PREDICT IT

Administrator Guide

Version 1.5

CC BY-NC-ND UCit 2020

Copyright

Copyright © 2015-2020, UCit.
All rights reserved.

We Would Like to Hear from You

You can help us make this document better by telling us what you think of the content, organization, and usefulness of the information. If you find an error or just want to make a suggestion for improving this document, please send feedback to UCit Support.
Although the information in this document has been carefully reviewed, UCit does not warrant it to be free of errors or omissions. UCit reserves the right to make corrections, updates, revisions, or changes to the information in this document.

UNLESS OTHERWISE EXPRESSLY STATED BY UCit, THE PROGRAM DESCRIBED IN THIS DOCUMENT IS PROVIDED "AS IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL UCit BE LIABLE TO ANYONE FOR SPECIAL, COLLATERAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING WITHOUT LIMITATION ANY LOST PROFITS, DATA, OR SAVINGS, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS PROGRAM.

Document Redistribution and Translation

This document is protected by copyright and you may not redistribute or translate it into another language, in part or in whole, without the express written permission of UCit.

CC BY-NC-ND UCit 2020

Trademarks

Linux is the registered trademark of Linus Torvalds in the United States and other countries.

Firefox® and Mozilla® are trademarks or registered trademarks of the Mozilla Foundation in the United States and/or other countries.

Apple®, Mac®, Mac® OS X® and Apple® Safari® are trademarks or registered trademarks of Apple, Inc. in the United States and other countries.

Altair® PBS Professional® is a trademark of Altair Engineering, Inc.

SLURM™ is a trademark of SchedMD LLC.

Google™ and Chrome™ are trademarks of Google Inc.

Red Hat® is a trademark of Red Hat, Inc.

Sun® and JavaScript® are registered trademarks of Oracle and/or its affiliates.

Other names mentioned in this document may be trademarks of their respective owners.

Learn about Predict-IT and UCit products

World Wide Web page

You can find the latest information about UCit Predict-IT on its web site https://www.ucit.fr/en/predict-it/
For more information about other UCit products and about the professional services provided by UCit you can refer to the company's web site https://www.ucit.fr/

UCit Predict-IT Download

The latest Predict-IT downloadable package and documentation are available at:
https://www.ucit.fr/download-products

UCit Support Contacts

Use one of the following to contact UCit technical support.

- Email: helpdesk@ucit.fr
- Helpdesk portal: https://helpdesk.ucit.fr/

CC BY-NC-ND UCit 2020

Table of Contents

CC BY-NC-ND UCit 2020

CC BY-NC-ND UCit 2020

1 - Introduction

The job scheduler is the central point for your HPC infrastructure. It dispatches and monitors the jobs on the available resources, keeping track of the jobs' allocation and their state. The generated data are stored in the job scheduler's logs, from which valuable information can be extracted in order to understand the cluster behavior.

For example, when jobs are submitted on the cluster, some of them may not terminate within the specified time limit (walltime) and consequently no results are obtained. This way, to increase cluster profitability and production, it is necessary to enforce that the submitted computations will end up correctly.

That is why we have developed **Predict-IT**. By analyzing the job-submission historical data in the logs, Predict-IT creates a computational model of your cluster based on machine-learning techniques. Such model is then applied at the moment of job submission to **identify which jobs are likely to end-up in failure**; moreover, it also provides **recommendations for the walltime you need to specify to prevent job failure**, and the **amount of memory** (max RSS) a job will consume per node.

Configured specifically for your cluster, Predict-IT **steadily improves over time**: it adapts to your HPC environment by learning from your cluster logs and newly arrived jobs, each time becoming more and more accurate in its predictions. The more data you feed it, the higher the prediction accuracy can be.

CC BY-NC-ND UCit 2020

2 - How it works

Predict-IT **learns from the job scheduler logs** and **creates a model** of your cluster (HPC system) which will be used to perform predictions at the time of job submission. This is done in order to provide the user with predictions for the following jobs characteristics:

- the jobs finishing state (or simply **STATE**)
- the job maximum memory consumption, i.e., maximum resident set size (**maxRSS**) on a node
- the waiting time (**WAITTIME**, i.e. the time between job submission and start time)
- the jobs maximum execution time (also called **EXECTIME** or **WALLTIME**)
- the time to result (**TIMETORESULT**, i.e. the time between job submission and the end of execution).

The operation of Predict-IT is done in two phases. In the first one (**learning phase**), the algorithms embedded in Predict-IT devise a set of models for your cluster, based on the history logs of your system. Those models are able to forecast the correct STATE of a job, its maxRSS , its WAITTIME, WALLTIME and TIMETORESULT up to a certain accuracy. In the second one (**prediction phase**), the trained models are queried by the user in order to obtain predictions for a particular job. Figure 1 depicts the basic flowchart for Predict-IT.

