



**FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE**

ASSIGNMENT OF MASTER'S THESIS

Title: Productivity CLI for work with grid network Metacentrum
Student: Bc. Jakub Tuček
Supervisor: Mgr. Jan Blažek, Ph.D.
Study Programme: Informatics
Study Branch: Web and Software Engineering
Department: Department of Software Engineering
Validity: Until the end of summer semester 2019/20

Instructions

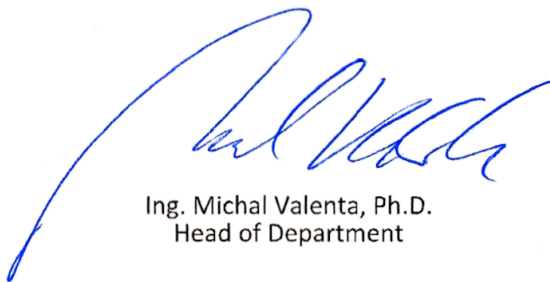
The main goal of the thesis is creation of an extension, a command line interface (CLI), for the job planing system in the grid network Metacentrum. The main part of this thesis is a software, which will simplify the Metacentrum usage in sense of hypotheses and algorithms testing. For a successful defence, common scenarios of a grid network usage must be done and a suitable generalization for solving common problems suggested:

- a) a configuration of computational nodes
- b) a versioning of source codes of algorithms as well as inputs and outputs
- c) a running job management and an ability to restart jobs on failure

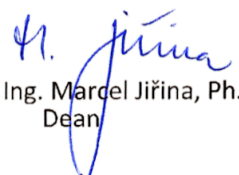
The CLI will simplify and make more efficient the usage of Metacentrum for hypotheses and algorithms testing.

References

Will be provided by the supervisor.



Ing. Michal Valenta, Ph.D.
Head of Department



doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague October 15, 2018



**FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE**

Master's thesis

Productivity CLI for work with grid network MetaCentrum

Bc. Jakub Tuček

Department of Software Engineering
Supervisor: RNDr. Jan Blažek, PhD.

May 7, 2019

Acknowledgements

I would like to thank my supervisor for his patience, consultations and needed notes. Important acknowledge also belongs to the MetaCentrum for providing access to the grid that was essential to develop and test the created tool.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the “Work”), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on May 7, 2019

.....

Czech Technical University in Prague

Faculty of Information Technology

© 2019 Jakub Tuček. All rights reserved.

This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Tuček, Jakub. *Productivity CLI for work with grid network MetaCentrum*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstract

In this thesis, we introduce a new command line interface for management of a grid computations. The proposed tool solves main weaknesses of the PBS planning system used in the MetaCentrum National Grid Infrastructure context.

We introduce a systematic view of the grid computation by adding a new level of abstraction, that allows solving detected bottlenecks in users workflow. Suggested solution offers features like versioning of executions, automatic re-submission or improved notifications.

Due to the highly specialized manner of this tool, users involvement in the CLI future development is discussed.

Keywords grid computing, hypothesis testing, command line interface, Meta-Centrum National Grid Infrastructure

Abstrakt

V této práci představujeme terminálovou aplikaci pro správu výpočtů na gridových sítích. Navržená aplikace řeší hlavní slabiny plánovacího systému PBS v kontextu Národní Gridové Infrastruktury MetaCentra.

Nová vrstva abstrakce umožňuje řešit problémy, které nejvíce omezují uživatele při gridovém počítání. Navržená aplikace řeší verzování spuštěných úloh, automatické znovu-spuštění výpočtů nebo vylepšené notifikace.

Protože CLI má (a bude mít) málo uživatelů z důvodu vysoké specializace, v závěru práce diskutujeme jejich zapojení do vývoje CLI.

Klíčová slova gridové sítě, počítání na gridových sítích, testování hypotéz, terminálová aplikace, Národní Gridová Infrastruktura MetaCentrum

Contents

Citation of this thesis	vi
Introduction	1
1 Grid computing	3
1.1 Computer cluster	3
1.2 Grid computing	3
1.3 Cluster vs Grid	4
1.4 Distributed computing applications	4
1.4.1 Portable Batch System	5
1.5 Virtual organizations	5
1.6 MetaCentrum National Grid Initiative	6
1.6.1 National participation efforts	6
1.6.2 MetaCentrum NGI details	7
1.7 Academic cluster services	8
1.7.1 Information systems and Technologies of CTU Super- computer	8
1.7.2 Technical University of Ostrava – IT4Innovations	9
1.7.3 Summary of academic cluster services	9
1.8 Cloud based services	11
1.8.1 Amazon Web Services Lambda	12
1.8.2 Amazon Web Services Fargate	12
1.8.3 Google Cloud Functions	13
1.8.4 Azure Functions	13
1.8.5 Summary of cloud-based services	13
2 MetaCentrum NGI example usage	15
2.1 Job lifecycle	15
2.2 Computational file	15
2.3 Example of usage	16

2.4	Versioning	16
2.5	Resource allocation	17
2.5.1	Hardware resources	17
2.5.2	Modules	18
2.5.3	Licenses and toolboxes	20
2.6	Notifications	21
2.7	Failed jobs and automatic resubmitting	21
3	Possible solution and analysis of CLI	23
3.1	Versioning of scripts and data	23
3.1.1	Versioning tools	24
3.1.2	Versioning summary	25
3.2	Allocation of resources	26
3.2.1	Virtual hardware resources	26
3.2.2	Software resources	26
3.3	Notifications	27
3.4	Failed jobs and automatic resubmitting	27
4	CLI structure and usage	29
4.1	Installation	29
4.2	Commands in the CLI	30
4.2.1	Configuration file	33
4.3	Storages	36
4.3.1	Metadata storage type	36
4.3.2	Storage type	37
4.4	User quick start	37
4.4.1	Quick guide	38
4.4.2	Detailed configuration guide	38
5	Technical details of realised CLI	41
5.1	Technology stack	41
5.1.1	Language and build system	41
5.1.2	File format	42
5.2	Implementation details	42
5.2.1	Notable interfaces	42
5.3	Local environment setup for development	47
5.3.1	Docker	49
5.3.2	Local environment solution	49
5.3.3	DEV profile for CLI	49
5.3.4	Docker configuration	50
5.4	Distribution	50
5.4.1	Wrapping execution script	50
6	Assessment of created CLI and future plans	53

6.1	Assessment of created CLI	53
6.1.1	Resource management	53
6.1.2	Notification and re-submission	54
6.1.3	Versioning	55
6.2	Further development plans	55
6.2.1	New releases and user participation	55
6.2.2	Resource profiles	55
6.2.3	Task types	56
6.2.4	Versioning	56
	Conclusion	57
	Bibliography	59
	A List of used terms	63
	B Content of attached CD	65

List of Figures

1.1	MetaCentrum NGI providers statistics based on provided number of CPUs [1]	8
1.2	Creation of new Function for AWS Lambda services via web interface	11
3.1	Versioning of large files using DVC.org, supported by Git server and external data storage [2]	25
4.1	Description of \$HOME/.clusterize folder structure	30
4.2	Activity diagram of submit command	31
4.3	Example of list command with debug output	32
4.4	Help command output	33
4.5	Metadata folder structure	37
4.6	Storage folder structure	38
5.1	High level view of submit action data flow	44
5.2	Class diagram for Action types	44
5.3	Time-flow visualization of Configurators execution before submit action	46
5.4	Class diagram for all submit TaskExecutors	48

List of Tables

1.1	Summary of main differences between clusters and grids [3]	4
1.2	Summary of basic PBSPro user commands [4, p. 50]	7
1.3	Queues available on MetaCentrum for all users. CPU/user represents maximum CPU cores per user	8
1.4	Overview of resources provided by CTU cluster	9
1.5	Queues available on Salomon super-computer [5]	10
1.6	Comparison of Microsoft azure plans [6] [7]	14
2.1	Overview of file types and their purpose	17
2.2	Overview of resource types with example value	20
5.1	Overview of notable interfaces	43

Introduction

Computation of hundreds or thousands of computation jobs on the grid brings specific issues for users, as the parallelization is more complex than programming itself. There are multiple ways how to parallelize a task. This thesis is focused on the one that splits the computation into hundreds or thousands of jobs, whose outputs are aggregated after their execution.

In this ecosystem, it is essential to introduce new systematic workflow, which deals with parametrization of jobs, the status of execution, user notifications or versioning. The thesis primary focuses issues occurring on the MetaCentrum grid network but the solution can be also applied to the alternative systems with identical environment.

This thesis presents an introduction to grid networks, clusters and supercomputers with focus on MetaCentrum National Grid Initiative operated by CESNET department. Both the academic and commercial alternative services are described and compared in terms of available resources and software that they offer.

The goal is to propose a possible solution and create a tool that allows systematic handling of the parallelization by splitting a computation into multiple jobs.

Assessment of the thesis is done based on implemented solution and how well it deals with the issues that are encountered while using the MetaCentrum grid network.

Grid computing

The thesis is dedicated to the analysis and solving issues occurring in grid computing usage with a primary focus on the MetaCentrum grid network. To introduce context to a reader, it is necessary to first establish definition of a grid versus a cluster. These definitions are not established terms, but for the needs of this work, the definition based on a hardware configuration of the systems is used.

The difference between a grid and cluster is then essential for grasping the discrepancy between the MetaCentrum grid and alternative services.

Another important part is a description of the planning system and its interface. The functionality of the interface effects user's workflow, as its options are limited. Understanding of the existing option is then needed for grasping the motivation behind the development of the own command line interface.

1.1 Computer cluster

A computer cluster is a single logical unit consisted of multiple units with a **homogeneous** hardware and operating system which are locally linked into one local network. Computers connected in a cluster behave as one single more powerful machine that provides parallelism and therefore much bigger computation power than a single computer. [8]

1.2 Grid computing

Grid computing is virtual (meta) computer composed of multiple nodes with a **heterogeneous** hardware configuration. Each node has its own resource management and memory allocation politics. Such nodes can be personal computers, clusters with powerful processors or graphics cards. Combination of nodes then provides a powerful computation source. [9]

Cluster	Grid
Same hardware	Different hardware
Same operating software	Various operating software
Centralized job management	Distributed job management
LAN	WAN
Centralized scheduling system	Distributed scheduling

Table 1.1: Summary of main differences between clusters and grids [3]

One way to classify the system as a grid is to meet three following conditions by Ian Foster:

- Computing resources are not administered centrally.
- Open standards are used.
- Nontrivial quality of service is achieved [10].

Under nontrivial quality of service is meant that a grid allows using resources in a coordinated fashion to deliver required service and allocation of multiple resource types meet complex user demands, so that utility of the combined system is significantly greater than that of the sum of its parts [10].

1.3 Cluster vs Grid

The difference between a grid and cluster is that clusters are **homogeneous** whereas grids are **heterogeneous**, everything else is somewhere in between and requires special care.

The computers that are part of a grid can run different operating systems and have different hardware whereas cluster computers all have the same hardware and OS [3]. Grid is inherently distributed by its nature over a LAN, metropolitan and WAN. On the other hand, the computers in the cluster are normally contained in a single location or complex [3] and thus connected via LAN. The difference between the two is summed in Table 1.1.

1.4 Distributed computing applications

As mentioned in the grid computing definition, a grid needs to provide a way for a user to effectively use resources. This functionality is provided by a job planning system, which is monitoring available resources and distributes computations across the grid.

1.4.1 Portable Batch System

PBS is an acronym for Portable Batch System, a scheduler developed initially by NASA in the early to mid-1990s, and still available as open source [11].

The original codebase was forked into at least three projects:

- Community version of PBSPro
- Torque – open source successor of PBSPro
- PBSPro – proprietary pay-for-core version

PBS optimizes job scheduling and workload management clusters, grid networks and supercomputers [12].

The node of a grid, that provides a user interface for submitting work to the grid is called *front-end* node.

PBS command line interface allows user to perform following core actions: [4, p. 18]

- Add a job to the queue.
- Remove a job from the queue.
- Rerun a job.
- Update queued job properties (alter, delete, hold, move).
- See the status of the queue where is among other submitted jobs.

The queue itself is a virtual computing space in the grid network where each job can require different memory, storage, CPU, GPU or application options.

1.5 Virtual organizations

Virtual organization in a grid computing context refers to a dynamic set of individuals or institutions sharing their resources according to defined rules and conditions. It's very common that participants have a common scientific topic or the same goal. [13]

The list of existing VO is large and they are both from academic and private sphere (see lists at 1, 2 or 3). This thesis primarily focuses on the organizations (VO and non-VO) based in the Czech Republic and alternative to grid or cluster computing. One of the academic VO is, for example, MetaCentrum National Grid Initiative, described in the following section.

1.6 MetaCentrum National Grid Initiative

MetaCentrum NGI is a national grid infrastructure operated by MetaCentrum, CESNET department, responsible for coordinating and managing grid activities in the Czech Republic [14].

The grid consists of resources across different locations and domains. The main focus of the project is to provide a virtual computer that allows effective utilization of connected hardware nodes. Such virtual computer can be then used for solving tasks whose memory and/or CPU requirements exceeds the possibility of an individual single computing machine [14].

The access to the grid is available for students and employees of academic or research subjects located in the Czech Republic [15]. Usage is free and without limitations for non-commercial use. The usage conditions are acceptance of rules and acknowledgement in user's publications.

