. Foster, I.: Service-Oriented Science. Science, vol. 308, no. 5723, pp. 814–817 (2005)

2. Sukhoroslov O., Volkov S., Afanasiev A. A Web-Based Platform for Publication and Distributed Execution of Computing Applications // 14th International Symposium on Parallel and Distributed Computing (ISPDC). IEEE, 2015, pp. 175-184.

3. Everest Web Site. http://everest.distcomp.org/

4. Richardson L., Ruby S. RESTful Web Services. O'Reilly, 2007.

5. Afanasiev A., Sukhoroslov O., Voloshinov V. MathCloud: Publication and Reuse of Scientific Applications as RESTful Web Services // Lecture Notes in Computer Science Volume 7979. Springer 2013. pp. 394-408

6. P. Kacsuk, "P-grade portal family for grid infrastructures," Concurrency and Computation: Practice and Experience, vol. 23, no. 3, pp. 235–245, 2011.

7. Foster I., Kesselman C. (ed.). The Grid 2: Blueprint for a new computing infrastructure. – Elsevier, 2003.

8. Buyya R., Abramson D., Giddy J. Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid //High Performance Computing in the Asia-Pacific Region, 2000. Proceedings. The Fourth International Conference/Exhibition on. – IEEE, 2000. – Т. 1. – С. 283-289.

9. W. M. Johnston, J. Hanna, and R. J. Millar, "Advances in dataflow programming languages," ACM Computing Surveys (CSUR), vol. 36, no. 1, pp. 1–34, 2004.

10. Volkov S., Sukhoroslov O. A Generic Web Service for Running Parameter Sweep Experiments in Distributed Computing Environment. Procedia Computer Science, Volume 66, 2015, pp. 477-486.