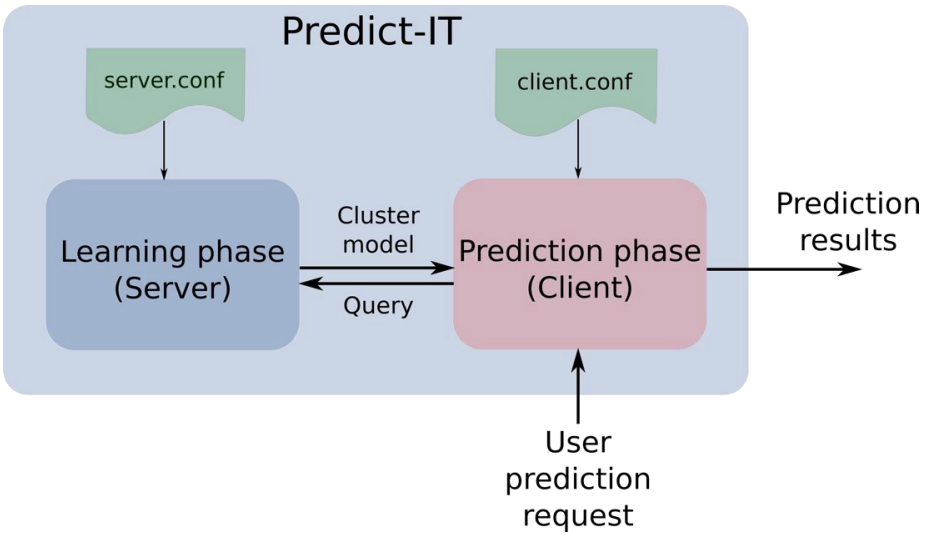

Figure 1: Overview of Predict-IT structure

CC BY-NC-ND UCit 2020

At submission time, the user provides specific parameter values to the job scheduler. For example, one can set the number of requested CPUs, requested memory (REQMEM), partition etc. Additionally, each user has an identity (UID) and belongs to a specific group (GID) of the cluster. All those parameters are combined in the "user prediction request" (see Figure 1), which is then passed into Predict-IT to obtain predictions for that job (or group of jobs).
Predict-IT is composed by two components, namely the **server** and the **client**:

- Server: that's where the learning phase happens. This environment must be activated first, otherwise no cluster model will be created, and consequently no prediction will be possible.
- Client: after the learning phase is completed, the user can request predictions from the client side by passing the job submission parameters. The client queries the server which then uses the submission parameters together with the cluster model in order to forecast the STATE, MEMORY, WAITTIME, WALLTIME and TIMETORESULT.

The configuration files `server.conf` and `client.conf` define parameters for the server side and the client side, respectively. Those parameters are user-customizable: their values depend on the characteristics of the dataset and the type of predictions the user wants to obtain.

3 - Key Features

Predict-IT provides recommendations for the following jobs' characteristics:

- Job STATE: Detects the risk of a job finishing in timeout, i.e., reaching the walltime (maximum time allowed for running a job);
- Job EXECTIME (WALLTIME): Predicts the walltime that should be set by the user at the submission to prevent it resulting in timeout;
- Job MEMORY (maxRSS): Predicts the memory requirement that should be set by the user at the submission.

Predict-IT keeps learning over time: the more data, the higher the accuracy.

CC BY-NC-ND UCit 2020

4 - Requirements

4.1 - Hardware

Predict-IT should be installed on a virtual machine or physical server separated from the main cluster frontend node. Minimal hardware requirements are:

- CPU: 4 cores
- Memory: 16GB
- Disk space: At least 4GB of free disk space (for tool-generated files and installation files)

Note that the strongest requirement is the RAM: the more jobs you have in your logs, the more RAM you need.

4.2 - System

- Operating Systems: RHEL 6/7
- A user account, different from the root account to run the server (e.g., pituser)
- Required dependencies: jq

4.3 - Job scheduler

The following job schedulers are supported:

Name	Version	Notes
SLURM™	14.11.6 or later	SLURM™ binaries must be installed on the Predict-IT Server host, and in the PATH. The sacct command must be usable by the Predict-IT user (e.g., pituser). SLURM™ SlurmDBD server must be reachable from the Predict-IT Server host. Accounting must be turned on with SlurmDBD: `AccountingStorageType` must be set to `accounting_storage/slurmdbd`, `JobAcctGatherType` must be set to `jobacct_gather/linux`[1] (see https://slurm.schedmd.com/accounting.html for more details)
Altair® PBS Professional®	13.0 or later	The PBS Professional® logging directory must be readable from the Predict-IT Server Host (default directory is `<PBS_HOME>/server_priv/accounting/`).
Torque	5.0 or later	The Torque logging directory must be readable from the Predict-IT Server Host (e.g., `<TORQUEROOT>/server_priv/accounting/`).

[1] `jobacct_gather/cgroup` does not gather correctly memory usage in SLURM™ version lower than 17.02

CC BY-NC-ND UCit 2020

The job scheduler must be accessible from Predict-IT Server host. If not, you will only be able to train Predict-IT with an input file; in this case the prediction model cannot be frequently updated automatically (see option `inputFile` in section "8.2 - Extract job scheduler logs

If you do not want Predict-IT to automatically and periodically extract new data from the job scheduler, you can use the scripts provided in `<INSTALLATION_PATH>/bin/extractData`. The first argument is the name of the job scheduler, subsequent arguments may vary depending on the job scheduler (type `<INSTALLATION_PATH>/bin/extractData JOBSCHED -h`, with `JOBSCHED`: `pbs`, `torque`, `slurm` or `sge`). The data extraction relies on different methods depending on the job scheduler:

- SLURM™: relies only on `sacct` and `scontrol`.
- Torque and Altair® PBS Professional®: extracts raw data from accounting files, and from `pbsnodes` and `qstat`.
- Grid Engine: extracts raw data from accounting file

Each script creates 3 text files in the current directory:

- **historical data on jobs**: `HOSTNAME_DATE.jobs`
  - From a given date up to now
  - Retrieve all data gathered by the job scheduler (job id, requested/obtained resources, node list, submit/start/run times…)
  - For confidentiality reasons, the output is anonymized: by default, usernames and groups are not retrieved, only UID and GID are
- current **list of nodes** and their description: `HOSTNAME_DATE.nodes`
  - Node name
  - Number of cores, number of cores per socket, memory
  - Specific resources (features described in the job scheduler)
- current **list of partitions/queues** and their description: `HOSTNAME_DATE.partitions`

The execution of the script is very lightweight. No processing of the data is done.
Each script comes with its own online help, run `./extractXXXXData.sh -h` to see this help.

For Predict-IT, only the first file is currently relevant: `HOSTNAME_DATE.jobs`, you need to specify the path to this file in the `server.conf` with the `inputFile` in the `jobScheduler` section.