1.6.1 National participation efforts

The MetaCentrum is flexible enough for any other academic subject to participate by integrating their resources as part of the grid network [14].

Participation in the project has several benefits for owners of clusters:

- qualified support during the selection of hardware and software, installation, operation of computing clusters, necessary administration, system updates etc.;
- account management, installation and operation of task management systems;
- shared operation monitoring;
- priority or even dedicated access to own machines [16].

Thanks to these advantages, ranging from CESNET organization to manage support of the machines and dedicated access to own resources, many notable Czech academic institutions joined the project:

- CERIT Scientific Cloud, [17]
- CEITEC Research Centrum, [18]
- Institute of Physics AS CR, [19]
- Information Technology at Masaryk University, [20]
- European Life-Science Infrastructure for Biological Information, [21]
- Technical University of Liberec, [22]

- University of West Bohemia [23].

The participation with the provided number of CPUs is shown in Figure 1.1.

1.6.2 MetaCentrum NGI details

MetaCentrum NGI is using the PBSPro planning system and submission is operated from front-end node with standard commands summed in table 1.2.

command	description
qstat	Status job, queue, server
qsub	Submit a job
qdel	Delete job
qrerun	Requeue running job

Table 1.2: Summary of basic PBSPro user commands [4, p. 50]

A queue is an abstract space under which the submitted job is executed. Each queue has associated hardware resources on which the actual execution is done and the system determines which resource to use based on availability and set priorities. Different queues can also provide specific options or values for parameters like CPUs cores, GPUs cores or memory size.

A single computational unit in the PBS environment is represented by a job. The job is usually bash script that prepares the environment and executes actual computation program and collects result.

One of the base parameters for submitting a job is a walltime that represents lower and upper bounds of the computational time. If the computation exceeds the upper bound of the queue's maximum walltime, it is terminated. The queue in the MetaCentrum environment is determined based on the walltime (see Table 1.3).

The MetaCentrum contains multiple types of queues, public and private ones. Public ones are available for all registered and approved users of the grid network. Private queues are dedicated to only some users and thus it is needed to have special permissions for submitting jobs there. Public queues are shown in Table 1.3.

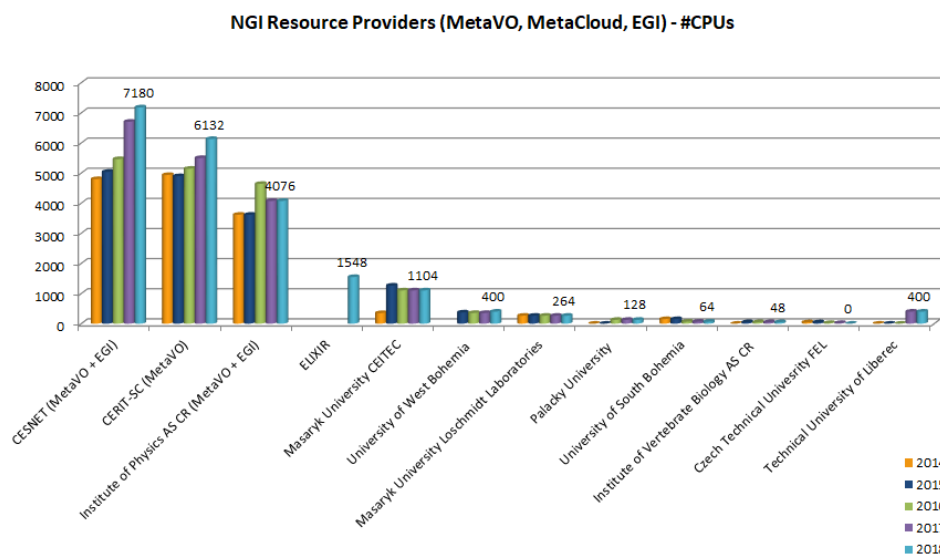
The grid offers a large variety of free and proprietary software along with commercial libraries for all registered users. The details of provided software is further described in subsection 2.5.2 and subsection 2.5.3.

1. GRID COMPUTING

queue	time limit (hh:mm:ss)	CPU/user	description
gpu	0 - 24:00:00	500	GPU queue
gpu_long	0 - 168:00:00	200	GPU queue for long jobs
q_2h	0 - 02:00:00	2000	2 hour CPU queue
q_4h	02:00:01 - 04:00:00	1800	4 hour CPU queue
q_1d	04:00:01 - 24:00:00	15000	1 day CPU queue
q_2d	24:00:01 - 48:00:00	1000	2 days CPU queue
q_4d	48:00:01 - 96:00:00	1000	4 days CPU queue
q_2w	168:00:01 - 336:00:00	1000	2 weeks CPU queue
phi	168:00:01 - 336:00:00	384	6 x Xeon Phi 7210

Table 1.3: Queues available on MetaCentrum for all users. CPU/user represents maximum CPU cores per user

Figure 1.1: MetaCentrum NGI providers statistics based on provided number of CPUs [1]



1.7 Academic cluster services

1.7.1 Information systems and Technologies of CTU Supercomputer

Information systems and Technologies department of Czech Technical University in Prague uses for difficult computations computers by SGI company [24].

The hardware consists of SGI ALTIX UV 100, one cluster of 7 nodes and

two SGI C1001 machines (see Table 1.4).

name	no. cpu x cores	memory	storage
SGI ALTIX UV 100	12 x 6	576GB	7.2TB
Cluster SGI XE node	2 x ?	24GB	?
SGI C1001-RP6	2 x 8	64GB	?

Table 1.4: Overview of resources provided by CTU cluster

The cluster uses PBS for task submission. Software like Mathematica, Matlab or Maple is not available on all machines and thus the user must know where to execute his computation.

The services are available to students and employees of Czech Technical University after approval [24].

1.7.2 Technical University of Ostrava – IT4Innovations

IT4Innovations National Supercomputing Center (IT4Innovations) is part of VSB-Technical University of Ostrava. In terms of theoretical peak performance, it is currently ranked as 87th most powerful supercomputer in the world [25].

The organization provides access to two supercomputers – Anselm and Salomon (see its queues in Table 1.5). For using them, specific front-end (access) nodes must be used. For task submission and resource allocation, PBS Pro system is available and is required to be used [5].

Access to both computers is available for both academic and industry subjects. For full access to all resources, academic applicants must be selected and approved, industry subjects have the option to buy a computation time on machines. Even the academic subjects have limited usage as they are given an only fixed number of core hours [5].

It is important to note that some of the resources are connected to the MetaCentrum grid network.

1.7.3 Summary of academic cluster services

It is not possible to directly compare MetaCentrum NGI with alternative services offered by other organizations, as MetaCentrum is grid network consisted of multiple clusters and computers, while other mentioned options are not grids but services offering clusters or super-computers not connected to one unified system.

The MetaCentrum grid offers many benefits in comparison to alternatives and those are:

- available for all academic subjects;

1. GRID COMPUTING

queue name	time limit	nodes per job	cores per node	description
qexp	1h	8	24	Express queue, testing and small jobs
qprod	48h	86	24	Production queue
qlong	144h	40	24	Long queue
qmpp	4h	1006	24	Massive parallel queue
qfat	48h	1	8	Dedicated queue, access to Nvidia accelerated nodes
qfree	12h	86	24	Allows utilization of free resources when project exhausted available ones
qviz	8h	2	4	Visualization queue, using OpenGL accelerated graphics
qmic	48h	8	-	Dedicated queue, access to Many Integrated Core nodes using Xeon Phi processors

Table 1.5: Queues available on Salomon super-computer [5]

- no restrictions on usage and used computed time;
- large quantity of resources and queues;
- both GPU and CPU queues;
- configuration of the environment by loading modules;
- licenses for proprietary software and libraries;
- dedicated access to queues with special permissions;
- unified environment on physical nodes.

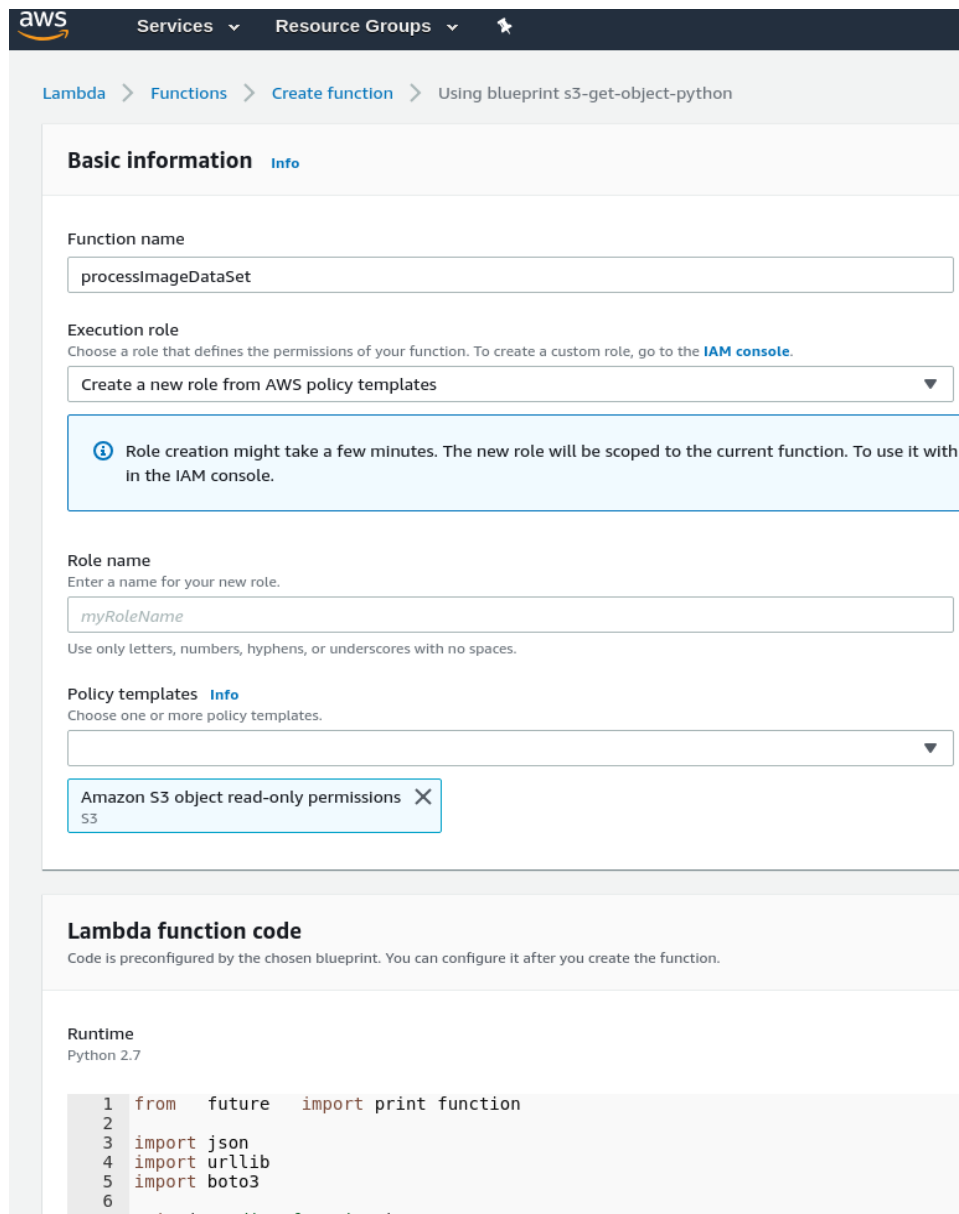
The easiest option for all academic organizations is to use the resources offered by MetaCentrum grid. Unless the academic subject has some specific needs or requires dedicated access to a specific cluster that is not part of the grid.

In the case of commercial subjects, some other option must be used. One of them is, for example, IT4Innovations, which offers an option to buy computation resources.

1.8 Cloud based services

Another approach is using one of suitable cloud-based cluster service. Cloud computing can be defined as a service that provides data storage and computing power without direct active management [8]. The term was popularized after announcing Amazon EC2 Service [26].

Figure 1.2: Creation of new Function for AWS Lambda services via web interface



The screenshot displays the AWS Lambda console interface for creating a new function. The breadcrumb navigation shows: Lambda > Functions > Create function > Using blueprint s3-get-object-python.

Basic information [Info](#)

Function name
processImageDataSet

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).
Create a new role from AWS policy templates

Role name
Enter a name for your new role.
myRoleName
Use only letters, numbers, hyphens, or underscores with no spaces.

Policy templates [Info](#)
Choose one or more policy templates.
Amazon S3 object read-only permissions ×
S3

Lambda function code
Code is preconfigured by the chosen blueprint. You can configure it after you create the function.

Runtime
Python 2.7

```
1 from future import print function
2
3 import json
4 import urllib
5 import boto3
6
7 print(f'Heading function')
```

1.8.1 Amazon Web Services Lambda

Amazon Web Services Lambda (AWS Lambda) offers running code without provisioning or managing servers. Payment is only for the time consumed while computing. [27]

Lambda offers virtual execution of code for any type of application of back-end service. The executed code is in this context called *Function*. Creation of a new function using web interface can be seen in Figure 1.2. [27]

The service has multiple resource limitations [28], that are much more strict compared to grid networks or mentioned clusters. Those are:

- maximum 1000 concurrent executions;
- maximum 75GB storage for function;
- memory allocation is limited 128 MB to 3008 MB, in 64 MB increments;
- time limit for function execution is 15 minutes;
- deployment package must be under 50 MB (zipped) and 250 MB (un-zipped).

