JOINT INSTITUTE FOR NUCLEAR RESEARCH
Laboratory of Information Technologies

FINAL REPORT ON THE SUMMER STUDENT PROGRAM

Increasing efficiency of IaaS-clouds by loading idle capacities with batch jobs

Supervisors:
Nikolay Kutovskiy
Nikita Balashov

Student:
Ruslan Kuchumov, Russia
Saint Petersburg State University

Participation period:
July 01 - August 26

Dubna, 2017

Abstract

This final report covers the project I've been working on during Summer Student Program at JINR. It is a software system called "Smart-Scheduler Idle-Utilizer" which goal is to increase the efficiency of Infrastructure as a service (IaaS) cloud by occupying its idle resources with batch jobs. This report describes project domain knowledge, its architecture including all its components, main algorithms, and user interfaces and also presents instruction on installation and usage.

Table of contents

Introduction

JINR has a cloud computing infrastructure supported by LIT. It's used for the variety of different purposes but mainly for scientific computations. For example, JINR participates in experiments such as BES-III or NOvA, which loads fair share of resources presented by the cloud with computational tasks [1].

JINR cloud is based on OpenNebula platform which orchestrates virtual machines, storage, and networking on the resources of distributed infrastructure [2].

NOvA experiment require JINR cloud to be integrated into NOvA infrastructure through Open Science Grid. This infrastructure defines the stack of technologies for sharing computation and storage resources, and interconnection protocols as well. In particular, HTCondor batch task scheduler is used for distributing, executing and monitoring computational tasks.

The goal of this project was to create a software system that would load idle resources of a cloud with tasks from the batch scheduler. It had to be a background process that would periodically check whether or not there are pending tasks in scheduler's queue. Depending on that it would spawn or terminate virtual machines in the cloud. In turn, these virtual machines would automatically load scheduler software and would be ready to execute these tasks.

As a result, the desired software system was designed, implemented and tested in a test cloud environment.