CC BY-NC-ND UCit 2020

8.3 - Server configuration file ”).

4.4 - Web browser

Predict-IT produces HTML which can be viewed with most popular browsers. Generally speaking, Predict-IT supports the latest versions of each major platform’s (Linux®, Apple® Mac® OS X®, and Microsoft® Windows®) default browsers (Google™ Chrome™, Mozilla Firefox®, Apple® Safari®, Microsoft® Edge®).
JavaScript® must be enabled on browsers.

5 - License

You need a valid license to install and run Predict-IT. If you do not have one yet, please contact helpdesk@ucit.fr or your Predict-IT reseller.

CC BY-NC-ND UCit 2020

6 - Installation

Predict-IT is installed via the execution of the file `predictit-xx_rhel-yy.run` (with `xx` the version of Predict-IT, and `yy` the version of RHEL)

1. Save the `predictit-xx_rhel-yy.run` file in a folder of your choice, and execute it:
   `./predictit-xx_rhel-yy.run`
2. The files will be uncompressed, and the following will be printed:

```
################################################################
# Welcome to Predict-IT installer
################################################################

This installer will guide you during the installation of Predict-IT.
################################################################
Press "Enter" to continue or press "q" to quit:
```

   Press "Enter" to continue.
3. The license agreement is shown. To continue with the installation, you should type "`accept`", otherwise type "`quit`" to decline (the latter will terminate the installation).
4. Choose the installation path (in this example the installation path is "`/home/pituser/predictit`") and press "Enter":

```
Enter the path where Predict-IT must be installed.
################################################################
Installation directory:
/home/pituser/predictit
```

5. Then specify which user will be running the Predict-IT service

```
Enter the user who will be executing Predict-IT server.
################################################################
User [pituser]:
```

6. Optionally, install the service files

```
Do you want to install and register init.d scripts? (you need to be root)
################################################################
Install init.d scripts (y or n):
y
```

7. The files will be installed in the chosen folder:

```
Installation of Predict-IT is now complete
Installation directory: /home/pituser/predicit
Log file: /tmp/Predict-IT-install-1.0-2018-02-02_12:44:31.log
################################################################
Press "Enter" to exit
```


CC BY-NC-ND UCit 2020

You will also need to copy your license file into `<INSTALLATION_PATH>/license/license.lic`. You can also place it in another directory, in this case you will need to edit the `<INSTALLATION_PATH>/conf/server.conf` file and set the `licenseFile` parameter with the path to the license file.

To start the Predict-IT service, just run
`service predictit start` on RHEL 6
`systemctl start predictit` on RHEL 7

7 - Usage

7.1 - Starting the Server

Before starting to predict for the submitted jobs, Predict-IT needs to extract information from the cluster logs and build a computational model of your HPC system. This is done as follows:

1. Go inside the Predict-IT installation folder (`cd <INSTALLATION_PATH>`). There you will find the folder `conf/` which stores the configuration files for both the server and client sides. All the parameters that control how Predict-IT learns from your cluster are defined in the `server.conf` file. Some examples of parameters that might be edited are:
    - `host` (default `127.0.01`): the host the server listens on;
    - `port` (default `9999`): the port server listens on;
    - `logLevel` (default is INFO/`20`): controls the amount of information that is displayed in the log file;
    - `driver`: the job scheduler driver to use. Predict-IT can automatically extract job scheduler's historical data on a periodic basis if this parameter is specified (along with the
      `inputFile` (optional): manually extract the cluster logs by executing the provided job-scheduler data extraction scripts provided with Predict-IT (see section "$

http://localhost:9999/1.1/

{
"action":

```
    "message": "Problem while parsing your input data",
    "status": 500,
    "sub_code": 1
}
```

").
The complete list of server-side parameters is provided in the section "8.2 - Extract job scheduler logs

If you do not want Predict-IT to automatically and periodically extract new data from the job scheduler, you can use the scripts provided in `<INSTALLATION_PATH>/bin/extractData`. The first argument is the name of the job scheduler, subsequent arguments may vary depending on the job scheduler (type `<INSTALLATION_PATH>/bin/extractData JOBSCHED -h`, with `JOBSCHED`: `pbs`, `torque`, `slurm` or `sge`). The data extraction relies on different methods depending on the job scheduler:

CC BY-NC-ND UCit 2020

- SLURM™: relies only on `sacct` and `scontrol`.
- Torque and Altair® PBS Professional®: extracts raw data from accounting files, and from `pbsnodes` and `qstat`.
- Grid Engine: extracts raw data from accounting file

Each script creates 3 text files in the current directory:

- **historical data on jobs**: `HOSTNAME_DATE.jobs`
  - From a given date up to now
  - Retrieve all data gathered by the job scheduler (job id, requested/obtained resources, node list, submit/start/run times…)
  - For confidentiality reasons, the output is anonymized: by default, usernames and groups are not retrieved, only UID and GID are
- current **list of nodes** and their description: `HOSTNAME_DATE.nodes`
  - Node name
  - Number of cores, number of cores per socket, memory
  - Specific resources (features described in the job scheduler)
- current **list of partitions/queues** and their description: `HOSTNAME_DATE.partitions`

The execution of the script is very lightweight. No processing of the data is done.
Each script comes with its own online help, run `./extractXXXXData.sh -h` to see this help.

For Predict-IT, only the first file is currently relevant: `HOSTNAME_DATE.jobs`, you need to specify the path to this file in the `server.conf` with the `inputFile` in the `jobScheduler` section.

CC BY-NC-ND UCit 2020

8.3 - Server configuration file ”.