Thanks to these limitations, especially the time limit, Lambda is not suitable for a long time running functions, which can be a significant issue for some computations.

1.8.2 Amazon Web Services Fargate

AWS Fargate is a compute engine for Amazon ECS that allows running containers without having to manage servers or clusters. With Fargate, all that needs to be done is to package an application in containers, specify the CPU and memory requirements, define networking and launch the application. The service provides seamless scaling of resources without the need to manage the infrastructure. [29]

For hypothesis testing, a user can package his computation to the container and execute it as a batch job or application. The difficulty here is that the environment in the container is solely in the hands of the user and can be hard to set up.

Another issue that could occur is with input and output files that are too large for the container to keep them. The solution is using a storage service, like Amazon S3 that offers large persistence storage accessible via HTTP protocol. This increases requirements on computation a script that needs to work with such service instead of local storage.

Maximum supported configuration for one instance is 4 virtual CPU cores and maximum 30GB memory in 1 GB increments.

1.8.3 Google Cloud Functions

Google Cloud Functions (GCF) is an alternative solution similar to AWS Lambda. It is an event-driven serverless computing platform that offers automatic scaling and does not require to manage resources. [30]

GCF offers the option to deploy code and execute a function on the Google cloud platform. This has some limitations similar to the AWS Lambda [31], which are:

- maximum number of functions per project is 1000;
- deployment size must be under 100 MB compressed and 500 MB uncompressed;
- time limit is 9 minutes;
- maximum memory a function can use is 2048 MB;
- maximum 100,000 GHz-seconds per 100 seconds.

Run-time limits of the solution are much stricter than the AWS Lambda ones. Some of the limits are defined in a different manner or missing entirely. This makes a comparison without deep analysis not possible. One of missing limit is, for example, maximum storage that can be used or available CPU computation power.

1.8.4 Azure Functions

Azure Functions (AF) is another alternative solution for easily running small pieces of code, or "functions," in the cloud. The solution allows writing the code without worrying about a whole application or the infrastructure. The code is executed on the scalable Microsoft Azure platform. [32]

The AF offers two plans, *Consumption* and *App Service*. The key difference between them is that the Consumption is based on scalable server-less architecture while the App Service provides execution of functions on own VM instances. The App Service also provides fewer limitations and the possibility to use more resources than in the case of *Consumption* plan. Complete comparison can be found in Table 1.6.

1.8.5 Summary of cloud-based services

The main motivation behind these services is to out-source server management to the external provider, while the clusters and grids are built with a focus on computing tasks. Although cloud services can be used as computing entities, they don't offer needed flexibility and have limitations that make them unsuitable for some computations.

Consumption plan	App Service plan
scalable serverless plan	deployed Virtual Machine instances
automatic scaling	scaling by adding more VM instances
pay for execution time	pay for used VM instances
max time limit 10 minutes	unlimited time limit
CPU core per function	4 CPU cores per VM instance
1.5 GB memory per function	14 GiB of memory per VM instance
maximum 200 instances	-
max 1 new instance per 1s (HTTP trigger)	-
max 1 new instance per 30s (non-HTTP trigger)	-

Table 1.6: Comparison of Microsoft azure plans [6] [7]

The execution of thousands of mostly long-running computations that use various libraries and licensed software is unsuitable for most of the services. Specialized software and licences are not supported out of the box and must be set up manually with possible restrictions. Strict time limits and unavailability of GPU cores is then another issue.

Needless to say, AWS Lambda, Google Cloud Functions and Azure with the Consumer plan are almost the same services that are running on different platforms. All of them then have very similar limits and usage scope.

One of the exceptions is Azure functions with the App Service plan (see subsection 1.8.4) and almost identical alternative, AWS Fargate. Although these services still have some of the crucial limitations, they are the only one that allow long-running executions making them viable for computations that do not need a massive amount of CPU cores and memory.

MetaCentrum NGI example usage

MetaCentrum NGI resources can be used in many ways. This chapter aims to describe one type of example usage that this thesis focused on. As a by-product of the description, all used and needed terms used in the context are described along the way.

2.1 Job lifecycle

The script is submitted to the grid network via *qsub* command. Based on the walltime parameter, a job is put into the proper queue.

The submitted job is waiting in the queue for required resources and is executed immediately when resources are available (and the job is first in the queue). Physical grid node executing the job is not guaranteed.

If a user wants to check job status, *qstat* command must be used (see Listing 2.1).

2.2 Computational file

The MetaCentrum can be effectively used for mapping high-dimensional space of possible parameterization of an algorithm, where each parameterization is represented by a job that performs the computation execution based on the parameters.

The mapping space can be defined as a **task**. A task is consisted of jobs, where each of them uses one parametrization of the space and is submitted to the queue.

The actual execution is represented by the computational file that accepts parameters as a function or uses environment variables to configure parameters of the computation.

Listing 2.1: Output of qstat command on the MetaCentrum grid

```

1 $(STRETCH)jtucek@tarkil: qstat | head -10
2
3 Job id           Name           User           Time Use S Queue
4 -----
5 3559877.arien-pro wiki_a4_splitte user1           00:00:00 H q_1d
6 3559881.arien-pro wiki_a4_splitte user1           00:00:00 H q_1d
7 3560548.arien-pro 3D-MPI-test    user2           00:00:00 H q_1d
8 3563328.arien-pro conll_train_tes user3           00:00:00 H q_1d
9 3563402.arien-pro meta_B100HOHMRC user4           00:00:00 H q_1d
10 5163329.arien-pro meta_UmbH-allre user5           0 Q q_1w
11 6140219.arien-pro DKS_toluene_com user6           0 H q_6d_ceitecmu
12 6140225.arien-pro DKH2_toluene_co user6           0 H q_6d_ceitecmu

```

The computation can be represented by **Matlab** or **Python** file and is the unit that performs one part of the computation or hypothesis testing based on a given configuration.

2.3 Example of usage

For the configuration itself, an example of the existing scenario is to create a **controlling** script that setups environment and executes the computation (see Listing 2.3). This script is then submitted to the queue. After submitting, the queuing system handles execution automatically. Chooses a physical node and executes the controlling script.

The issue here is that executing controlling script prepares computation just for one set of parameters. This means that the controlling script itself must be generated by other script or manually edited for each set and submitted to a queue.

The second script can be called *parametrization* script. The parametrization script contains nested loops that generate requested combinations of parameters for the computation (see Listing 2.2) and then submit a created script to a queue. See all file types summed in Table 2.1.

2.4 Versioning

In the given example an user typically uses different variations of *controlling* and *parametrization* scripts for generating scripts. These files are being modified for different computations or re-execution of the same task.

Such behaviour can possibly end up in the state, where the user is unable to identify which configuration was used for the execution and cannot easily

file type	description
controlling script	prepares environment, configuration and executes computation file
parametrization script	generates controlling script and submits it to a queue
computation file	performs computation based on set configuration

Table 2.1: Overview of file types and their purpose

track changes in case files on the filesystem were edited directly by the user.

Input and output data is another part which should be versioned in an experiment. In case of these files, the issue is that input and output data sets can be very large in size (gigabytes and more) and thus easily fill up available local storage.

A relevant requirement for a user (in the example given), would be the versioning of all the data needed for each run and output data. The motivation is the ability to track history executions and to easily re-execute the computation with identical configuration.

2.5 Resource allocation

2.5.1 Hardware resources

For successfully submitting a job to a queue, a user must have knowledge of how the *MetaCentrum* grid works and how to use available resources.

Without such knowledge, and of course, the grasp how demanding is the computed job itself, a proper configuration of a job is unlikely.

A user has to determine which hardware resources a job will use:

- if the task uses CPU or GPU for computation,
- how many cores will be used,
- task memory consumption,
- and file storage size.

This narrows available queues that can be used for computation.

The other expected parameters are runtime duration of a job, walltime, based on which the system attaches a job to a queue. All available public queues are summed in Table 1.3.

Listing 2.2: Shortened sample of parametrization script generating controlling script and submitting it to the queue

```
1 #/bin/bash
2 RUN_COUNTER=1
3
4 for (( VICINITY=2; VICINITY<=2; VICINITY++ )); do
5     for (( MIN_TRANSL=0; MIN_TRANSL<=19; MIN_TRANSL++ )); do
6         let "RUN_COUNTER++"
7
8         controlling_SCRIPT=~ /bash_scripts/tmp/tmp_${RUN_COUNTER}.sh"
9
10        # put env. variables into script that will be submitted
11        echo "export VICINITY=\${VICINITY}" >> "$controlling_SCRIPT"
12        echo "export MIN_TRANSL=\${MIN_TRANSL}" >>
13            "$controlling_SCRIPT"
14
15        cat "$HOME_DIR/bash_scripts/controlling_script_template.sh" >>
16            "$controlling_SCRIPT"
17
18        qsub -l select=1:ncpus=12:mem=30gB:scratch_local=10gB \
19            -l walltime=2:00:00 \
20            -l Matlab=1 \
21            -l Matlab_Neural_Network_Toolbox=1 \
22            "$controlling_SCRIPT"
23    fi
24 done
25 done
```

The main user requirement is for a job to finish as fast as possible with success status. Selection of proper parameters is a non-trivial task for user-beginner because of its complexity. Proper selection depends on the specific behaviour of the job, such as effective scaling based on a number of CPU/GPU cores or current state of queues in the grid network (number of submitted jobs to queues).

Even if a user selects best possible walltime for his job, it can mean that the assigned queue is currently very overcrowded from jobs of other users, which could result in job waiting for a long time in the queue before actual execution. In this case, it could be much more efficient to select a queue that is freer and thus waiting for the execution would be much shorter.

2.5.2 Modules

On the MetaCentrum grid, the tools necessary for the execution of the computational script are not available in a user environment by default [33]. The

Listing 2.3: Controlling script executing Matlab function

```

1 export OUT_DIR='/user/dir/out/1'
2 export Matlab_THREADS=5
3 export Matlab_DIR='/user/dir/storage'
4 export MIN_TRANSL=1
5 export USE_GPU='yes'
6 export LAYERS=[10, 20, 30]
7
8 module add Matlab # load Matlab module
9 module add jdk-8
10 # execute Matlab command
11 Matlab -nodesktop -nosplash -nodisplay -r      \
12 "try,                                           "\
13 "    main_batch01($MIN_TRANSL, 'useGPU', '$USE_GPU', "\
14 "                'layers', $LAYERS),          "\
15 "catch e,                                       "\
16 "    disp(getReport(e)),                       "\
17 "    exit(1),                                  "\
18 "end,                                           "\
19 "exit(0)"                                       \
20     &> "$OUT_DIR/Matlab_stdout.log"
21
22 if [ "$?" -eq 0 ]; then
23     echo "run number '$RUN_ID' successfully finished" >
24         "$OUT_DIR/success" # create file 'success'
25 else
26     echo -e "\nMatlab script completed with exit status != 0\n"
27 fi

```

grid uses a modular subsystem that provides a command line interface for modification of user environment by loading required modules and thus option to easily set up his environment without the need to perform manual configurations.

After module is loaded as is shown in Listing 2.4, chosen application is integrated into the user environment, including availability in the PATH variable.

Listing 2.4: Modular subsystem terminal commands

```

1 // lists all available modules on the grid
2 $ module available
3 // loads modules into a user environment
4 $ module add <modulename> # e.g., module add Matlab

```

User must load a module each time he logs in, as the state is not persisted from the last session.

resource type	description	example
walltime	Time based on which queue is selected	2:00:00
chunk	Set of resources allocated on 1 node	1
ncpus	Number of CPU cores on each chunk	2
mem	Size of memory	4gb
ngpus	Number of GPU cards	2
scratch_local	Size of file storage on a node	1gb
scratch_shared	Size of shared file storage	1gb

Table 2.2: Overview of resource types with example value

In case of a job execution on a physical node, the environment is in the default state and thus before actual execution, proper modules must be loaded. This can be seen in Listing 2.3, where JDK and Matlab are loaded as modules.

2.5.3 Licenses and toolboxes

Some software resources need to be specified directly in the queuing system to ensure that the environment where a job is executed will have proper configuration and licences. On the MetaCentrum grid they are specified via *toolbox* option.

As every PBSPro option, it can be specified in two ways, directly when submitting a job (see Listing 2.5) or via metadata headers in the submitting file (see Listing 2.6). The headers are located at the beginning of the file and are written as *comments* in a custom format that is recognized by the PBS. [4, p. 34]

Listing 2.5: Submitting and specifying Matlab toolbox directly

```
1 $ qsub ... -l Matlab=1 -l Matlab_Statistics_Toolbox=1
```

Availability of licenses is another complexity factor waiting for a user-beginner. In case of an unavailable license (whole capacity is used by other jobs), the job will wait in the queue until the license will be available. This can be a big issue in case of hundreds or thousands of jobs requiring the same license. In such case, only a limited number of jobs can run at the same time as license works as a required token for execution.

The need for the license allocation is that capacity for the proprietary software execution is limited and only a fixed amount can be used on the grid at the same time. For example, according to the official grid documentation [34], the total number of Matlab licences is 450 with toolboxes having much lower count. The current state of available licenses can be found in the official MetaCentrum documentation located in the License section.