Domain Knowledge

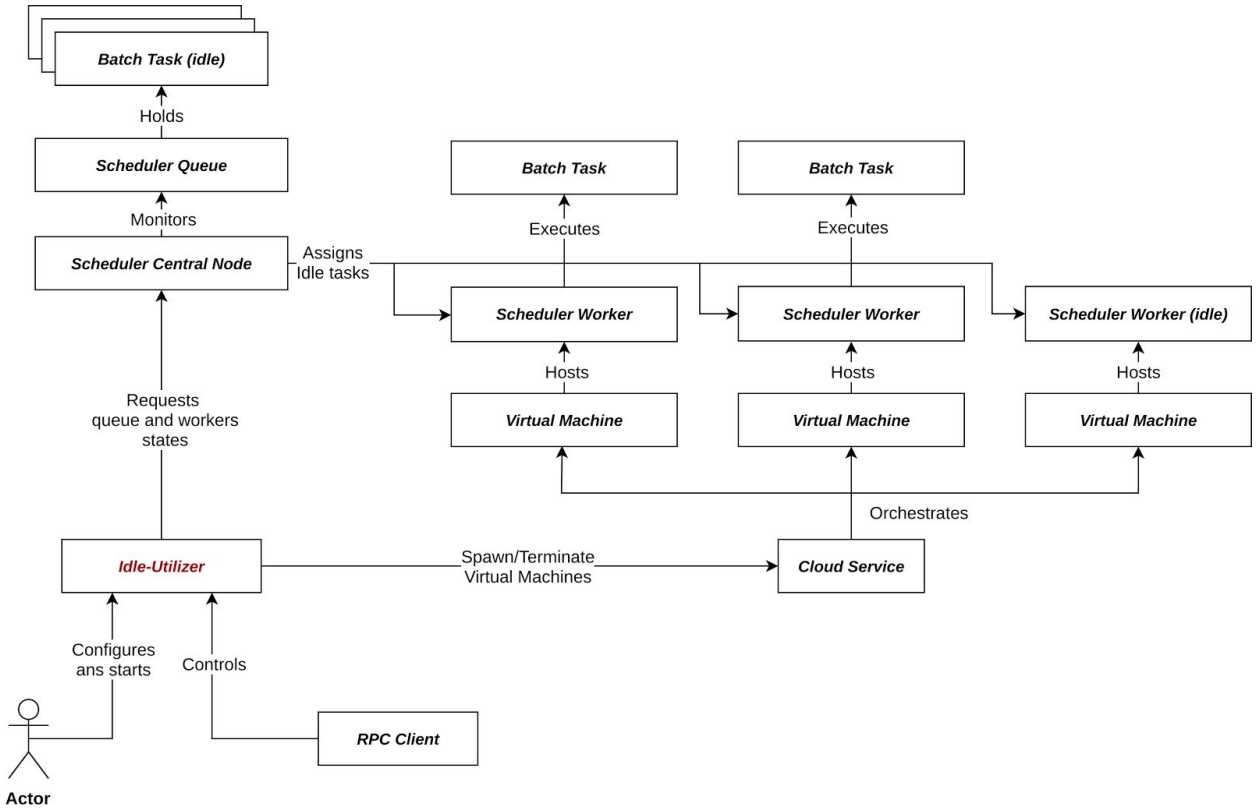

Figure 1. Domain knowledge diagram

The main components of domain knowledge are:

- **Batch task scheduler** – a software for controlling unattended execution of tasks.
    - **Batch task** – unit of execution created and submitted to the scheduler from outside world. The task is called “idle” when it’s not assigned to any host.
    - **Queue** – a container (usually FIFO) of idle batch tasks.
    - **Central node** – the main node of a scheduler which is responsible for monitoring the queue, assigning idle tasks and providing an interface for controlling the scheduler
    - **Worker node** – a node running scheduler software and that is responsible for accepting and executing batch tasks. The host is called “idle” when it has no tasks assigned. All the worker nodes of a scheduler are called “host pool**”**.
- **Cloud service** – IaaS service that’s used for orchestrating virtual machines.
    - **Virtual Machines (VM)** – provides platform-independent environment for guest OS. There, they are expected to automatically appear in task scheduler’s host pool when booted and to be ready to accept its tasks.
- **RPC Client** – Any external service or a user that can send RPC request (for controlling Idle-Utilizer)

User scenario:

1. Installs the program and all its dependencies
2. Modifies program's configuration file
3. Start program in a background (daemon) process
4. All further interactions are expected to happen through program's RPC interface

When started program performs the following actions in a loop:

1. Fetches the number of idle tasks in scheduler queue
2. If there are idle tasks present in the queue, it sends a request to the cloud to deploy a new VM. When booted this VM shall appear in scheduler host pool automatically.
3. In case there are pending deploy requests already, Idle-Utilizer waits for new VM to appear in scheduler pool before performing any other actions.
4. If there are no idle tasks in the queue and some host are also idle, Idle-Utilizer removes one of these hosts from scheduler's pool and terminates corresponding VM in the cloud.

Architecture

Idle-Utilizer components were designed to be flexible, reusable and testable which allows for changes in external services and possible updates of project requirements. Right now it supports only OpenNebula as cloud service and HTCondor task scheduler.

