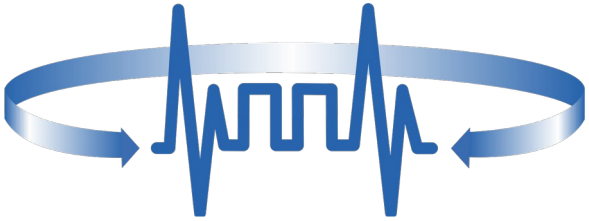
The 10th International Conference "Distributed Computing and Grid Technologies in Science and Education" (GRID'2023)

A comprehensive approach to running optimization workflows in the Everest platform

Sergey Smirnov

Center for Distributed Computing,
Institute for Information Transmission Problems
of the Russian Academy of Sciences (Kharkevich Institute)



This work was supported by the Russian Science Foundation under grant no. 22-11-00317

Optimization Workflow

Optimization subsystem:

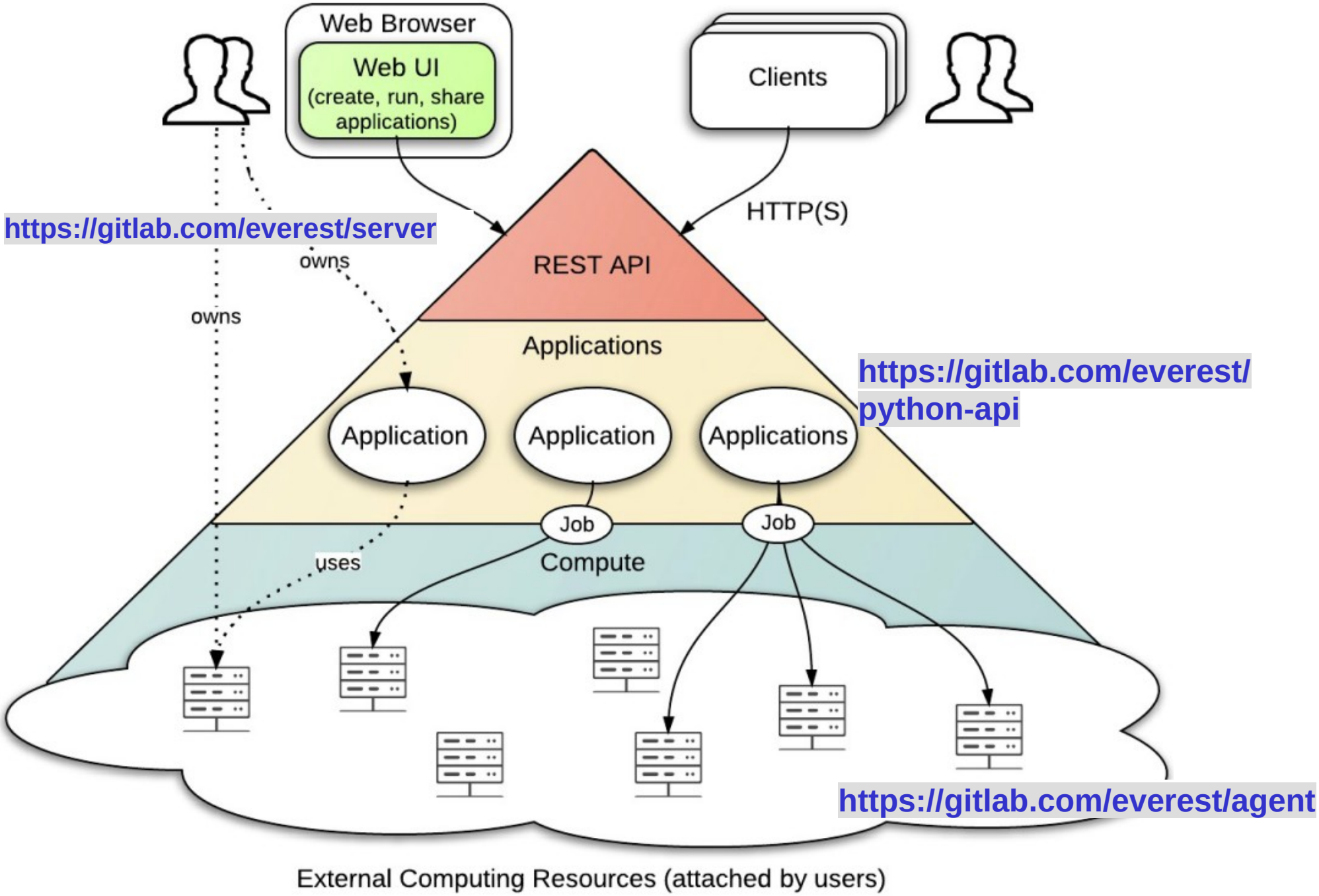
- Gets input data
- Checks data consistency
- Prepares data for further optimization steps
- Performs one or more optimization steps (build and solve a model with a solver)
- Saves results for further processing