Listing 2.6: Submitting Matlab job with toolbox specified in a file's metadata using PBS directives

```
1 #!/bin/bash
2 #PBS -N PythonTaskType
3 #PBS -l walltime=01:00:00
4 #PBS -l select=1:ncpus=1:mem=1gb:scratch_local=1gb
5 #PBS -l Matlab=1
6 #PBS -l Matlab_Statistics_Toolbox=1
```

2.6 Notifications

The PBSPro system provides email notifications for notable changes in a job lifecycle. Main lifecycle points are: job cannot be routed to a node, is deleted by a user, is aborted or begins and ends execution. [4, p.50]

A user can modify default lifecycle points [4, p.50] as part of configuration during submission. Following methods can be used:

The `-m <mail points>` option to `qsub`

The `#PBS -WMail_Points=<mail points>` PBS directive

To configure email recipient list available methods are [4, p.50]:

The `-M <mail recipients>` option to `qsub`

The `#PBS -WMail_Users=<mail recipients>` PBS directive

The case where this functionality stops being sufficient is when a user submits hundreds or thousands of jobs. In this case, the user will consecutively receive thousands of emails for each job and the only result will be his overfilled email inbox. On the other hand, with proper filtering and searching it is possible to detect failed jobs and possible issues in submitted task much faster than by manually checking state via remote terminal connection to the grid network.

2.7 Failed jobs and automatic resubmitting

The nature of some computations does not have to have a deterministic result if for example, used algorithm works based on randomization and thus execution is not ensured to be successful on every run.

In this case, a user wants to submit a job with a given configuration to the queue multiple times until computation ends properly. Unfortunately, PBSPro or the grid does not offer an option to automatize this task.

Needless to say, PBSPro has rerun functionality, but it works only when a job is terminated before finishing [4, p. 189]. So even if a job ends with

2. METACENTRUM NGI EXAMPLE USAGE

failure status code, the queue system considers a job as finished and will not rerun such job. In cases where second execution would cause a problem (like rewriting needed historical data), a user can mark a job as not rerunnable.

The only way to rerun a finished job is by manually resubmitting it, which can be quite a time-consuming process in case of multiple jobs.

Possible solution and analysis of CLI

The section 2.2 defined the task as a unit consisting of jobs, where each of them uses one parametrization of the configuration space and is submitted to the queuing system. Development of this abstract concept further and most importantly, integrating it into the CLI as the core approach, allows new innovative approach for solving issues encountered on the MetaCentrum grid. The detailed solution to these issues and others are described in this chapter.

3.1 Versioning of scripts and data

One of the first steps for the successful versioning is the ability to track historical execution in such way that it can be resubmitted to the planning system. The files needed for re-submission are input data, controlling scripts and computational files (see overview in Table 2.1). The controlling scripts must be either directly versioned or there must be an option to generate them using previous configuration of parametrization space.

More flexible option is to only remember user configuration, as the controlling scripts can be generated in the same manner as in the previous execution. This configuration, along with state of the run is called metadata. The simplest option is to store the files as copies in a structured directory hierarchy.

The size of data files is often overwhelming and thus can cause issues by exceeding available capacity. The solution is to exclude large files from versioning and for the metadata to only contain absolute paths to them. Exclusion can be then based on a user configuration option. Excluding of data files brings a new issue in the re-submission process, that must use the same input data files and, if not configured properly, can overwrite output data.

It is also possible to use some existing Version Control System and for a user to version the files manually using VCS CLI. The requirement for this

solution is for data to have folder structure allowing initializing VCS repository in it.

Extension of this idea is to directly integrate the VCS into CLI and automatize the process. The advantage is that automatic versioning can be done in each CLI execution and not to rely on a user.

3.1.1 Versioning tools

Version Control System provides management of changes to documents, source code and other files. VCS is usually a stand-alone application that handles tasks to manage multiple versions of files. In the case of distributed VCS, the application also provides an interface for synchronization of the state with a remote repository. Wide-spread VCS are for example Git, Mercurial or SVN. [35]

Size of data files cause issues for the remote repositories with the limitation for the maximum file and overall size. The Github, Git-based hosting service has the limitation on one file 100 MB and recommends to have repositories under 1 GB each [36]. Another hosting service, Bitbucket, has a soft limit 1 GB and 2 GB hard limit for git repositories [37]. Although alternative services exist and a user can use own hosting server, the main issue here is that Version Control Systems was designed primarily for source code version control [38].

The solution for versioning large data files is to use a different approach by versioning only pointers to the actual remote location of the large files. Distributing data across different physical locations and not in the actual repository allows then much bigger flexibility in storage options [39] [40].

DVC.org

One of the solutions for versioning large files is DVC [39] project that specializes on version control of machine learning projects. Project is designed for working upon large data sets and solves the issue of tracking files that are too large for the VCS (like Git) to handle, as the remote storage location for each file can be defined separately so the large files can be stored in other location. DVC offers its own command line interface to configure versioning of large files. [39]

The repository only stores internal DVC pointers to the external location of the large files (see Figure 3.1).

Unless a user works with large files, other basic operations and versioning of other files are done using a VCS command line utility. This makes DVC independent on the underlying version control system.

Git Large File Storage

More Git native-based approach is the usage of Git Large File Storage (LFS). LFS works on the basis of replacing large binary files in the repository with

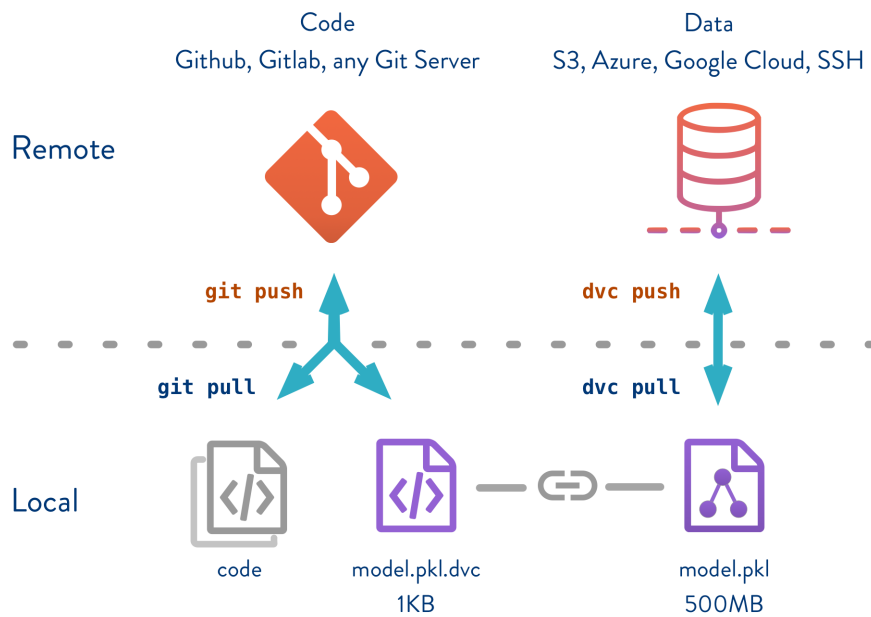


Figure 3.1: Versioning of large files using DVC.org, supported by Git server and external data storage [2]

text pointers. Actual files are then hosted on a dedicated server where LFS is supported [40].

Git and DVC comparison

The core difference between approaches is that DVC can store large files on remote servers that do not have installed dedicated software like LFS and thus can be used for any remote storage. Remote storage for the DVC can be for example Amazon S3, which is cloud-based file storage service [41]. The second difference is that DVC does not depend on specific under-lying VCS, so a user can choose the one he needs or prefers.

3.1.2 Versioning summary

Although VCS and both solutions for versioning large data can be technically incorporated into the CLI, it was not implemented. The main issue here is that the potential policy on how to version the data is not clear. Taking a stand to one way of versioning the data, could mean limiting the usage of CLI for other users or forcing them to use solution that they do not require. To solve this, further analysis of user behaviour is required.

The solution that offers flexibility and still solving main issues is the implementation of the systematic view on the execution data and version them locally. This can be developed further by integration of VCS and large data storage tools.

3.2 Allocation of resources

3.2.1 Virtual hardware resources

Straightforward solution for specifying resources would be to rely on a user configuration and follow it blindly. The biggest disadvantage in this solution is that a user must have good knowledge about MetaCentrum grid.

User knowledge is necessary for a determination whether CPU or GPU suits well for a job computation as well as for the task wall-time, memory consumption and disk space.

And even in this case, the best configuration is not ideal in every moment of the grid, as the number of submitted jobs of other users in queue varies in time. This could be the cause of jobs waiting in the overflowed queue instead of being executed in the queue that is not used as much.

The opposite approach is to compute resources dynamically and automatically by executing a script with various configurations and determining the best one in a given time.

The complexity of this solution depends on a number of parameters that are being determined. The dimension of the parameters could be then reduced by a user providing part of the configuration. Such parameter that could be very helpful in narrowing the queues used for computation is a queue type (CPU vs GPU allocation), because not knowing it means that jobs must be submitted to both types of queues and prolong computation itself even more.

The next issue in this approach is choosing the right size of memory and storage size, where its impact on the computation in most cases will be if it fails or not. The automatic solution is then choosing some base values and try them until successful ones are found.

3.2.2 Software resources

The software resources depend on the type of computational task and specific libraries that are used. Thanks to this, a tool without deep inspection of the computational script, cannot determine every needed *module* or *toolbox* that are needed for execution in the queue system.

Even so, with minimal knowledge about the task, *some of the modules* can be determined automatically. For example, in case we knew, that the task is a Matlab one, we can predict that a user wants to use the newest available Matlab.

Although if a user wants to use specific libraries, it cannot be solved automatically and a user will need to input them.

3.3 Notifications

Notification support on the MetaCentrum grid has its disadvantages for a large number of submitted jobs, as from PBS's point of view they are stand-alone jobs without any connection. The task abstraction allows a new approach to the notification and status checking, as it is possible to track submitted jobs as a whole.

A new mechanism for checking the status of submitted jobs belonging the same computation must be introduced. The solution can then notify a user about the most crucial events in a task lifecycle.

These events are if a job is failing or when all jobs of a task end and computation can be then considered as finished. The viable notification method is by email, supported on the grid by using terminal command **sendmail** (see Listing 3.1).

Listing 3.1: Example of sendmail usage

```
1 $ sendmail -tv < "./file"
2
3 // file content
4 To: to@email.com
5 Subject: Subject of the mail
6 From: from@email.com
7
8 Mail text
```

3.4 Failed jobs and automatic resubmitting

Implementing re-submission of failed jobs could be considered as an extension of the notification system described in the previous section. To avoid an infinite re-submission loop, a user needs the option to configure maximum resubmit count.

Such parameter also makes sense, as with subsequently failing jobs, the probability that next submission would end successfully is lower with each execution.

The proposed functionality is that in automatic check mechanism used for notifications, the failed jobs are submitted to the queue.

Keeping historical data is another aspect of re-submission, where the straightforward re-execution could overwrite existing outputs of the failed job.

3. POSSIBLE SOLUTION AND ANALYSIS OF CLI

The solution to this issue is changing the submitted script in a way that new output data does not override the existing one. That can be done by changing output locations and the directory in which the script is submitted so no data are overwritten.

CLI structure and usage

Based on possible solutions mentioned in chapter 3, CLI solving the most critical issues was created. The solution reduces the complexity of MetaCentrum and PBS submission system to one configuration file and *clusterize* application. Moreover, the tool offers a solution for versioning, user notification and task monitoring.

This chapter is dedicated to the introduction of the created tool from a user perspective. Installation, available commands and configuration file are described.

At last, documentation and quick guide accessible outside of the thesis is introduced. The external documentation allows users to learn about the CLI functionalities without the need to read the whole thesis. The quick guide is then an effective way for the introduction of the tool by providing examples that can be executed with minimal additional configuration.

4.1 Installation

Installation for a user was designed to be as straight forward as possible. The CLI was created with the focus on the MetaCentrum grid environment. Although the tool can be also installed on other systems using PBS and modular subsystem, they are directly supported.

Installation on the front-end node is done via running one command in a shell terminal. The command downloads installation script and runs it with a parameter that represents a version to be installed.

The command for installation is:

```
1 $ sh -c "$(curl -fsSL URL_TO_SCRIPT/install.sh) 0.13"
```

Installation script deals with preparing folder structure, downloading distribution file and generating wrapping run script. The distribution tar file is

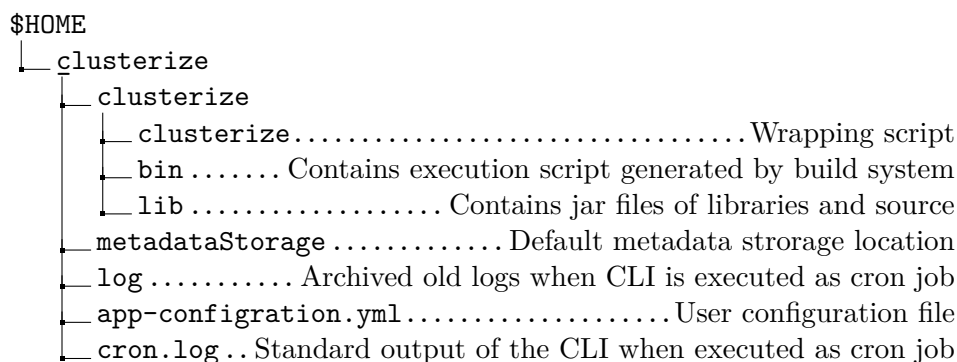


Figure 4.1: Description of \$HOME/.clusterize folder structure

downloaded from the GitHub releases section ¹.

