AUTOMATION OF SCIENTIFIC AND ENGINEERING COMPUTATIONS WITH EVEREST PLATFORM

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

Abstract: *The complexity of high-performance computing infrastructures and applications require the use of problem-oriented interfaces and automation of basic activities in order to support the solving of complex scientific and engineering problems. The use of service-oriented approach and cloud computing models can improve the research productivity by enabling publication and reuse of computing applications, as well as creation of cloud services for automation of computations. The paper describes the implementation of this approach in the form of a web-based platform following the Platform as a Service model. Unlike other solutions, the presented Everest platform runs applications on external computing resources connected by users, implements flexible binding of resources to applications and provides an open programming interface.*
KEYWORDS: *RESEARCH AUTOMATION, CLOUD, SERVICES, DISTRIBUTED COMPUTING, WORKFLOW, PARAMETER SWEEP*

1. Introduction

Computational methods and high-performance computing resources are now widely used for solving complex scientific and engineering problems. However, the inherent complexity of corresponding software and infrastructures, along with the lack of required IT expertise among the researchers, require the use of problem-oriented interfaces and automation of basic activities in order to support the process of problem solving. The examples of such activities include 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.

The use of service-oriented approach and cloud computing models can improve the research productivity by enabling publication and reuse of computing applications, as well as creation of cloud services for automation of the mentioned activities [1]. The paper describes the implementation of this approach in the form of a web-based platform following the Platform as a Service model. The implementation of the platform and its use for automation of computing activities are discussed.

2. Everest Platform

Everest [2] is a web-based distributed computing platform. It provides users with tools to quickly publish and share computing applications as services. The platform also manages execution of applications on external computing resources attached by users. In contrast to traditional distributed computing platforms, Everest implements the Platform as a Service (PaaS) model by providing 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. The platform is publicly available online to interested users [3].

A high-level overview of Everest architecture 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 the web user interface (Web UI) and client libraries.

Applications are the core entities in Everest that represent reusable computational units that follow a well-defined model. An application has a number of *inputs* that constitute a valid request to the application and a number of *outputs* that constitute a result of computation corresponding to some request. Upon each request Everest creates a new *job* consisting of one or more computational *tasks* generated by the application according to the job inputs. The tasks are executed by the platform on computing resources specified by a user. The dependencies between tasks are currently managed internally by applications. The results of completed tasks are passed back to the application and are used to produce job outputs or new tasks if needed. The job is completed when there are no incomplete tasks are left. The described model is generic enough to support a wide range of computing applications.

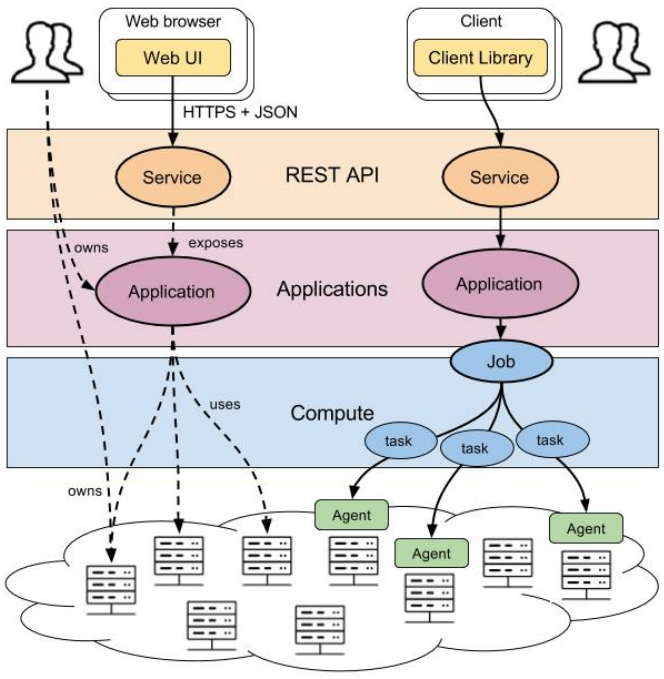


Fig. 1 *Everest Architecture.*

Users can publish applications via provided generic application template that makes it possible to avoid programming. The template supports running arbitrary applications with command-line interface and produces a single task corresponding to a single command run. Recent developments made it possible to dynamically add new tasks or invoke other applications from a running application via the Everest API. This enabled users to create and publish complex many-task applications with dependencies between tasks, such as workflows. Everest also provides a built-in generic application for running parameter sweep experiments consisting of a large number of independent compute tasks discussed in Section 3.