It is good to be able to vary:

- Settings
- Version
- Data

Everest Web-based Platform, everest.distcomp.org

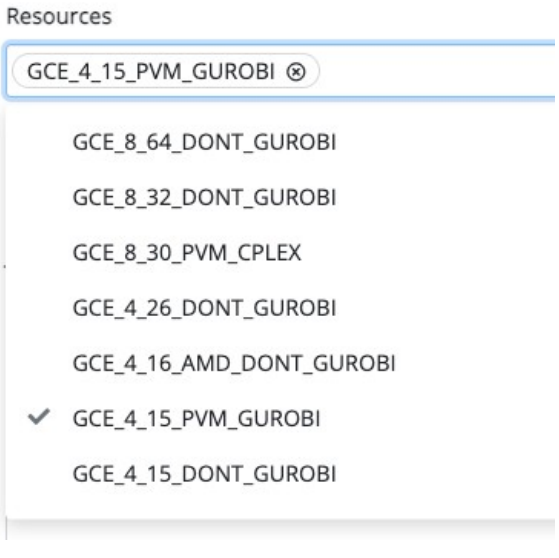
Describe/Develop/Deploy REST-services representing existing applications

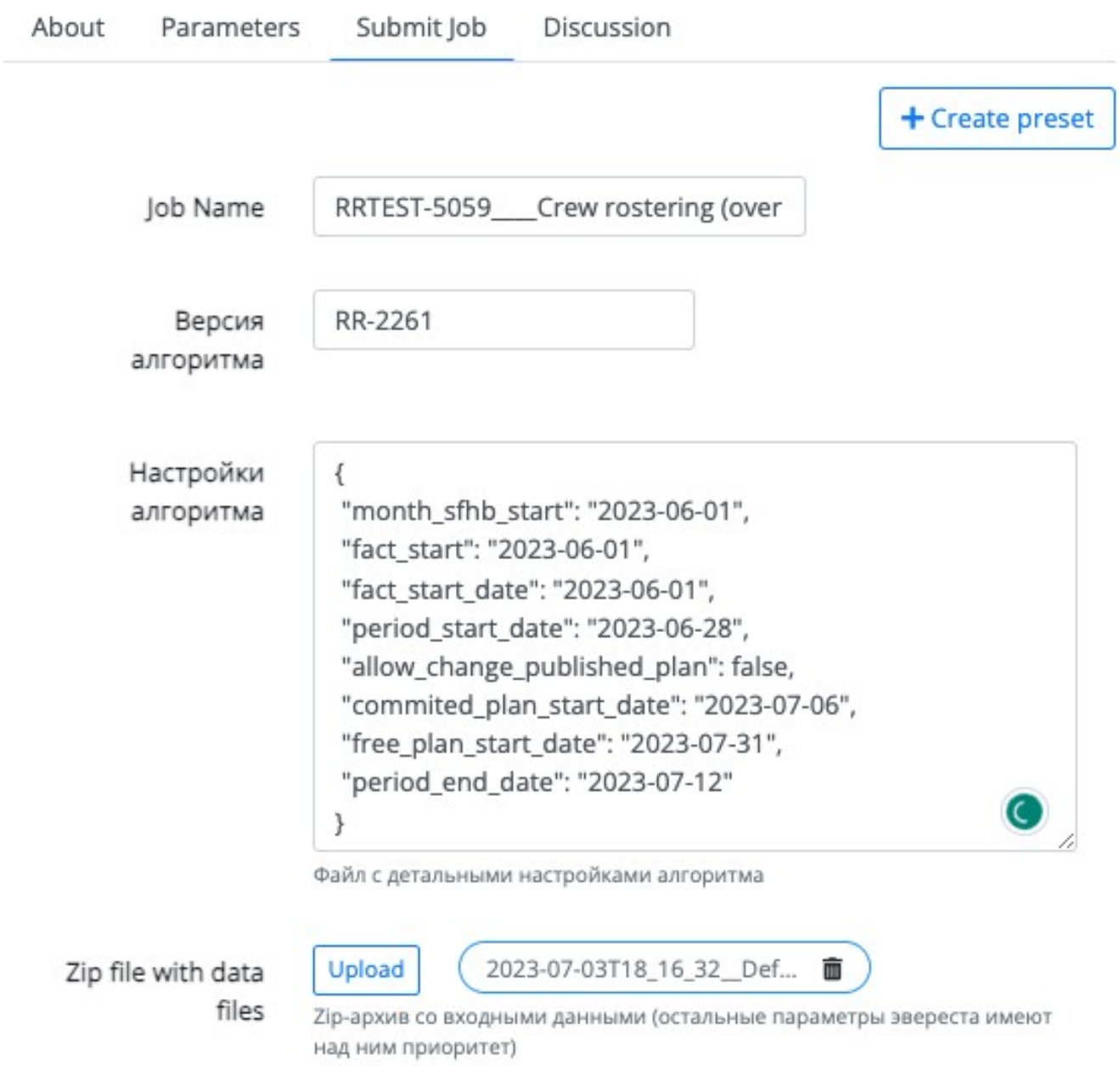


Everest Job Inputs

Inputs:

- Model's version
- JSON parameters (editable text field)
- Input data as archive
- Computing resource can be selected



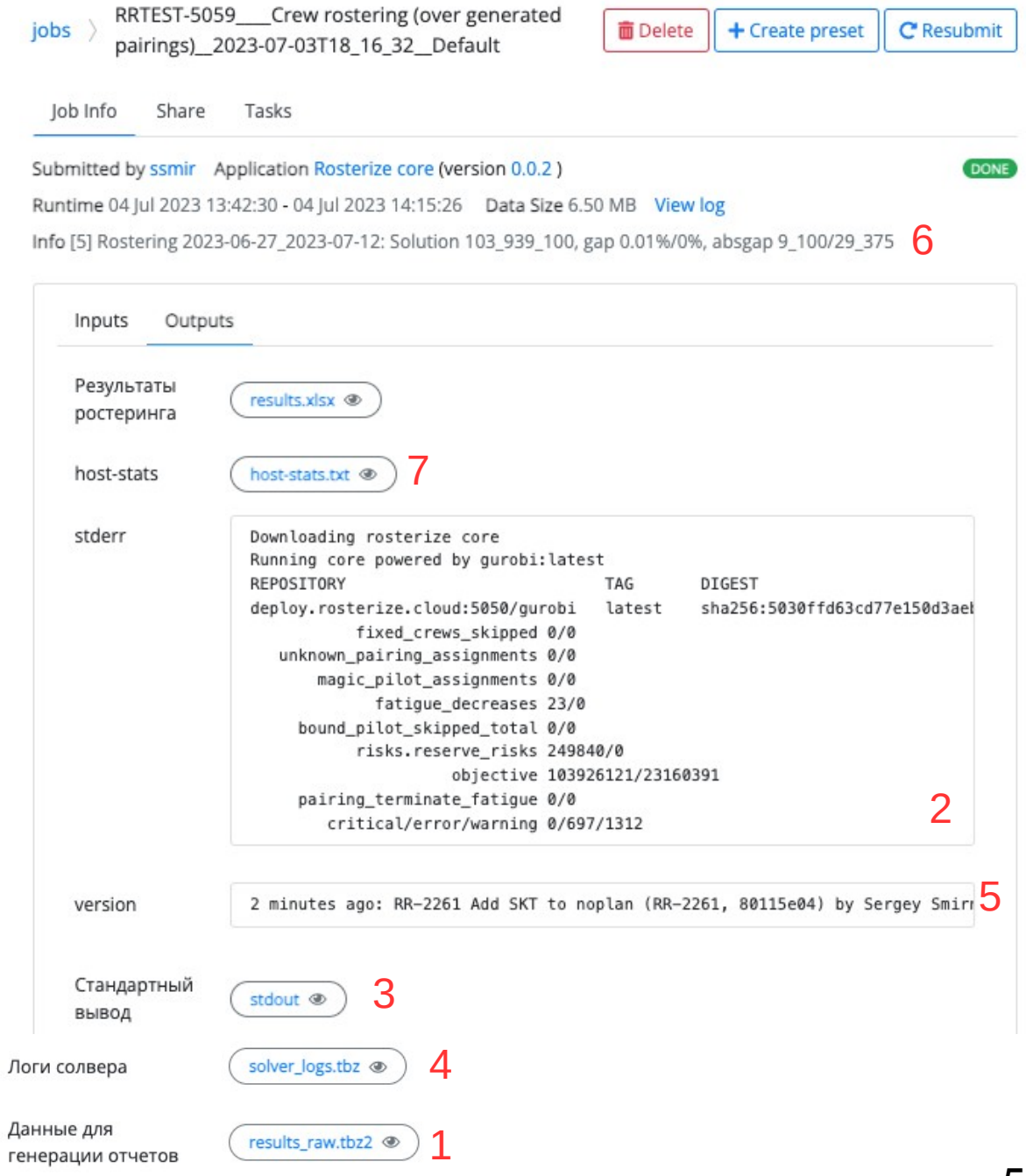


Everest Job Outputs

Outputs:

1. Results as archive
2. stderr (summary)
3. stdout viewed in one click
4. Archive with files for debugging purposes (controlled by parameter)
5. Model's version
6. Last status message
7. CPU and memory statistics collected by a simple script

Resubmit button is very handy



Everest Application Settings

Medatada | Description | Parameters | Configuration | Files | Resources | Access | Versions

Inputs + Add input

- version string
- profile string — Файл с детальными настройкам...
- zip-file string — Zip-архив со входными данным...