Execution files are saved in a user's home directory `.clusterize` folder (see Figure 4.1). An important part of the script is checking for `JAVA_HOME` environment variable during installation, so it can be set automatically in wrapping script before actual CLI execution.

Removing clusterize from system can be done by recursively deleting `.clusterize/clusterize` folder. For complete deletion of CLI from a system, a user must delete the configuration and metadata about past runs. This can be done by deleting remaining files in the `.clusterize` folder.

4.2 Commands in the CLI

This section aims to describe the usage of all commands that the CLI offers, along with a detailed explanation of the configuration file used for task specification.

The availability of the CLI binary called **clusterize** in the `PATH` variable is assumed in all following examples. The general usage can be defined like this:

```
$ clusterize <command> [specific parameters or switches]
```

Submit

Submit is a command, that submits a task's jobs to the queue based on a given configuration file. Folders, scripts and other files are prepared before actual submission. If some required information is missing in the configuration file, a user can interactively input such data. If the path is not given, CLI tries to use **clusterize-configuration.yml** file in the current directory. A user

¹The new release is create based on the newly pushed tag to the git repository. Automation is supported by Travis Continuous Integration platform [42]

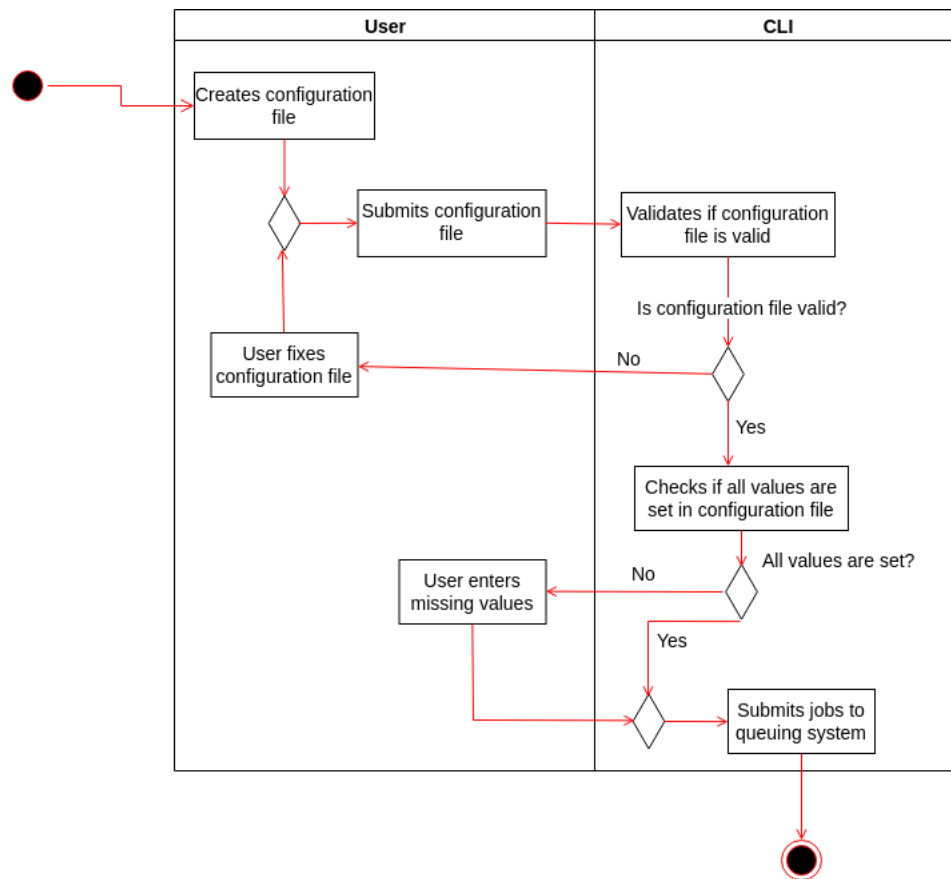


Figure 4.2: Activity diagram of submit command

and CLI interaction for submit command is visualized in Figure 4.2. Usage is shown in Listing 4.1.

Listing 4.1: Submit command usage example

```

1  $ clusterize submit [optional path to configuration file]
2  $ clusterize submit
3  $ clusterize submit /path/to/configuration-file.yml

```

List

The list will check metadata storage folder (containing information about past runs) and return the state of executed tasks with their id. This command is very useful for checking the status of the task that contains thousands of jobs, as we can see the exact state as a whole in a structured format (see Figure 4.3).

The CLI assumes that metadata folder is in the home directory, but it is possible to specify a custom location. A path can be either specified directly

4. CLI STRUCTURE AND USAGE

```
[2019-03-16 10:27:48] [INFO] Main:13 - Starting app. Profile: DEV Arguments: list
[2019-03-16 10:27:50] [INFO] ShellServiceDockerProxy:23 - ShellServiceDockerProxy run command wrapper
=====
Found tasks:
* 1 - PythonTaskType - 26/02/2019 08:00 - 41/41 FAILED
* 2 - PythonTaskType - 26/02/2019 09:37 - 41/41 FAILED
* 3 - PythonTaskType - 26/02/2019 09:44 - 41/41 FAILED
* 4 - PythonTaskType - 26/02/2019 09:55 - 41/41 FAILED
* 5 - PythonTaskType - 26/02/2019 09:56 - 41/41 FAILED
* 6 - PythonTaskType - 26/02/2019 10:05 - 41/41 FAILED
* 7 - PythonTaskType - 26/02/2019 10:07 - DONE
=====
[2019-03-16 10:27:50] [INFO] Main:20 - App finished
```

Figure 4.3: Example of list command with debug output

or using the configuration file containing such path. Structure of configuration file is described in Configuration file section. See example in Listing 4.2.

Listing 4.2: List command usage example

```
1 $ clusterize list
2 $ clusterize list -p path/metadataFolder
3 $ clusterize list -c /path/to/configuration-file.yml
```

Resubmit

Resubmit command reruns task based on its id using the same data and configuration. One of the usages is when task execution is not deterministic and unexpectedly fails, in which case, a user can rerun the whole task and retrieve the second result.

Important note is that **resubmit** depends on **list** command to be executed before. This shouldn't be an issue for a user, as the only way to find out task id is from list command. See resubmit example in Listing 4.2.

Listing 4.3: Resubmit command usage example

```
1 $ clusterize resubmit [task id]
2 $ clusterize resubmit 12
```

Cron

CLI needs some mechanism to periodically check for updates and notify the user without needing him to have an active session on the front-end node. The native system solution is to run the CLI command action as a cron job in a defined interval.

Running *cron start* will register new *cron job* that executes **cron-start-interval** command. Before registering (starting) *cron job* for the first time, a user is asked for his email, which is then saved to the configuration file in *\$HOME/.clusterize* directory.

```

[2019-03-16 10:26:33] [INFO] Main:13 - Starting app. Profile: DEV Arguments: help
===== HELP =====
Usage: clusterize <command> [...additional variables]

  submit [optional path to configuration file] - submits new task to cluster according to configuration structure
                                                - default path is used if not specified (clusterize-configuration.yml)
  resubmit [task id] - resubmits failed task
  list [OPTIONS] - lists tasks
    -p [VALUE] - define path to metadata folder or default is used
    -c [VALUE] - specify path to configuration file
  cron [start|stop|restart] - run cron to watch executed tasks and receive notifications
  help - displays help
[2019-03-16 10:26:34] [INFO] Main:20 - App finished

```

Figure 4.4: Help command output

CLI also provides additional manipulation operations like stopping or restarting periodic check (see Listing 4.4).

Listing 4.4: Cron command usage example

```

1  $ clusterize cron start
2  $ clusterize cron end
3  $ clusterize cron restart

```

Cron start internal

Cron-start-interval is an internal command that is executed in periodic cron job execution. Command basically extends **list** functionality and also resubmits jobs if it is enabled in the configuration file. When completion of the job is detected, the user is notified about the current state in an email notification.

Help

Displays help and usage examples to standard output, see usage example and output in Figure 4.4.

4.2.1 Configuration file

The configuration file is a file that specifies the task and its jobs that are supposed to be submitted to queues. Default and only supported serialization format is YAML (YAML Ain't Markup Language), which is a standardized human-readable data serialization language [43].

The file can be divided into 4 sections:

- Iterations – array that specifies loops
- General – general configuration
- TaskType – specific configuration for task
- Resources – specification of queue resources

Iterations

Iterations section emulates nesting loops in a parametrization script and is used for generating all required parameters that a user wants to use for execution of the computation.

Possible iteration types are:

- ARRAY – iteration over values given in an array;
- INT_RANGE – provides standard for-cycle iteration like programming language C.

Configuration example of the iterations part is shown in Listing 4.5, its equivalent programmatic representation using two inner loops is in Listing 4.5.

Listing 4.5: Iterations of two loops, one as an array and one from value 1 to 2.

```
1 # defines iterations in a given order
2 iterations:
3   - type: ARRAY # will iterate over values given in an array
4     name: VICINITY_TYPE # variable name
5     values: [10, 20, 30]
6   - type: INT_RANGE # will iterate over given integer range
7     name: MIN_TRANSL
8     # initial value (inclusive)
9     from: 1
10    # last value (inclusive)
11    to: 2
12    # step for each iteration
13    step: 1
14    # operation type (PLUS, MINUS, MULTIPLY, DIVIDE)
15    stepOperation: PLUS
```

Listing 4.6: Visualization how the iterations listing from 4.5 would look like in Javascript

```

1  const options = []
2  for (i in [10, 20, 30]) {
3    for (j = 1; j <= 2; j++) {
4      options.add({VICINITY_TYPE: i, MIN_TRANSL: j})
5    }
6  }

```

General

The general section is mainly for specifying the location of folders where data are kept or retrieved from. Metadata storage path contains information about task execution. Its content is created after submitting and updated after each status (list) check. Storage path contains execution files, which are generated script, outputs and job status information.

The source path is the location of data or other resources needed for computation. Resubmit count defines how many times each job can be resubmitted in case of failure.

The **dependentVariables** section specifies variables, whose values depend on a value of a derived variable, i.e. by changing it with some static modifier. The derived variable can be from the environment or the iteration section. Modifier must be bash expression for it to work.

Listing 4.7: General configuration part example

```

1  general:
2    # optional
3    metadataStoragePath: 'folderName/metadataStorage'
4    storagePath: 'storage' # usually storage
5    sourcesPath: 'sources' # sources, data used for computation
6    maxResubmits: 3
7    # environent variables
8    variables:
9      ENV_VAR: 123
10   # variable that depends on other variable value
11   dependentVariables:
12     - name: MAX_TRANSL
13       dependentVarName: VICINITY_TYPE
14       modifier: '+1'

```

Task type

Task type specifies how user computation is executed. Currently supported task types are **Matlab** and **Python**. From a user perspective, the configuration slightly differs (see Listing 4.8 and Listing 4.9).

The CLI based on action type determines needed modules and sets their latest versions to the environment.

The main part of task type is a function (or command) call, which specifies execution of a computational script and parameters, environment variables or static values that should be passed on.

Listing 4.8: Matlab task type configuration example

```
1 taskType:
2   type: Matlab # type of runner
3   # Matlab function call with variables and specified values
4   functionCall: |—
5                 main_batch01($VICINITY_TYPE, $MIN_TRANSL, 'useGPU')
```

Listing 4.9: Python task type configuration example

```
1 taskType:
2   type: PYTHON # type of runner
3   # bash-like function call with variables and specified values
4   command: |—
5           python -c "from python_fibonacci import main; main($FROM, $TO)"
```

4.3 Storages

CLI uses two types of storages: **storage** and **metadata**. Each one has a different purpose, which is described in the following subsections.

Each execution creates two folders of the mentioned types, with name in this format:

```
task-(TASK_ID)-YYYY-mm-dd_hh:mm:ss
```

4.3.1 Metadata storage type

Metadata storage stores *metadata* file for each task in the created directory. The file keeps status of a job, the configuration used for execution and other additional information.

For a task to be recognized in the metadata directory, task's directory must be present in the mapping file. The mapping file lies directly in the metadata folder and maps task id with the corresponding directory.

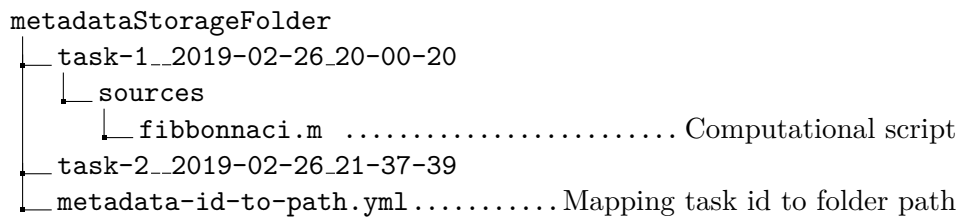


Figure 4.5: Metadata folder structure

4.3.2 Storage type

Storage type contains the job’s generated script and output files. The generated script is the one that is submitted to the queuing system and is similar to the **controlling script** discussed in section 2.3.

The script setups environment before execution and executes the computation script with proper parameter values. The script is also responsible for the creation of files that are used for tracking the status of a job because the information that can be retrieved from the queuing system is limited. These information ranges from job’s exit code to when the computation of a job actually started and ended. The remaining files are standard and error output of the script accompanied by the computation script standard output saved in the file called *stdout.job*.