Components and Modules

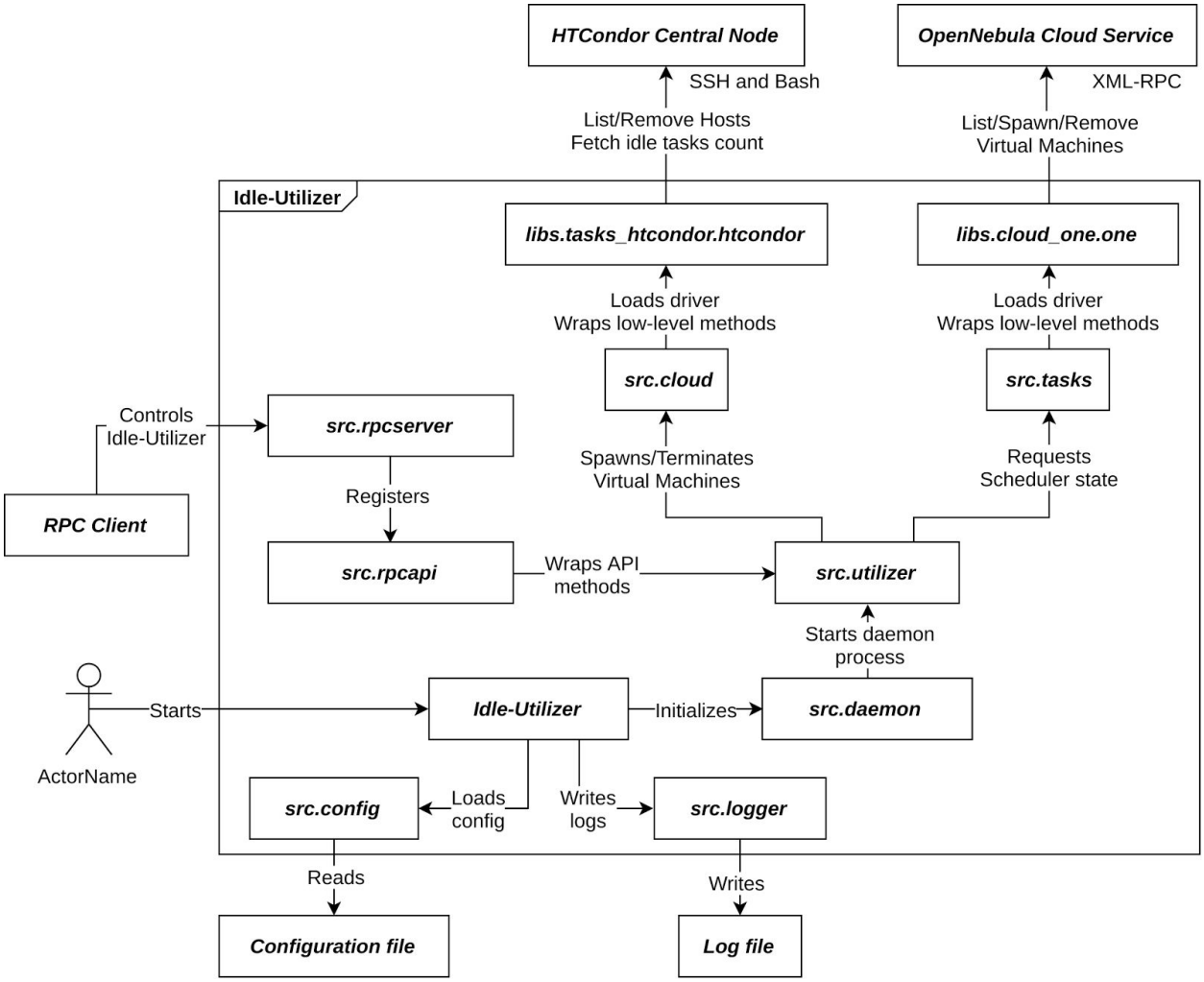

Figure 2. Project components architecture

Idle-Utilizer includes the following components:

- **src.config** – responsible for reading specified parameters from the configuration file and for presenting them in a form that can be used by other components.
- **src.logger** – initializes and configures a logger object that writes messages to the log file.
- **src.daemon** – allows to continue the flow of execution in a daemon process
- **src.rpcapi** – defines RPC API methods and wraps methods of other modules which are presented to API
- **src.rpcserver** – responsible for spawning and joining the thread with XML-RPC server and for registering its API

- **libs.tasks_htcondor.htcondor** – HTCondor stateless driver. Encapsulates low-level and scheduler-specific methods. Communication with HTCondor hosts is done by executing BASH scripts over SSH.
- **src.tasks** – responsible for loading tasks scheduler driver, wraps its API and provides convenience functions for controlling the scheduler.
- **libs.cloud_one.one** – OneNebula stateless driver. Encapsulates low-level and api-specific methods. Communication with OpenNebula is done by accessing its XML-RPC API methods.
- **src.clouds** – responsible for loading cloud driver, wraps its API and provides convenience functions for controlling the cloud.
- **src.utilizer** – implements the main logic of spawning and terminating VM depending on scheduler state.