2. Once finished editing `server.conf`, save it and start the server:
   a. Either run it manually by running the `predictit-server` binary:
      `./bin/predictit-server`
      The server comes with its own online help. To display it, just type `./bin/predictit-server -h`

```
Predict-IT server

optional arguments:
  -h, --help     Show this help message and exit
  -v, --version  Print version and exit
  -c CONFIG, --config CONFIG
                 Configuration file path (can also be set with the
                 PIT_SERVER_CONF environment variable)
```


CC BY-NC-ND UCit 2020

b. Or start the service
`service predictit start` on RHEL 6
`systemctl start predictit` on RHEL 7

This starts the learning task. Some metrics will be displayed in the terminal prompt and/or saved in the Predict-IT log files. The learning phase is finished when the following line is displayed in the logs:

```
======= End of test metrics =======
2018-01-10 20:18:58,969 : INFO : prediction.State : trainingTask._job Listener
: Training job finished.
```

3. Now, access `http://host:port/metrics` (in case you are using the default values, `host=127.0.0.1` and `port=9999`) from your browser. This page displays the metrics for training and testing the computational model of the cluster (see Figure 2).

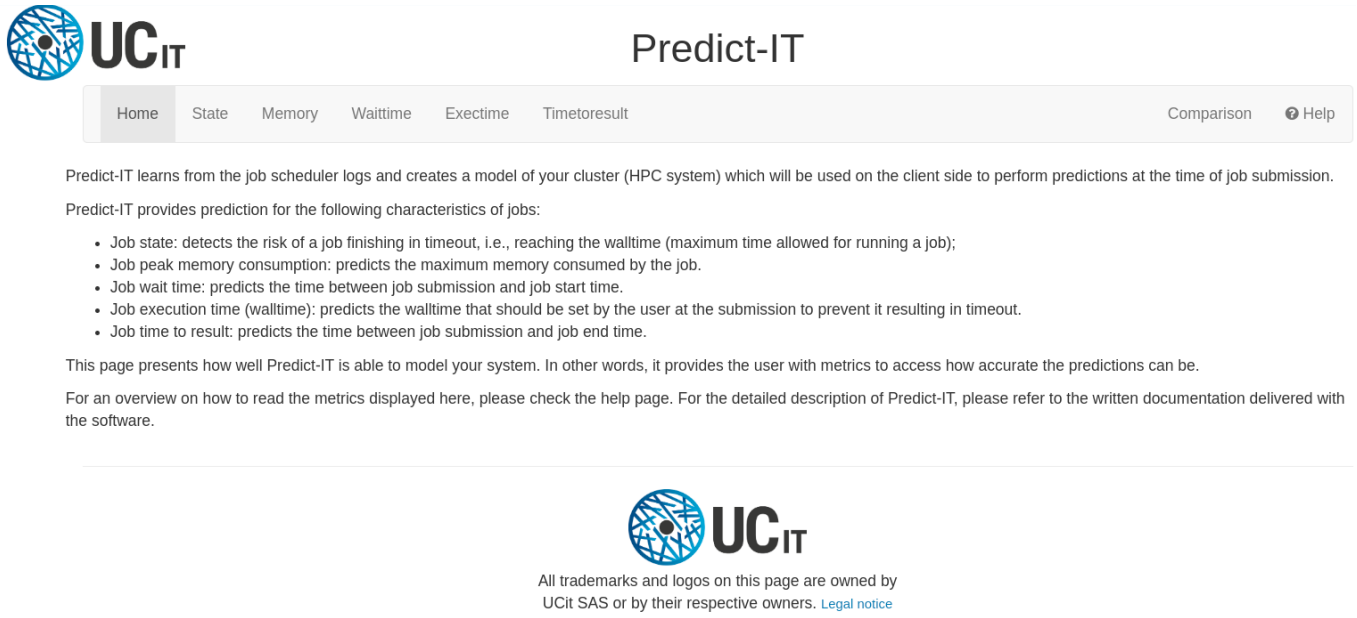

Figure 2. Metrics web page

CC BY-NC-ND UCit 2020

4. There are different metrics for each target of interest (click on the links "State", “Memory”, “Waittime”, "Exectime", or "Timetoresult"). Among them is the accuracy level that one should expect when requesting predictions for submitted jobs. See **Erreur ! Source du renvoi introuvable.** for an example of the State page.