For each job in a task, a folder with a name equal to a job’s id is created. If a job fails, rerun option is enabled and maximum rerun quota is not exceeded, the job will be executed in a different folder. This folder will be created in the same level as the original job folder, with name in the format:

```
{JOB_ID}_RERUN_{RERUN_COUNT}
```

The size requirement for each job folder is not very demanding. The generated *controlling script* has the approximate size of 4 KB and supporting files have size lower. As for standard and error output, the size depends on a computation script output.

4.4 User quick start

CLI requires basic user knowledge of the MetaCentrum and the CLI. For this purpose, there is documentation of MetaCentrum (wiki) and several files describing CLI: README, QUICK_GUIDE and CONFIGURATION. Files are located at the root of GitHub repository. The backup of repository is also on the attached CD under **sources** folder (see Appendix B).

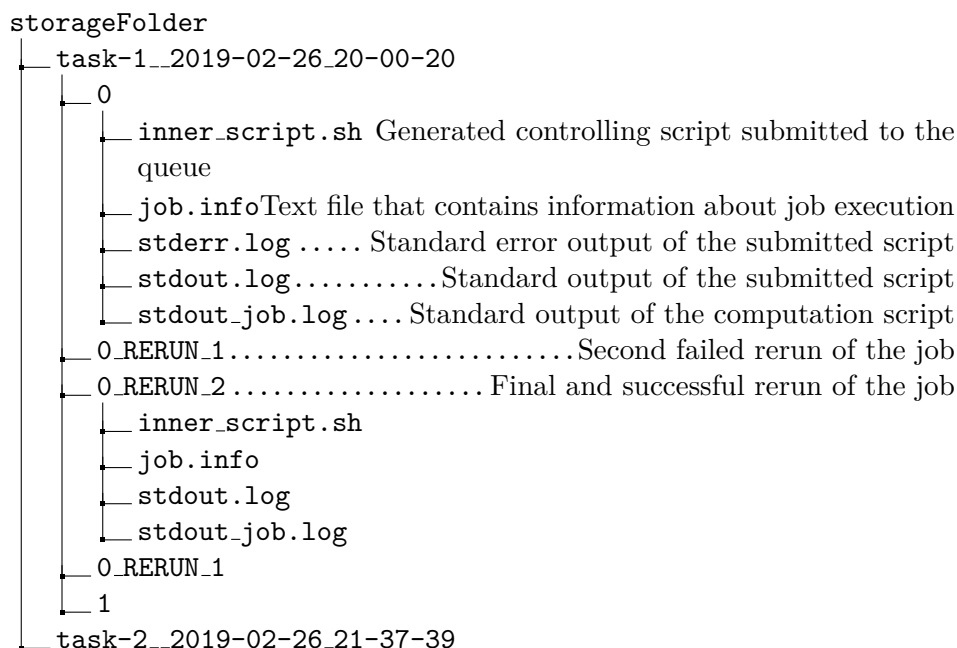


Figure 4.6: Storage folder structure

4.4.1 Quick guide

For very fast warm-up of the user of CLI, several examples were prepared along with steps on how to submit the first task. Documentation and sources of the quick guide are part of the Git repository. Source files for examples are located at **examples** folder, documentation is in **docs/QUICK_GUIDE.md** file.

Quickstart contains two main examples, first uses a simple Python function that calculates Fibonacci sequence based on given parameters. Configuration to this example is very simplistic, containing only required values along with a low count of generated parameters for the computation.

The second example uses a Matlab script that accepts multiple arguments, prints their values to console and sleeps itself for a few seconds. The configuration file contains almost all possible configuration options, like the specification of resources directly in the file, which make the example much more interesting if a user wants to use it as a template for his own computation.

4.4.2 Detailed configuration guide

The quick guide provides a basic understanding of task submission and overall possibilities of the CLI. To use the CLI effectively, a user must have knowledge about the task configuration details before submission. For this purpose, second guide was created – **docs/CONFIGURATION.md**.

The guide describes every part of configuration file while using *Quick guide*'s modified examples described in the previous section.

The aim of the guide is for a user to gain to ability modify and create the file for his use-case.

Technical details of realised CLI

While the previous chapter 4 describes possibilities of the tool and usage from a user perspective, this chapter is dedicated to used technologies, patterns and solutions for encountered issues.

5.1 Technology stack

This section describes used technologies used for implementing the CLI and the reasons why they were selected.

5.1.1 Language and build system

The CLI is written in Kotlin [44], which is statically typed programming language running on JVM (Java Virtual Machine). One of the main motivations was to use mainstream technology that is supported by MetaCentrum's modular system.

Another aspect is the ability to use an existing large quantity of stable libraries and tools from the Java ecosystem. Thanks to the CLI running on a virtual machine, distributed compiled files (releases) do not need to depend on specific system version and thus it is not needed to create multiple distribution versions.

The reason why Java itself wasn't chosen was to be able to use new and advanced programming features like lambda expressions and simultaneously not depend on specific JDK that brings such features. This is the case where Kotlin excels, as it is possible to target old JDK versions and yet use modern features of the language [45].

The future of the tool can benefit from static typing, which provides an option for other developers to extend the functionality of the created tool much more efficiently thanks to existing interfaces and defined corresponding types.

The used build system for assembling, building and management of libraries is Gradle [46]. Unified building process via Gradle ensures that each assembling is done the same way on every machine and that the resulting build will have the same versions of libraries that will be automatically downloaded during the build.

To avoid the need for a developer to download and manage Gradle versions locally, Gradle wrapper is used. This simplifies build even more, as the only thing that needs to be done is to execute:

```
$ ./gradlew build installDist
```

The Gradle wrapper then downloads Gradle automatically based on version specified in source files and uses it for building source code.

5.1.2 File format

The chosen file format for serialization of user configuration files is *YAML Ain't Markup Language*, shortened **YAML**. The goal was to choose a standardized format that is both human-readable, easily deserializable and not having a strict form which a user must follow. All of these features are fulfilled by the format.

For serialization and deserialization of YAML files, multiple Java or Kotlin libraries exists: SnakeYAML, YamlBeans, Jackson Yaml Dataformat or Kaml.

Used library for the created tool is Jackson deserializer, extended with mentioned *Jackson Yaml DataFormat* for working with YML format. The reason for Jackson is that possible change or extending support for another format would be much more efficient, because Jackson can be easily extended to support other standardized formats like JSON or XML, without the need to completely rewrite deserialization implementation.

5.2 Implementation details

5.2.1 Notable interfaces

The key focus when designing the architecture of CLI was to ensure extensibility of supported features. These features are either user commands or core functionality that is done under the hood of CLI (see Table 5.1).

Commands

The action is a representation of user command. Each action belongs to one CLI command and differs in additional arguments that a user has to provide. The data flow of submit command in application is shown in Figure 5.1. *ActionSubmit* and *ActionService* code snippet is in Listing 5.1.

Action	Commands that user of CLI wants to execute
Configurator	Configuration of resources based on user input
ConfigValidator	Validation of configuration file
TaskExecutor	Defines execution steps for specific action

Table 5.1: Overview of notable interfaces

From an implementation perspective, an action is represented by the *Action* interface which contains command specific data and defines which *ActionService* implementation needs to be called for the execution of the action logic.

Listing 5.1: Action and ActionService code snippet

```

1 // action that submits new jobs to queue
2 sealed class ActionSubmit() : Action()
3
4 // ActionService implementation for handling submit action
5 class ActionSubmitService() : ActionService<ActionSubmit> {
6     override fun processAction(argumentAction: ActionSubmit) {
7         // handle action
8     }
9 }

```

ConfigValidator

Data given by a user must be validated so the program properly terminates in case they are not valid. This ensures that CLI won't go into an unexpected state during runtime and will be able to help the user update his configuration so it contains proper values.

ConfigValidator is an interface that contains one type of validation for the *ConfigFile* provided by a user. Found errors are wrapped and represented by *ValidationResult* class (see Listing 5.2). This ensures that users can see all found errors in the controlled output, unlike the case when the CLI would be forced to end after first validation failure.

Configurator

Configuration of the submit command is represented by the *ConfigFile* class, where the initial values are deserialized from the file specified by a user.

As the values do not have to be complete, in the sense that the configuration file does not have to contain all the information needed for submission, additional values must be collected. To gather these missing values, *Configurator* interface (see Listing 5.3) and its implementations exist.

5. TECHNICAL DETAILS OF REALISED CLI

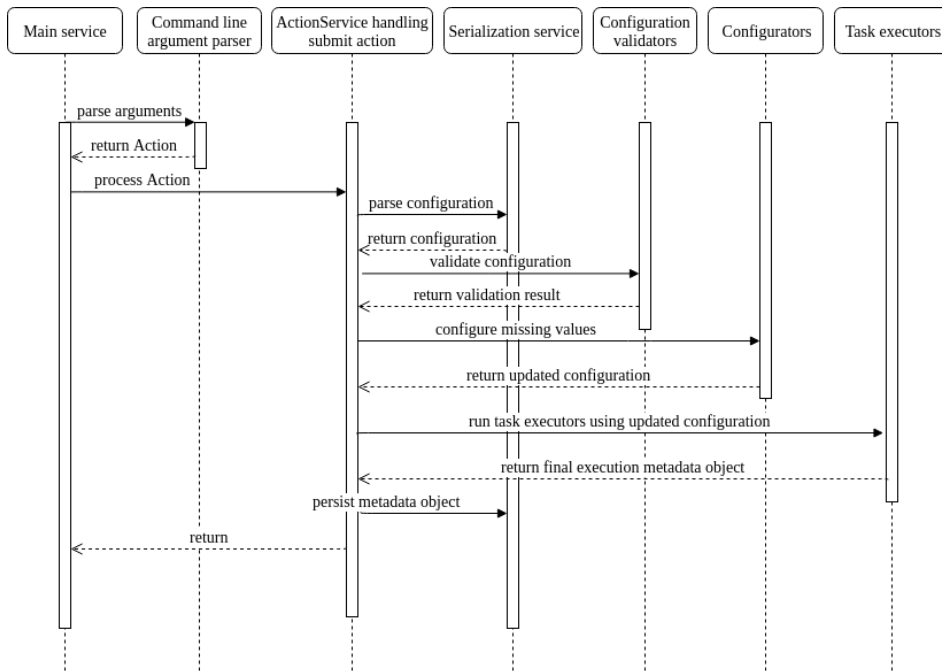
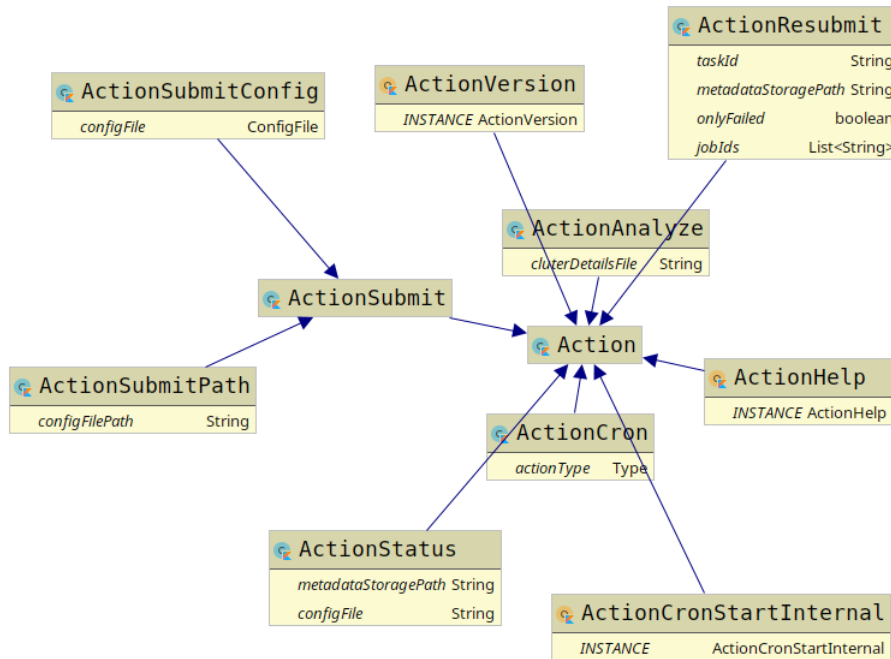


Figure 5.1: High level view of submit action data flow

Figure 5.2: Class diagram for Action types



Powered by yFiles

Listing 5.2: ConfigValidator interface and ValidationResult data class

```
1 interface ConfigValidator {
2     fun validate(configFile: ConfigFile): ValidationResult
3 }
4
5 data class ValidationResult(val messages: List<String> = emptyList(),
6                             val success: Boolean = true) {
7     constructor(message: String, success: Boolean) :
8         this(listOf(message), success)
9     companion object {
10        fun merge(a1: ValidationResult, a2: ValidationResult):
11            ValidationResult {
12            val newList = ArrayList<String>()
13            newList.addAll(a1.messages)
14            newList.addAll(a2.messages)
15            return ValidationResult(newList,
16                a1.success.and(a2.success))
17        }
18
19        fun merge(a1: ValidationResult, newMessages: String):
20            ValidationResult {
21            val newList = ArrayList<String>()
22            newList.addAll(a1.messages)
23            newList.add(newMessages)
24            return ValidationResult(newList, false)
25        }
26    }
27    operator fun plus(newRes: ValidationResult): ValidationResult {
28        return merge(this, newRes)
29    }
30
31    operator fun plus(newMessages: String): ValidationResult {
32        return merge(this, newMessages)
33    }
34 }
```