Main Algorithms

Spawning a new host

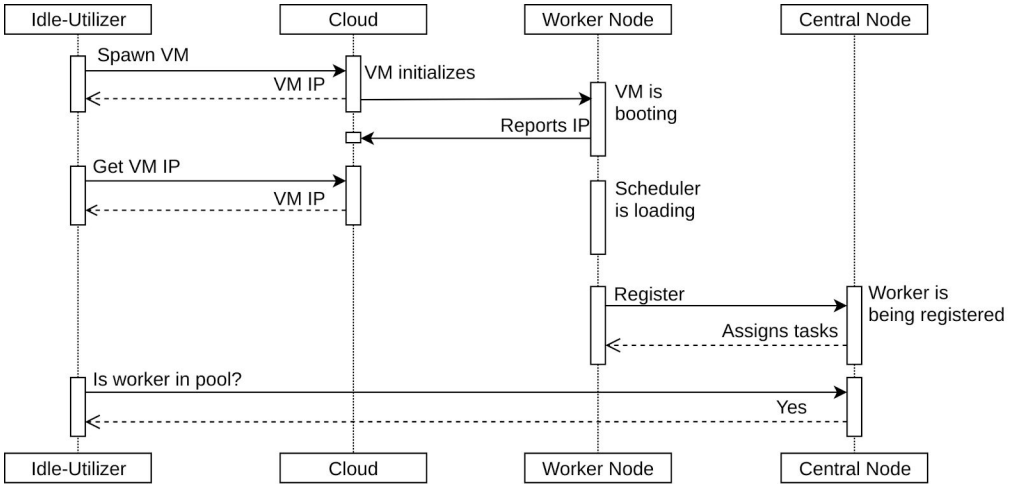

Figure 4. Spawning a new host sequence diagram.