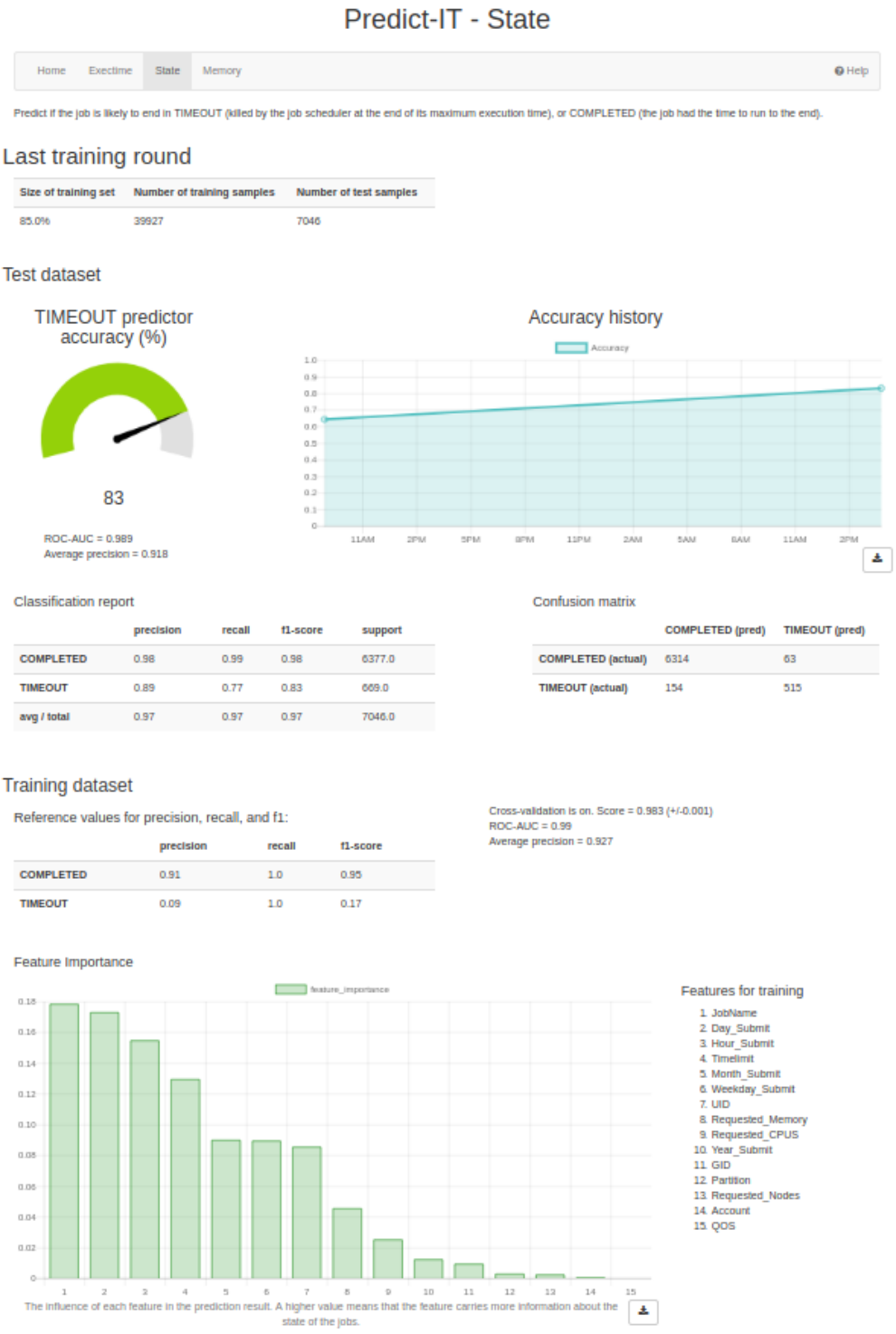

Figure 3. State metrics page example

CC BY-NC-ND UCit 2020

For a complete description of the metrics page, including the meaning of each metric and how to interpret it, please refer to the help page accessible via the link "Help" on the upper right corner of the metric pages.

7.2 - Client side

After the generation of a cluster model on the server side, the user can start requesting predictions for the state, memory, waiting time, runtime (i.e. walltime), and time to result of his jobs.

7.2.1 - Command line client

The client comes with its own online help. To display it, just type `./client -h`

```
Predict-IT client

positional arguments:
  Command         Select the command to run
    state         Predict job end state (Timeout or Completed)
    memory        Predict job peak memory consumption
    waittime      Predict job wait time
    timetoresult  Predict job time to result
    version       Retrieve server version

optional arguments:
  -h, --help      Show this help message and exit
  -v, --version   Print version and exit
```

As aforementioned, there are currently five types of predictions (targets):

- state, which predicts the job final state (TIMEOUT or COMPLETED)
- memory, which predicts the peak memory consumption
- waitime, which predicts the time between job submission and start
- runtime, which predicts the maximum execution time (i.e. walltime) for the job
- timetoresult, which predicts the time between job submission and the end of its execution

CC BY-NC-ND UCit 2020

Each target has its own help menu accessible by typing `./client <target> -h`. For example, for target Memory:

```
usage: prediction.py memory [-h] [-di | -i json | -f inputFile]
                            [-j {slurm,torque,oar,swf,pickleFile}]
                            [-s %Y-%m-%d %H:%M:%S] [-e %Y-%m-%d %H:%M:%S]
                            [-op OUTPUTPATH] [-jp JSPARAMS]

optional arguments:
  -h, --help            show this help message and exit
  -di, --describe       Describe service
  -i json, --input json JSON input string
  -f inputFile, --file inputFile
                        Input file (JSON file by default, or job scheduler
                        logs if used in conjunction with the -j option)
  -j {slurm,torque,oar,swf,pickleFile}, --jobscheduler {slurm,torque,oar,swf
                        ,pickleFile}
                        Job scheduler parser to use for the input file (-f
                        option)
  -s %Y-%m-%d %H:%M:%S, --starttime %Y-%m-%d %H:%M:%S
                        Only keep jobs that where submitted after this date
  -e %Y-%m-%d %H:%M:%S, --endtime %Y-%m-%d %H:%M:%S
                        Only keep jobs that where submitted before this date
  -op OUTPUTPATH, --outputpath OUTPUTPATH
                        Path and name to save the output files in JSON
  -jp JSPARAMS, --jsparams JSPARAMS
                        Job scheduler additional parameters
                        (key=value,key=value...).See documentation for the
                        list of supported parameters.
  -u URL, --url URL     URL to replace server baseURL
```

By default, if you call the client for any target without any other argument, you will enter an interactive mode. In this mode, the client first queries the server for the relevant fields you have to specify, and then asks you to fill them in.
Note: All fields are optional, the server sets default values for each field if not set manually by the user.

The interactive client also automatically fills in some of the fields:

- your user ID (UID, taken from the current user)
- your group ID (GID, taken from the current user)
- the submission time (supposed to be “now”)

CC BY-NC-ND UCit 2020

Here is an example for state prediction:

```
Please provide the following parameter specific to the job you want to predict
(all parameters are optional):
Account (string): lab1
JobName (string): ASAP
Partition (string):
QOS (string):
Requested_CPUS (integer, number): 2
Requested_Memory (integer, bytes): 62914560000
Timelimit (integer, seconds): 3600

Predictions for jobs' state:
+ Job 1 (ASAP) predicted state is COMPLETED (confidence 100%)
    Job details: {'Account': 'lab1', 'GID': '1000', 'JobName': 'ASAP',
'Requested_CPUS': '2', 'Requested_Memory': '62914560000', 'Timelimit': '3600',
'UID': '1000', 'confidence': '1.000', 'observed': 'None', 'prediction':
'COMPLETED'}
```

Note that in the interactive mode you can only request the prediction for a single job, while with the `-i` and `-f` arguments you could query for multiple jobs: if you do not use the interactive mode, they should point respectively to the JSON input (provided directly in the command line) or the JSON file (which stores the JSON command).