Listing 5.3: Configurator interface

```

1 interface Configurator {
2     fun configure(config: ConfigFile): ConfigFile
3 }

```

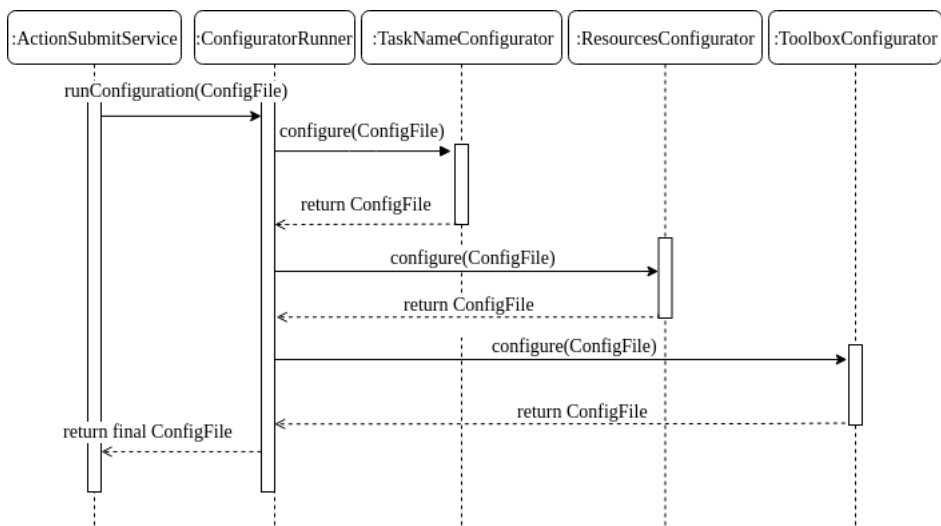


Figure 5.3: Time-flow visualization of Configurators execution before submit action

Implementations are executed in the defined order, where each accepts *ConfigFile* instances, modifies it and returns new one, as the class is immutable and cannot be modified. The created instance is then passed to the next implementation until the last implementation in the chain creates the final instance.

Implementations can be divided into two types: the first one is the computing of additional values automatically based on given data, while the second interactively collect values by asking user in the command prompt. Such values are for example resources wanted for the job or job's name under which CLI remembers current execution.

TaskExecutor

TaskExecutor is an interface that defines and performs one part of execution for a specific *Action* (see implementation in Listing 5.4).

TaskExecutor implementations are the core of the CLI and contain code that is performing actual business logic. The motivation behind such modular architecture and splitting logical components is that multiple *TaskExecutors*

Listing 5.4: TaskExecutor

```
1 interface TaskExecutor {  
2     /**  
3     * Performs execution and returns result in ExecutionMetadata.  
4     */  
5     fun execute(metadata: ExecutionMetadata): ExecutionMetadata  
6 }
```

can be grouped in *task executor groups* and reused for different actions, which improves re-usability of the code and effective implementing new functionality.

TaskExecutor accepts *ExecutionMetadata* class, which contains information about current task like its state, configuration, jobs to be generated, job's current status, etc.

If we would talk only about executors that are used in the submit action, then their job is to prepare an environment, create data, create jobs, submit them to the queue and store information about the task in the proper location (see class diagram in Figure 5.4).

This means execution is split into parts where each is represented by the implementation of the *TaskExecutor*

TaskExecutor interface and its behaviour is similar to previously mentioned interfaces and their implementations. Executors are grouped into the collection and executed in sequence, while their input and output is saved to *ExecutionMetadata* instance (see Figure 5.3 with Configurator example).

Groups of TaskExecutors can be divided either by the *Action* or by the type of task, for example, Matlab or Python. In the case of task type, current implementation reuses almost all executors for both types in the chain except for the one which generates execution script. This composition allows to easily extend currently supported task types with new ones.

5.3 Local environment setup for development

Created CLI heavily depends on the queuing system PBSPro to be available on the target machine. This is quite a complication in the local environment, where the queuing system would have to be installed and even in such case, the installation would have different options available in opposite to the MetaCentrum one.

Such options are available application modules, queues and resources that can be used.

The first issue that needed to be solved is local development and availability of the queuing system itself. As a solution for simple and unified installation process, virtualization using docker was selected.

5. TECHNICAL DETAILS OF REALISED CLI

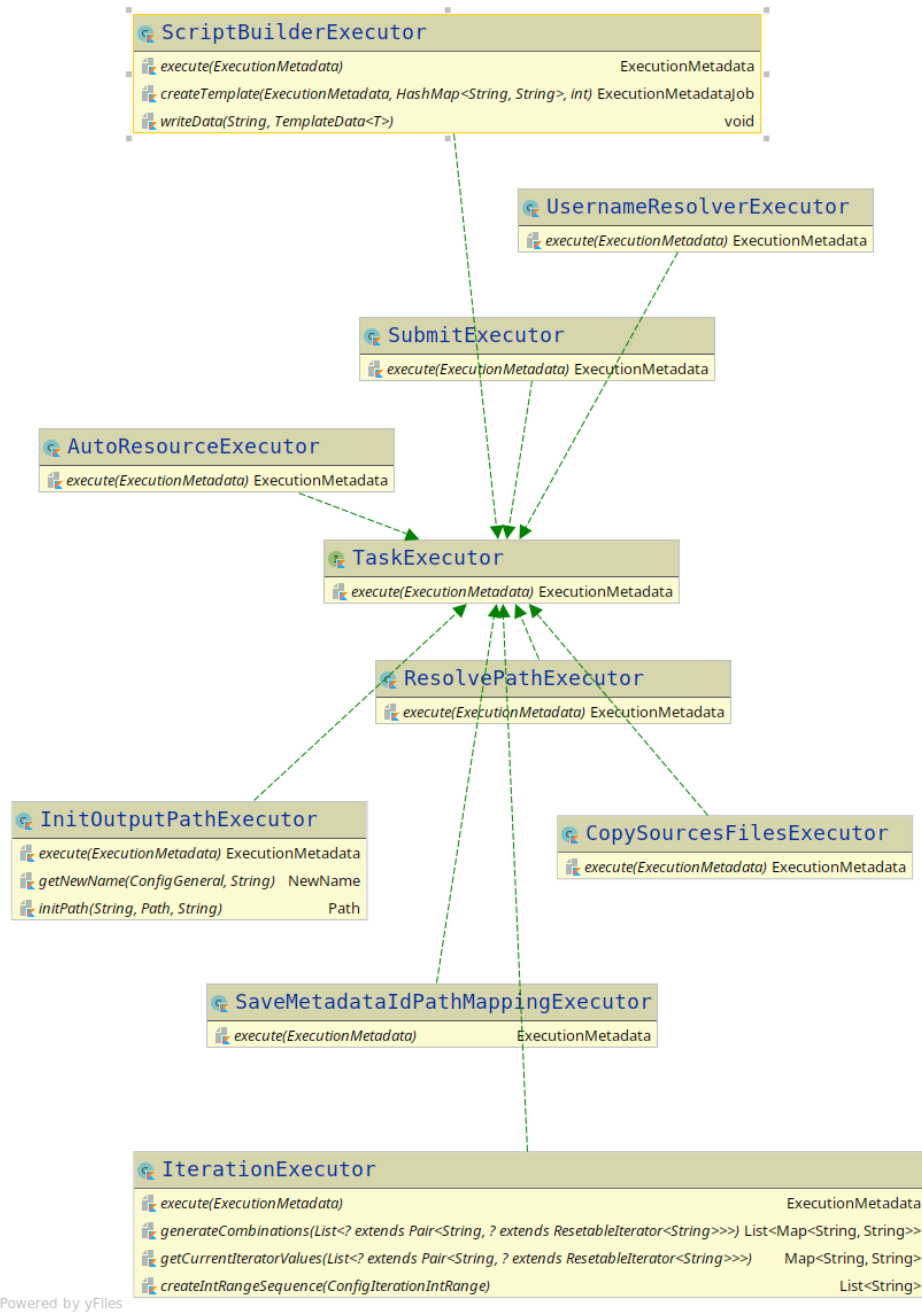


Figure 5.4: Class diagram for all submit TaskExecutors

5.3.1 Docker

A container is a standard unit of software that packages up the code and all its dependencies so the application runs quickly and reliably from one computing environment to another. [47]

Docker container is created from Docker images at runtime when they run on Docker Engine. Docker Engine is available on main platforms like Linux, Windows and Mac OS X systems and ensure that software will run the same on all of them. [47]

In the case of queuing systems, an official image for PBSPro software container exists (see on DockerHub). For the needs of created CLI, this image is extended with some additional functionality, along with few more steps that user must follow for a successful installation.

5.3.2 Local environment solution

The main requirement for basic local development is the ability to run proprietary software like Matlab or just software that needs additional installation steps like Python.

As actual Matlab or Python computation is not needed in the minimal development setup, a script that can be used for mocking software was developed (see Listing 5.5). The script can be added by following a few steps, to the image's PATH variable under the name of the software that needs to be mocked. From the CLI perspective, it then behaves like the actual execution on the target machine. The script emulates an asynchronous task that is completed in random time and does not have a deterministic result as it may sometimes fail.

Listing 5.5: Script for mocking computation in a local development

```
1 #!/bin/bash
2 sleep `grep -m1 -ao '[0-9][0-9][0-9]' /dev/urandom \
3     | sed s/0/10/ | head -n1`
4 echo $@
5 RETURN_ID=`grep -m1 -ao '[0-2]' /dev/urandom | head -n1`
6 echo $RETURN_ID
7 exit $RETURN_ID
```

5.3.3 DEV profile for CLI

The local environment relying on Docker container causes a new issue that must be solved. First is the need for commands to execute in the container environment and second is that available resources in the container are limited. These resources are the number of CPU cores to be used, RAM, queue types, toolboxes and modules.

The solution to these issues is introduction of the environment variable **CLUSTERIZE_PROFILE**. After setting the variable with `dev` value, the CLI behaves differently.

First changed behaviour is that dependency injection provides a different implementation of service that executes terminal commands. In fact, provided service is actually a proxy class of shell service used in production, that prefixes every executed command so it is executed in the docker container. For proper functioning, the container must be created as is noted in 5.3.4.

The second behaviour is again solved thanks to dependency injection configuration, so the provided **Configurator** array contains additional implementation that changes resource configuration in such way, that it wouldn't cause an error when a job is submitted in the local environment.

5.3.4 Docker configuration

Configuration for docker container extends official PBSPro tutorial on how to set up a queuing system in the base container. Step by step solution can be found in attachment under path `sources/src/env/README.md` (see Appendix B).

5.4 Distribution

One of the significant issues with created CLI was to ensure proper functionality on the front-end node of MetaCentrum grid network. The first issue is that `JAVA_HOME` variable is not available right after a user login but only after adding the JDK module to the current session. This causes problems when CLI is executed as a cron job for watching tasks. Another task was to ensure easy distribution option for users and way to update CLI version if a new release exists.

To deal with these issues, bash scripts for installation and for wrapping actual execution were created.

5.4.1 Wrapping execution script

Standard Gradle distribution plugin creates an installation bundle, which contains libraries, application jar file and execution script. The execution script depends on JDK and `JAVA_HOME` environment availability in the current session. Unfortunately, this is not possible on the MetaCentrum environment, where Java must be set from available modules in each session.

Although CLI could depend on a user to set module every time before running tool, this becomes a significant issue when CLI is executed as a *cron job*, where it is difficult to add the module to the current session before execution.

This is solved by installation script to create a wrapping bash script, which sets `JAVA_HOME` with the value known during installation and then executes the tool with given parameters (see Listing 5.6).

Listing 5.6: Wrapping execution script. `HOME` represents absolute path to user directory

```
1  #/bin/bash
2  export JAVA_HOME=/packages/run/jdk-8/current
3  HOME/.clusterize/clusterize/bin/clusterize "$@"
```

The alternative solution for the script content would be loading Java using a modular system. The disadvantage and reason why it wasn't used is that this creates dependency on the module. This would cause issues on a system where the module is not available or if the module name is changed. The advantage of determining the absolute path to `JAVA_HOME` during installation is that potential issues can be solved with the re-installation of the tool.

Assessment of created CLI and future plans

The chapter is dedicated to the assessment of the CLI, description of implemented features and also discussion about future plans for further development of the tool.

6.1 Assessment of created CLI

The created CLI solves core issues that were introduced in chapter 2. The core idea of the CLI is to track whole computation as one task composed of hundreds and thousands of jobs that are submitted to the queuing system and represent actual computation. The created solution is able to generate multiple configurations and successfully submit them to the queuing system while helping a user to configure some of the parameters required for computation.

These functionalities are able to heavily improve the workflow of users and to provide features that wouldn't be possible without the tool.

6.1.1 Resource management

The CLI supports basic configuration of wall-time, CPU, memory, storage and GPU cores. These values can be either inputted via the configuration file or interactively as part of *submit* command (see section 4.2). Thanks to that, a user has the structure to follow and thus avoid errors that could occur during submission via PBS because of incomplete or invalid settings.

Apart from hardware resources, CLI also supports configuration of needed modules and toolboxes, which are essential on the MetaCentrum grid for the execution of computation scripts.