Removing idle host

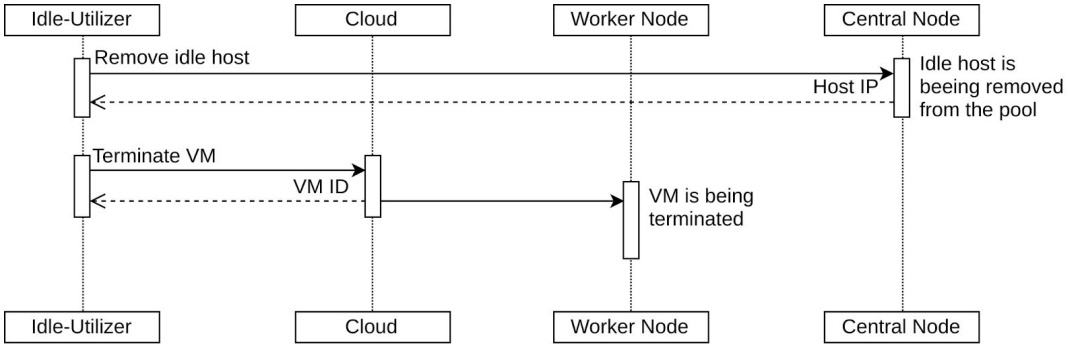

Figure 5. Removing host sequence diagram

A single iteration of Idle-Utilizer loop

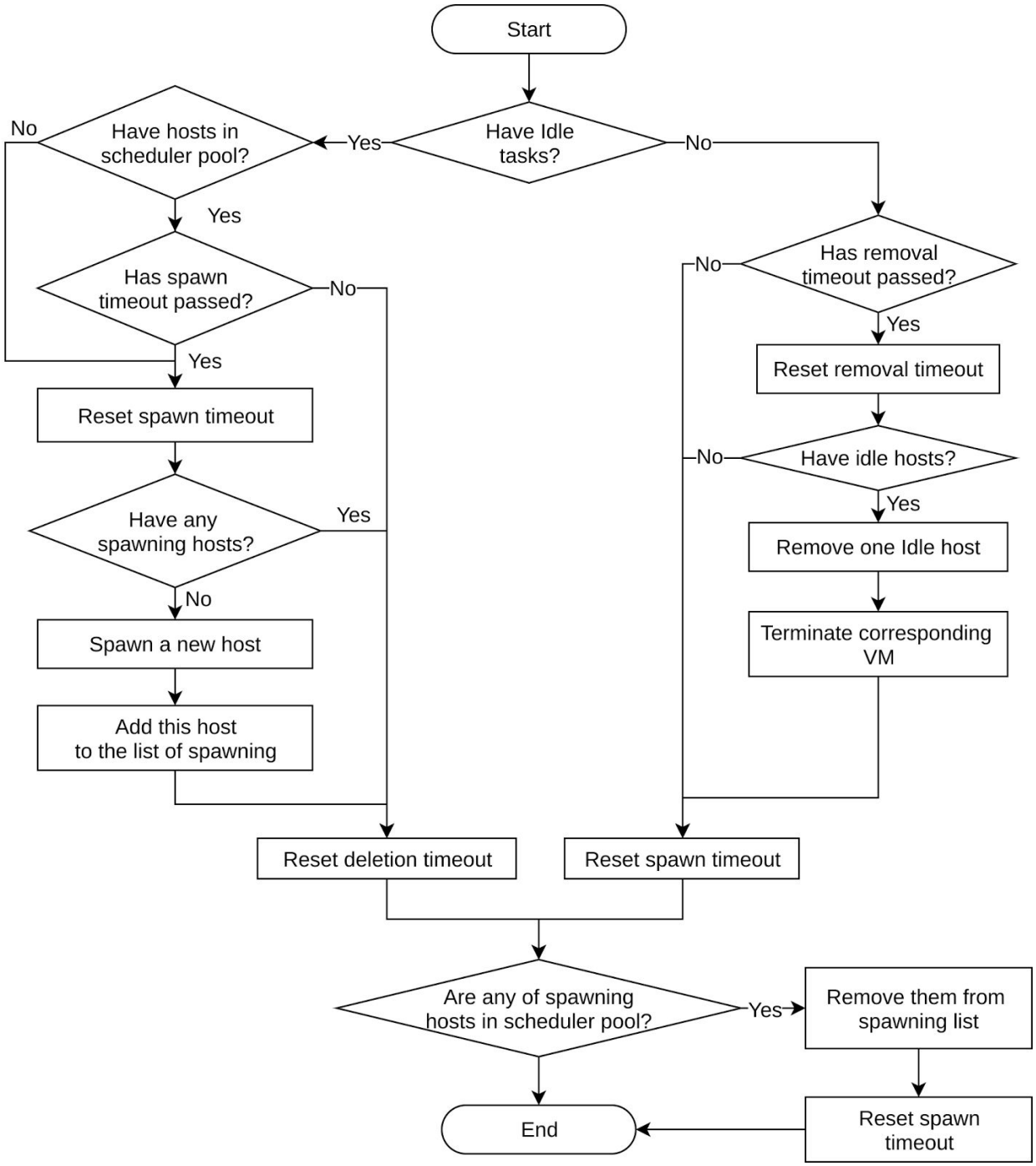

Figure 3. Flowchart of a single iteration of Idle-Utilizer loop

User Interface

Command line Interface

The first and the only command line argument specifies configuration file. In case it's not set, Idle-Utilizer would use default located at `/etc/smartscheduler/config.cfg`.