An application is automatically published as a RESTful web service with a unified interface. This enables programmatic access to applications, integration with third-party tools and composition of applications into workflows as discussed in Section 3. The platform's web user interface also generates a web form for running the application via web browser. 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. The application owner can manage the list of users that are allowed to run the application.

Instead of using a dedicated computing infrastructure, Everest performs execution of application tasks on external resources attached by users. The platform implements integration with standalone machines and clusters through a developed program called *agent*. The agent runs on the resource and acts as a mediator between it and Everest enabling the platform to submit and manage computations on the resource. Everest also supports integration with other types of resources, such as grid infrastructures.

Everest users can flexibly bind the attached resources to applications. In particular, a user can specify multiple resources for running an application, thus seamlessly combining these resources into a single computing pool [4]. Everest manages execution of tasks on remote resources and performs routine actions related to staging of input files, submitting a task, monitoring a task state and downloading task results. The platform also monitors the state of resources and uses this information during scheduling.

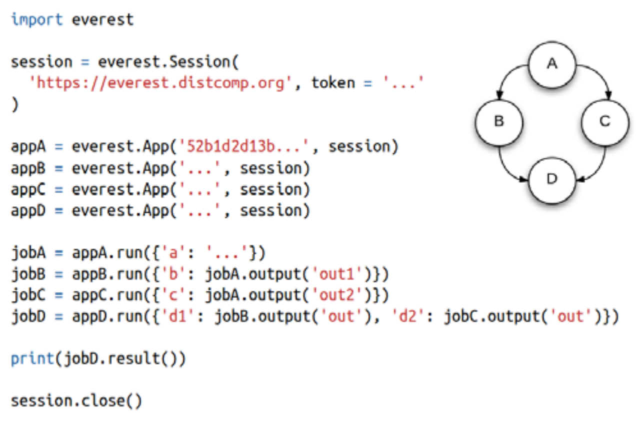
3. *Automation of Many-Task Computations*

Automation and application composition via Everest API

Running Everest applications via the platform's web interface 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 a 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, the web interface is not suitable if one wants to run an 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 2 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. 2** An example of workflow described using Everest Python API.*

The nonblocking semantics of Python API, similar to the dataflow programming paradigm, has a number of advantages. It makes it simple to describe computational pipelines without requiring a user to implement the 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.

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. 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.

In order to facilitate running PSAs on distributed computing resources, a generic web service called Parameter Sweep was developed on Everest [5]. At the core of the 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.

Upon job submission the user should upload the plan file and application files, and specify resources to be used for running the experiment. Upon submission the service parses the submitted plan file and generates job tasks representing parameter sweep experiment. 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 an output archive with results of the filtered tasks which can be downloaded by the user.

4. *Conclusion*

The paper described the implementation of Everest platform and its use for automation of computations. The platform, based on service-oriented approach and cloud computing models, enables publication and reuse of computing applications, as well as creation of generic services for automation of common activities, such as parameter sweep experiments. Unlike other solutions, the platform runs applications on external computing resources connected by users, implements flexible binding of resources to applications and provides an open programming interface.

References

1. Sukhoroslov O., Putilina E. Cloud Services for Automation of Scientific and Engineering Computations // Science. Business. Society. Issue 2, 2016, pp. 6-9.

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. Sergey Smirnov, Oleg Sukhoroslov, and Sergey Volkov. Integration and Combined Use of Distributed Computing Resources with Everest // Procedia Computer Science, Volume 101, 2016, pp. 359-368.

5. 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.