6.1.2 Notification and re-submission

Thanks to tracking whole computation as one task, the CLI is able to check task state based on submitted jobs and notifies a user when all jobs in task finish using email.

The email contains result and information about all the tasks that finished. The email also contains information about the task's jobs to provide a useful general overview of the state, without the need to login on the front-end node (see Listing 6.1).

Listing 6.1: Shortened version of the notification email when example Python task finishes

```
1 ===== Task information =====
2
3 Task name:          RExample
4 Creation time:      2019-04-30T10:23:46.905
5 Update time:       2019-04-30T10:23:48.382
6 Output path:
   /home/clusterize/examples/out-python/task-11__2019-04-30_10-23-46
7
8
9 ===== Resources =====
10 Profile:           CUSTOM
11 modules:
12 toolboxes:         python-3.6.2-gcc,
13
14 walltime:          00:04:00
15 chunks:            1
16 mem:               5gb
17 ncpus:             8
18 scratchLocal:     1gb
19 ngpus:
20
21
22 1 - RExample - 30/04/2019 10:23 - DONE
23 ===== Job history =====
24
25 Total resubmits count: 0
26
27
28 Job_0) Path: task-11__2019-04-30_10-23-46/0
29   - State: DONE
30   - Start: 2019-04-30T08:25:52 | End: 2019-04-30T08:25:55
31   - Resubmit count: 0
```

The notification mechanism can be configured to resubmit failed jobs while keeping the failed executions data intact.

6.1.3 Versioning

The CLI versions each submission in a systematic way, that allows checking historical outputs and re-submission of tasks. The versioning is tightly connected with the status checking, as part of the versioned files are metadata, which contains not only configuration used for parameterization of jobs but also locations of sources and latest status of jobs.

6.2 Further development plans

The plan for the future is to test the CLI by current users in the production environment. Bug-fixes and new features will be implemented based on the feedback. It is also available for other developers to participate in the project using open source project on the GitHub platform.

At the moment, all current users are using MetaCentrum NGI as their only platform for grid computing, but it is possible to extend support for other grid networks and clusters.

6.2.1 New releases and user participation

User participation is required for further development of the tool. Although the CLI solves some of the detected issues, its range can be much wider than it is currently known. For this, user participation is crucial.

Releasing new versions of the CLI is automatized process supported by the integration of Travis platform and GitHub, making it possible to release versions with each small improvement. The CLI can be updated version executing the installation bash command with the new version.

Thanks to that, users feedback can be gathered in short intervals without increasing the complexity in the software cycle. Gathering users feedback quickly allows directing development based on users requirements much more efficiently.

Further improvement is then implementation of auto-updater directly to the CLI along with automatic checking for a new version.

6.2.2 Resource profiles

The CLI offers configuration of resources either using the configuration file or by interactively within the *submit* command. The more user-friendly approach is to offer profiles for most common configurations. These could be based on available queues and their resources.

Another approach is to determine the best configuration by analyzing the computed task and capacity of queues on the grid. Determination of parameters is a very difficult task that needs to analyse the computed task by executing it with multiple configurations. The parameters could be selected

based on the benchmark. The issue here is the possibility of tasks' jobs complexity to vary and thus selection of parameters for each job would have to be dynamic.

The state of queues is another attribute that affects computations. If the queue is filled by jobs of other users it can mean that computation will spend a long time in the queue waiting.

6.2.3 Task types

The CLI currently supports **Python** and **Matlab** task types. One of the possible features to be added is to define a new task type definition, that executes any bash command specified by the user. Implementation of such task type is almost the same as the Python one with the difference that additional automatized configuration for the Python environment would not be enabled.

6.2.4 Versioning

The current implementation saves metadata about executions. These metadata contain the used configuration of the task, state of submitted jobs and absolute location of files needed for computation (sources) and output directory.

This local versioning allows keeping track of past executions and straightforward re-submission of the whole task if the location of sources is still valid.

First, possible improvement of versioning is to split source files into computation files and input data sets. This allows versioning computation files separately from input data. Versioning input along with other files is not a trivial task as the potential size could easily fill up available storage.

The solution for storage issues is to either limit versioning data exceeding predefined size or to integrate tools that are built for tracking large data-sets. More detailed analysis of the solution was presented in section 3.1.

Another aspect is to support versioning of all files by integration VCS system and to use the remote repository to track these files.

Conclusion

The issues occurring in the grid network have a heavy impact on the productivity of users who are using it for hypothesis testing, algorithms and other computational tasks. Users are forced to manually set up the whole workflow by reusing existing scripts which then lacks structure and important features like versioning files used for executions, overall management of executed jobs and proper notifications of finished executions.

This thesis proposes a possible solution to these issues and presents an innovative view on job management of grid computation tasks with a focus on the MetaCentrum grid. This view can be also applied to all systems that use PBS submission system.

The created CLI provides an extension of base submission commands by tracking computation as one task divided into multiple jobs. With this abstract view on the computation, it was possible to implement features like the generation of high-dimensional space of computation parameters for a given task, versioning of executions, advanced notification system and automatic re-submission of failed jobs.

Additional development is planned and will be based on the feedback by users, which currently undergo a first wave of testing in the production usage. The overall plan is to extend functionality even further and to be able to cover most of the use-cases that occurs while computing on the grid via PBS submission software.

Bibliography

- [1] MetaCentrum statistics 2018. [cit. 2019-04-22]. Available from: <https://metavo.metacentrum.cz/cs/state/stats/2018/>
- [2] Use cases — Machine Learning Version Control System. [cit. 2019-03-31]. Available from: <https://dvc.org/doc/use-cases/data-and-model-files-versioning/>
- [3] Comparison of Grid Computing vs. Cluster Computing. [cit. 2019-04-26]. Available from: http://www.jatit.org/research/introduction_grid_computing.htm
- [4] PBSPro User's guide. [cit. 2019-04-06]. Available from: <https://www.pbsworks.com/pdfs/PBSProUserGuide13.1.pdf>
- [5] Resource Allocation Policy — IT4Innovations Documentation. [cit. 2019-04-21]. Available from: <https://docs.it4i.cz/general/resources-allocation-policy/#job-queue-policies>
- [6] Azure Functions scale and hosting — Microsoft Docs. [cit. 2019-05-03]. Available from: <https://docs.microsoft.com/en-us/azure/azure-functions/functions-scale/>
- [7] What is Cloud Computing? – Amazon Web Services. [cit. 2019-05-03]. Available from: <https://docs.microsoft.com/en-us/azure/app-service/overview-hosting-plans>
- [8] What is Computer Cluster? – Definition from Techopedia. [cit. 2019-02-24]. Available from: <https://www.techopedia.com/definition/6581/computer-cluster>
- [9] Rouse, M. What is grid computing? – Definition from WhatIs.com. Mar 2013, [cit. 2019-02-24]. Available from: <https://searchdatacenter.techtarget.com/definition/grid-computing>

BIBLIOGRAPHY

- [10] Foster, I. What is the Grid? A Three Point Checklist. June 2002, [cit. 2019-02-24]. Available from: <http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf>
- [11] PBSPro. [cit. 2019-02-24]. Available from: http://www.softpanorama.org/HPC/PBS_and_derivatives/index.shtml
- [12] Industry-leading workload manager and jobscheduler for high-performance computing. [cit. 2019-02-24]. Available from: <https://www.pbspro.org/>
- [13] Foster, I.; Kesselman, C.; et al. The Anatomy of the Grid. *Wiley Series in Communications Networking Distributed Systems Grid Computing*: p. 169–197, doi:10.1002/0470867167.ch6.
- [14] Metacentrum NGI. [cit. 2019-02-24]. Available from: <https://www.metacentrum.cz/en/>
- [15] Registration form. [cit. 2019-04-22]. Available from: <https://metavo.metacentrum.cz/en/application/index.html/>
- [16] MetaCentrum Grid/. [cit. 2019-02-24]. Available from: <https://www.metacentrum.cz/en/Sluzby/Grid/>
- [17] CERIT Scientific Cloud. [cit. 2019-02-24]. Available from: <https://www.cerit-sc.cz/>
- [18] CEITEC – Výzkumné centrum. [cit. 2019-02-24]. Available from: <https://www.ceitec.eu/>
- [19] Zprovoznění clusteru luna. [cit. 2019-02-24]. Available from: https://metavo.metacentrum.cz/cs/news/novinka_2014_0001.html
- [20] Cloud and High Performance Computing. [cit. 2019-04-21]. Available from: <https://it.muni.cz/en/categories/cloud-and-high-performance-computing/>
- [21] Elixir – Metacentrum. [cit. 2019-02-24]. Available from: <https://wiki.metacentrum.cz/wiki/Elixir>
- [22] Faculty of Mechatronics, Informatics and Interdisciplinary Studies. [cit. 2019-04-21]. Available from: <https://www.fm.tul.cz/veda-a-vyzkum/vypocetni-cluster-charon/informace-pro-uzivatele>
- [23] Nový cluster Nympha. [cit. 2019-04-21]. Available from: https://support.zcu.cz/index.php/Aktuality:Nov%C3%BD_cluster_Nympha
- [24] Supercomputing center at CVUT. [cit. 2019-04-21]. Available from: <https://ist.cvut.cz/nase-sluzby/superpocitani/>

-
- [25] What is IT4Innovations? — IT4Innovations. [cit. 2019-04-21]. Available from: <https://www.it4i.cz/about-us/what-is-it4innovations/?lang=en>
- [26] Announcing Amazon Elastic Compute Cloud (Amazon EC2) - beta. Available from: <https://aws.amazon.com/about-aws/whats-new/2006/08/24/announcing-amazon-elastic-compute-cloud-amazon-ec2---beta/>
- [27] AWS Lambda – Serverless Compute – Amazon Web Services. [cit. 2019-04-22]. Available from: <https://aws.amazon.com/lambda/>
- [28] AWS Lambda Limits – AWS Lambda. [cit. 2019-04-22]. Available from: <https://docs.aws.amazon.com/lambda/latest/dg/limits.html>
- [29] AWS Fargate – Run containers without having to manage servers or clusters. [cit. 2019-04-22]. Available from: <https://aws.amazon.com/fargate/>
- [30] Cloud Functions - Event-driven Serverless Computing — Cloud Functions — Google Cloud. [cit. 2019-04-22]. Available from: <https://cloud.google.com/functions/>
- [31] Quotas — Cloud Functions Documentation — Google Cloud. [cit. 2019-04-22]. Available from: <https://cloud.google.com/functions/quotas>
- [32] Azure Functions Overview — Microsoft Docs. [cit. 2019-05-03]. Available from: <https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview/>
- [33] Application modules. [cit. 2019-04-04]. Available from: https://wiki.metacentrum.cz/wiki/Application_modules/
- [34] Matlab – MetaCentrum. [cit. 2019-04-04]. Available from: <https://wiki.metacentrum.cz/wiki/Matlab/>
- [35] OSullivan, B. *Mercurial the definitive guide*. OReilly Media, 2009.
- [36] What is my disk quota. [cit. 2019-04-06]. Available from: <https://help.github.com/en/articles/what-is-my-disk-quota>
- [37] What kind of limits do you have on repository/file size? [cit. 2019-04-27]. Available from: <https://confluence.atlassian.com/bitbucket/what-kind-of-limits-do-you-have-on-repository-file-size-273877699.html>
- [38] Bhattacharjee, S.; Chavan, A.; et al. Principles of dataset versioning. *Proceedings of the VLDB Endowment*, volume 8, no. 12, 2015: p. 1346–1357, doi:10.14778/2824032.2824035.

BIBLIOGRAPHY

- [39] Data Science Version Control System. [cit. 2019-03-31]. Available from: <https://dvc.org/>
- [40] Git large File Storage. [cit. 2019-04-06]. Available from: <https://git-lfs.github.com/>
- [41] Cloud Object Storage — Amazon Simple Storage Service. [cit. 2019-04-06]. Available from: <https://aws.amazon.com/s3/>
- [42] Core Concepts for Beginners – Travis CI. [cit. 2019-04-22]. Available from: <https://docs.travis-ci.com/user/for-beginners/>
- [43] The Official YAML Web Site. [cit. 2019-03-31]. Available from: <https://yaml.org/>
- [44] Kotlin. [cit. 2019-04-06]. Available from: <https://kotlinlang.org/>
- [45] Using Gradle – Kotlin programming language. [cit. 2019-04-06]. Available from: <https://kotlinlang.org/docs/reference/using-gradle.html/>
- [46] Gradle. [cit. 2019-04-06]. Available from: <https://gradle.org/>
- [47] What is a Container? — Docker. [cit. 2019-03-10]. Available from: <https://www.docker.com/resources/what-container>

List of used terms

- CLI** Command line interface
- CPU** Central processing unit
- GPU** Graphics processing unit
- JDK** Java developmen kit
- PBS** Portable Batch System
- NGI** National Grid Initiative
- AWS** Amazon Web Services
- VCS** Version Control System
- LAN** Local Area Network
- WAN** Wide Area Network

Content of attached CD

readme.txt.....	content overview
build.....	build output
bin	
clusterize	Execution script for unix systems
lib.....	Jar files with source code and libraries
sources.....	Code repository content
docs.....	User documentation markdown files
examples.....	Example configurations and supporting files
src.....	Source code
README.md	Base information about CLI
thesis	Source files of thesis text in L ^A T _E X format