Outputs + Add output

- version string — Rosterize core version
- results_raw_tbz2 string — Данные для генерации отчетов ...
- host-stats string — Host cpu and memory load stats
- solver_logs string — Архив с логами солвера
- stdout string
- stderr string

Medatada | Description | Parameters | Configuration | Files | Resources | Access | Versions

Command

```
/bin/bash bootstrap.sh "${version}" "" "" 1 1 1 false 100 false 1 RR 0 "" 1 false "" "" "" "" "" "${version}"
```

Refer to input values as ${param}

Input Mappings + Add input mapping

profile	profile.json
pattern	
zip-file	data.zip
pattern	

Output Mappings + Add output mapping

stderr	stderr
stdout	stdout
solver_logs	solver_logs.tbz
host-stats	host-stats.txt
version	rosterize-version.txt
results_raw_tbz2	results_raw.tbz2

Medatada | Description | Parameters | Configuration | Files | Resc

Upload

bootstrap.sh
1.25 KB
Public
Stage in

Medatada | Description | Parameters | Configuration | Files | Resources | Access | Versions

Allow List: @rosterize

Allow Edit List:

Jobs Auto-Share: @rosterize_debug

Comma-separated list of entries, eg. "entry, @groupEntry".

Public

Everyone can see the application in applications list.

Enable Workflow (experimental)

Enable this application to invoke other applications on behalf of user (workflow).

Files Monitoring

Model Deployment (versioning)

Model's source code can be stored on GitLab.com.

REST API: https://docs.gitlab.com/ee/api/api_resources.html

Token is needed to access a private repo.

Useful API endpoints:

- Download source code for a commit (rate limited):
  GET /projects/:id/repository/archive[.format]?sha=$COMMIT
- Download CI/CD job's artifact file (no rate limit):
  GET /projects/:id/jobs/artifacts/:ref_name/raw/*artifact_path?job=name
- Get commit's metadata:
  GET /projects/:id/repository/commits/$COMMIT

Cloud Resources in Everest

Everest creates VMs on demand and kills them if idle.

Supported providers:

- Google Cloud Platform
- Yandex Cloud

Preemptible instances are supported (much cheaper)

VM image:

- Can be set up using Packer
- Needs Python version supported by Everest Agent (2.7, 3.6–3.9) + required modules
- Python modules and solvers for the model
- Can include tokens required to download model's source code

Helper scripts

bootstrap.sh script:

- Stored in Everest application files
- Downloads model's source code
- Runs run-task.sh script from the model
- Token to download source code can be stored there or in image's environment variables

run-task.sh script:

- Prepares human-readable version string
- Unarchives input files
- Sets up necessary environment variables
- Converts parameters to command line args of the model
- Runs model and auxiliary processes
- Collects output files

Live status

Everest task messages are used to update job's status

Environment variables:

- EVEREST_AGENT_PORT
- EVEREST_AGENT_ADDRESS
- EVEREST_AGENT_TASK_ID

```
def connect(self):
    self.connected = False
    try:
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.connect((self.address, self.port))
        self.sock.sendall(struct.pack('>Ib', len(self.task_id) + 1, 0))
        self.sock.sendall(self.task_id.encode('utf-8'))
        self.connected = True
    except socket.error as err:
        print("Error connecting to Everest agent:", err)
        self.close()

def send(self, msg):
    sent = False
    while not sent:
        try:
            self.sock.sendall(struct.pack('>I', len(msg)))
            self.sock.sendall(msg.encode('utf-8'))
            sent = True
        except (RuntimeError, socket.error) as err:
            print("Error while sending status to Everest agent:", err)
            time.sleep(5)
            self.close()
            self.connect()
```

Conclusion

- Our setup allows both model debugging and production use
- Live calculation status is available
- Minimal costs: VM instances launched an destroyed on demand, preemptible instances can be used
- Model's source code can be stored outside of Everest and VM image

Thank you!

Sergey Smirnov,
IITP RAS (Kharkevich Institute)
sasmir@gmail.com