CC BY-NC-ND UCit 2020

Whether via command line or file, the JSON input should contain the fields (or a subset of them) described when typing `./client <target> -di`.
Example for state:

```
Description of state prediction:
The goal of state prediction is to identify jobs with high probability of
finishing in TIMEOUT (killed by the job scheduler at the end of its maximum
execution time). Once identified, the user should take actions to avoid the
job termination (usually, the solution is to increase the WALLTIME parameter
at submission time).
The state metrics show how well Predict-IT can catch potential TIMEOUT (or
COMPLETED) jobs.
Such task involves analyzing the cluster data in order to identify what
differentiates a TIMEOUT from a COMPLETED job (the job had the time to run to
the end). Ideally, we want to catch all the TIMEOUT jobs without mistakenly
putting a COMPLETED job in the same basket.
This is the list of columns you can/should have in your input data:
- Account
- Allocated_CPUS
- Day_Submit
- GID
- Hour_Submit
- JobName
- Month_Submit
- Partition
- QOS
- Requested_CPUS
- Requested_Memory
- TimeLimit
- UID
- Weekday_Submit
- Year_Submit
```


CC BY-NC-ND UCit 2020

Here is an example of a JSON input describing the submission values for two jobs:

```
{
    "jobInput": {
        "Account": [
            "default",
            "default"
        ],
        "GID": [
            "1200",
            "550"
        ],
        "JobName": [
            "rae_cruise",
            "acwssi1"
        ],
        "Partition": [
            "debug",
            "prod"
        ],
        "QOS": [
            "1",
            "1"
        ],
        "ReqCPUS": [
            "24",
            "120"
        ],
        "Submit": [
            "2018-08-28 17:15:59",
            "2015-08-28 17:15:24"
        ],
        "Timelimit": [
            "7200",
            "21600"
        ],
        "UID": [
            "1201",
            "550"
        ]
    }
}
```

The snippet above can be read like so: taking for instance the first job ("`rae_cruise`"), it was submitted by UID (user ID) 1201, who belongs to GID (group ID) 1200, and requests 1 CPU, with a time limit (walltime) of 7200 seconds (2 hours).

The JSON above is stored in a file which will be used as the input to the prediction client (for the purposes of this documentation, let's call it `client.inputfile.json`). Two examples of predictions are provided below, one for STATE and the one for WALLTIME. The prediction for both targets is requested to the server via the following command:

```
./client <target> -f  client.inputfile.json
```

Calling that command with target 'state':

```
./client state -f  client.inputfile.json
```


CC BY-NC-ND UCit 2020

The command returns the following results:

```
Predictions for jobs' state:
+ Job 1 (rae_cruise) predicted state is TIMEOUT (confidence 64.2%)
Job details: {'Account': 'default', 'GID': '1200', 'JobName': 'rae_cruise',
'Partition': 'debug', 'QOS': '1', 'ReqCPUS': '24', 'Submit': '2018-08-28 17:15:59',
'Timelimit': '7200', 'UID': '1201', 'confidence': '0.642', 'observed': 'None',
'prediction': 'TIMEOUT'}
+ Job 2 (acwssi1) predicted state is TIMEOUT (confidence 71.2%)
Job details: {'Account': 'default', 'GID': '550', 'JobName': 'acwssi1', 'Partition':
'prod', 'QOS': '1', 'ReqCPUS': '120', 'Submit': '2018-08-28 17:15:24', 'Timelimit':
'21600', 'UID': '550', 'confidence': '0.712', 'observed': 'None', 'prediction':
'TIMEOUT'}
```

The first set of parameters (a dictionary) depicts the values for the first job in the JSON snippet. Similarly, the parameters for the second job are shown in the second dictionary. Notice that both dictionaries have a parameter 'observed' that will be filled if the observed value of the predicted target is provided in the input file.

More importantly, notice the last parameter called `'prediction'`: *this shows the predicted state for the job if those parameters are selected*. For both jobs it is predicted that it will result in TIMEOUT.

This result is also reported on the server side by only displaying the amount of jobs in each prediction category:

```
127.0.0.1 - - [02/Feb/2018 22:29:34] "POST /1.0/state HTTP/1.1" 200 -
2018-02-02 22:30:55,592 : INFO : prediction : licenseVerifier.checkLicense : License
is valid
2018-02-02 22:30:55,705 : INFO : prediction : stateResource._formatPrediction :
Prediction stats:
- 2/2 jobs have a TIMEOUT status
- 0/2 jobs have a COMPLETED status
```

Knowing that the jobs will probably result in TIMEOUT is valuable information to the user: changing the parameters here is a way to find the appropriate set of parameters which will provide the related job with a high probability of ending in COMPLETED state when really submitted on the cluster.