Configuration file parameters

Configuration file is expected to be in the version of INI format supported by ConfigParser Python 3.6 module. Below is a list of configuration file section with it parameters:

- **[cloud]** – Cloud wrapper settings
  - *driver* – cloud driver to load. Right now supports only libs.cloud_one.one
- **[cloud.one]** – OpenNebula cloud settings.
  - *endpoint* – address of OpenNebula RPC server
  - *one_auth* – authorization string (user:password)
  - *spawn_rpc_{method,args,filter}* – RPC method, arguments and result filter for spawning a new host
  - *remove_rpc_{method,args,filter}* – RPC method, arguments and result filter for removing specified host by its id
  - *get_host_rpc_{method,args,filter}* – RPC method, arguments and result filter for fetching a list of dicts containing active host ID and IP
  - Note: **_filter* is specified in JMESPath format
- **[tasks]** – Tasks scheduler wrapper settings (src/cloud.py)
  - *driver* – driver to load. Right now supports only libs.tasks_htcondor
- **[tasks.htcondor]** – HTCondor tasks scheduler settings (libs/taks_htcondor/htcondor.py)
  - *command_template* – allows to wrap all other commands in, for example, ssh calls
  - *command_get_hosts* – the command for listing hostname, status and the address of all machines in the pool
  - *command_get_idle_tasks_count* – this command is expected to return the number of idle tasks in queue (a single integer)
  - *command_remove_idle_host* – this command is called when Idle-Utilizer tries to remove any idle host. It's expected to return IP of this host on success or nothing otherwise.
- **[rpcserver]** – XML-RPC server settings
  - *path* – URL for receiving XML-RPC requests
  - *host, port* – server address
  - *log_requests* – if True, incoming requests will be logged
  - *whitelist* – A list of IP addresses which are allowed to access XML-RPC server. Empty list ([ ]) allows to connect from any IP address
- **[utilizer]** – Idle-Utilizer main logic settings

    - *interval* – the number of seconds to sleep between iterations of utilizer loop
    - *remove_timeout* – the number of seconds to wait before trying to remove idle host from scheduler pool and cloud
    - *spawn_timeout* – the number of seconds to wait before trying to spawn a new host

RPC Interface

Idle-Utilizer starts XML-RPC server in the background providing an API for external services. Right now it supports the following methods:

- *remove_random_host()* – Removes one random host from task scheduler pool and terminates corresponding VM

Code structure

Version control is done through git: https://git.jinr.ru/cloud-team/SmartScheduler-IdleUtilizer

Project directories (in bold) and files structure:

- **libs/** – services drivers and third-party libraries
    - **cloud_one/** – OpenNebula driver sources
    - **tasks_htcondor/** – HTCondor driver sources
- **src/***.py – internal modules sources
- **tests/** – unit tests files
    - **libs/**test_*.py – unit tests for drives and third-party libraries
    - **src/**test_*.py – unit tests for internal components
    - nose.cfg – configuration file for unittest utility (nosetests)
- **scripts/** – helper scripts required by third-party libraries and misc scripts
    - **condor-remove-host.bash** – a script for removing hosts from HTCondor pool. Used by libs.tasks_htcondor.htcondor module
- config.cfg – Idle-Utilizer configuration file
- idle-utilizer.py – Idle-Utilizer main executable file
- setup.py – setup script for installation

Installation and Usage

To install Idle-Utilizer run the following command from project root directory:

```
sudo python3 setup.py install
```

To execute all unit tests, you'll need to install "nosetests" Python package and then execute the following commands:

```
cd tests/
nosetests -c nose.cfg
```

To start Idle-Utilizer execute:

```
python3 idle-utilizer.py
```

It'll use default configuration file located at "/etc/smartshed/config.cfg".

Conclusion

As a result, the desired software system was designed, implemented and tested in a test cloud environment. All of the current requirements were met and it is ready for further improvements, optimizations and fine-tuning for production environment.

References

1. N. Balashov, A. Baranov, V. Korenkov, N. Kutovskiy, A. Nechaevskiy, R.Semenov. JINR Cloud Service: Status and Perspectives. Trudy Instituta sistemnogo programmirovania RAN [Proceedings of the Institute for system programming of the RAS], 2015, vol. 27, issue 6, pp. 345-354
2. OpenNebula Release Notes 5.0 / OpenNebula Project URL: http://docs.opennebula.org/5.0/intro_release_notes/release_notes

Acknowledgments

I would like to thank N. Kutovskiy and N. Balashov for mentoring my work and especially A. Baranov for spending some time on testing this program in the cloud environment.