CC BY-NC-ND UCit 2020

For instance, for the first job a possible action would be to increase the "TimeLimit" (WALLTIME) value. For the purposes of demonstration, if we increase "TimeLimit" from 7200.0 to 72000.0 (ten times more), Predict-IT forecasts that the first job will now be COMPLETED:

```
Predictions for jobs' state:
+ Job 1 (rae_cruise) predicted state is COMPLETED (confidence 69.3%)
Job details: {'Account': 'default', 'GID': '1200', 'JobName': 'rae_cruise',
'Partition': 'debug', 'QOS': '1', 'ReqCPUS': '24', 'Submit': '2018-08-28 17:15:59',
'Timelimit': '72000', 'UID': '1201', 'confidence': '0.693', 'observed': 'None',
'prediction': 'COMPLETED'}
+ Job 2 (acwssi1) predicted state is TIMEOUT (confidence 71.2%)
Job details: {'Account': 'default', 'GID': '550', 'JobName': 'acwssi1', 'Partition':
'prod', 'QOS': '1', 'ReqCPUS': '120', 'Submit': '2018-08-28 17:15:24', 'Timelimit':
'21600', 'UID': '550', 'confidence': '0.712', 'observed': 'None', 'prediction':
'TIMEOUT'}
```

Now, we request a prediction of the maximum execution time (runtime/walltime) using the same input file with the increased "TimeLimit" (from 7200.0 to 72000.0) for the first job:

```
./client runtime -f  client.inputfile.json
```

The results are:

```
Predictions for jobs' maximum runtime:
+ Job 1 (rae_cruise) predicted maximum runtime is 03:00:00 (confidence 46.2%)
Job details: {'Account': 'default', 'GID': '1200', 'JobName': 'rae_cruise',
'Partition': 'debug', 'QOS': '1', 'ReqCPUS': '24', 'Submit': '2018-08-28 17:15:59',
'Timelimit': '72000', 'UID': '1201', 'confidence': '0.462', 'observed': 'None',
'prediction': '03:00:00'}
+ Job 2 (acwssi1) predicted maximum runtime is 00:01:00 (confidence 33.2%)
Job details: {'Account': 'default', 'GID': '550', 'JobName': 'acwssi1', 'Partition':
'prod', 'QOS': '1', 'ReqCPUS': '120', 'Submit': '2018-08-28 17:15:24', 'Timelimit':
'21600', 'UID': '550', 'confidence': '0.332', 'observed': 'None', 'prediction':
'00:01:00'}
```

This time, '*prediction' provides us with a forecast of the* WALLTIME *for each of the jobs*. Here we are especially interested in the first job since we know it was predicted to be COMPLETED. This way, it makes sense to focus only on it: we see that it was forecast to be completely executed within 3 hours.

Additional arguments can be passed to the job scheduler using –jp or --jsparams:

- "keep_all_states=True": keep PENDING jobs for the prediction. Default is False and only ended jobs are kept for running predictions.

Example calling the client with this additional argument:

```
./client state -f client.inputfile.json -jp keep_all_states=True
```


CC BY-NC-ND UCit 2020

7.2.2 - REST API

The server can be queried through its REST APIs, accessible on its listening IP and port (e.g., http://localhost:9999/).

A - Server version

`GET srv-version` or `version`

Returns the current version of the server, for example:

```
$ curl -s -X GET "http://predictitserver:9999/version" | python -mjson.tool
{
      "version": "1.1"
}
```

All other methods must be queried with the server version prepended to the name of the prediction, i.e., http://HOSTNAME:PORT/VERSION/PREDICTION (with PREDICTION being either `runtime` or `state`).

CC BY-NC-ND UCit 2020

B - Get prediction description

`GET` *`VERSION/PREDICTION`*

Returns a description of the prediction.

Example for `state`:

```
$ curl -s -X GET "http://predictitserver:9999/1.1/state" | python -mjson.tool
{
  "description": "The goal of state prediction is to identify jobs with high probability
of finishing in TIMEOUT (killed by the job scheduler at the end of its maximum execution
time). Once identified, the user should take actions to avoid the job termination
(usually, the solution is to increase the WALLTIME parameter at submission time). The
state metrics show how well Predict-IT can catch potential TIMEOUT (or COMPLETED) jobs.
Such task involves analyzing the cluster data in order to identify what differentiates
a TIMEOUT from a COMPLETED job (the job had the time to run to the end). Ideally, we
want to catch all the TIMEOUT jobs without mistakenly putting a COMPLETED job in the
same basket.",
  "features": [
     "Account",
     "JobName",
     "Requested_CPUS",
     "Requested_Memory",
     "QOS",
     "Allocated_CPUS",
     "UID",
     "Partition",
     "TimeLimit",
     "Year_Submit",
     "Weekday_Submit",
     "Month_Submit",
     "Hour_Submit",
     "Day_Submit",
     "GID"
   ],
   "url": "state"
}
```

The `description` field is a human-readable description of what is predicted.
The list of `features` is the name of the parameters you can provide to the prediction algorithm as input to query a prediction. None of them are mandatory, but the more you provide, the better the prediction is likely to be. The `url` field reminds the URL used to query the prediction.

CC BY-NC-ND UCit 2020

C - Request a prediction

`POST` *`VERSION/PREDICTION`*

With `PREDICTION` one of `state, memory, waittime, exectime or timetoresult.`

Data: a JSON structure containing the parameters related to the job submission. All parameters must be grouped inside the `jobInput` parameter. Then each parameter is a list, each element of this list referring to 1 job. Example:

```
{"jobInput":{"GID":["3000","3000"],"UID":["3177","3177"],"Requested_CPUS":["1
024","128"],"Requested_Memory":["10485760000","104857600"],"Timelimit":["1800
0","7200"]}}
```

Returns a JSON structure containing the list of all requested jobs with the associated prediction in the `prediction` parameter. Each job has its own `prediction` parameter with the prediction value. (Note that the structure of the output is a bit different than from the input).

```
$ curl -s -X POST -d
'{"jobInput":{"GID":["3000","3000"],"UID":["3177","3177"],"Requested_CPUS":["1024","1
28"],"Requested_Memory":["10485760000","104857600"],"Timelimit":["18000","7200"]}}'
"http://predictitserver:9999/1.1/exectime"
{
  "prediction": [
    {
      "GID": "3000",
      "Requested_CPUS": "1024",
      "Requested_Memory": "10485760000",
      "Timelimit": "18000",
      "UID": "3177",
      "confidence": 0.739,
      "prediction": "04:00:00"
    },
      "GID": "3000",
      "Requested_CPUS": "128",
      "Requested_Memory": "104857600",
      "Timelimit": "7200",
      "UID": "3177",
      "confidence": 0.629,
      "prediction": "03:00:00"
    }
  ]
}
```


CC BY-NC-ND UCit 2020

D – Errors

Here are some of the errors you can get, and the associated explanation.

If the URL does not exist:

```
{
    "error": "Not found"
}
```

If the server has not finished preparing its prediction algorithms:

```
{
    "action": "Retry later",
    "message": "No prediction algorithm available",
    "status": 400,
    "sub_code": 1
}
```

If there are no input parameters:

```
$ curl -s -X POST -H 'Content-Type: application/json, accept: application/json' -d
'{}' "http://localhost:9999/1.1/exectime | python -mjson.tool
{
    "action": "Update your input parameters",
    "message": "Needed input parameters: data",
    "status": 400,
    "sub_code": 2
}
```

If the input parameters have the wrong format:

```
$ curl -s -X POST -H 'Content-Type: application/json, accept: application/json' -d
'{"jobInput":{"UID":"12"}}' "http://localhost:9999/1.1/exectime | python -mjson.tool
{
    "action": "Check your input data (If using all scalar values, you must pass an
index)",
    "message": "Problem while parsing your input data",
    "status": 500,
    "sub_code": 1
}
```


CC BY-NC-ND UCit 2020

7.3 - Tracking script

The tracking script tracks all the jobs in a specific time window and requests predictions to the server for the trained targets. The time window is defined from X hours before the running time to the running time (when the tracking script is executed). The user can specify the lower boundary of the time window (in hours) as an argument of the tracking script. If no argument is given, 23 hours is taken as the default value.

7.3.1 - How to run

The tracking script can be run:

1. Either manually by running the `tracking-script` binary
   `./bin/tracking-script`
2. Or by using cron to schedule execution time (`crontab -e` to edit or modify a crontab entry). In the example below, the tracking script will run every day at 23:30 on a time window going from 8:30 (23:30 – 15 hours) to 23:30.

```
30 23 * * * /opt/pituser/predictit/bin/tracking-script -t 15
```

The tracking script comes with its own help. To display it just type:
`./bin/tracking-script -h`

```
Help documentation for tracking script.

Basic usage: ./tracking-script

Command line switches are optional. The following switches are recognized
-t  --Sets the hours before now (default: 23) to define the time window to track
      input jobs. For example if 20 is given, the time window will be 20 hours
      ago from now.
-c  --Sets the path to a server.conf file.
      Useful when needed to request predictions to multiple Predict-IT servers.
-h  --Displays this help message.
```

7.3.2 - How it works

When executed, the tracking script works as follows:

1. It extracts logs from the job scheduler (Slurm only) for jobs in the specified time window. The jobs are separated in two categories depending on their states: ended (cancelled, completed, failed, node_fail, preempted, timeout, boot_fail, deadline) or pending (jobs still queued). For jobs in ended category the job scheduler tracks jobs achieving the ended state within the given time window. For pending jobs, it tracks pending jobs submitted within the given time window.
2. Predictions for the state, memory, wait time, maximum execution time and time to result are then requested through the Predict-IT client for the tracked jobs.
3. Output predictions are formatted in JSON as follows:

```
{{ "JobID": "79", "prediction": "COMPLETED", observed: "COMPLETED", "confidence": 0.865
}, …, {"JobID": "2530", "prediction": "COMPLETED", observed: "COMPLETED", "confidence":
0.529 }}
```


CC BY-NC-ND UCit 2020

This formatting needs `jq` to manipulate JSON files (cf. 4.2 - Requirements - system).

4. Predictions are saved in JSON into `./data/dirName/tracking_script` folder. Each time the tracking script runs, ten output files are created containing the predictions (one file per target (state, memory, wait time, maximum execution time and time to result) x category (ended or pending)). Names of the output files follow this nomenclature:
   "YYYY-MM-DD_hh-mm-ss_target_CATEGORY.json"

For example:

```
2019-11-20_22-06-09_memory_ENDED.json
2019-11-20_22-06-09_memory_PENDING.json
2019-11-20_22-06-09_runtime_ENDED.json
2019-11-20_22-06-09_runtime_PENDING.json
```

N.B.: To request predictions successfully for the tracked jobs, Predict-IT server must be up.

CC BY-NC-ND UCit 2020

7.3.3 - The user interface to access accuracy metrics

Once the server has returned predictions for the tracked jobs, accuracy metrics are computed for the ended jobs (*i.e.* for the jobs having an observed value for the targets). Access `http://host:port/tracking` (in case you are using the default values, `host=127.0.0.1` and `port=9999`) from your browser to see the metrics. It can take a few minutes to display the metrics depending on the time window used to request predictions, but a caching system allows to speed the loading if you access this page later on the same day. See Figure 4 for an example of the tracking metrics page.

Figure 4. Tracking metrics page example

CC BY-NC-ND UCit 2020

This page is composed of 2 main parts:

- Global: shows the accuracy history for all targets as a function of prediction dates. A slider allows to filter the jobs by selecting the minimum confidence level associated to their predictions
- Per metrics: shows the confusion matrix and classification report for each target. These metrics are calculated for each prediction date (choose the date using the dropdown menu) depending on the selected minimum confidence level.

7.3.4 - Request predictions for a specific job

You can request predictions for a job using its ID.
Example for job 7305291_1: `http://host:port/tracking/job?id=7305291_1`. Figure 5 shows the output with the requested predictions. The job must be tracked and predicted by the tracking script during the tracking time windows in order to request its predictions.

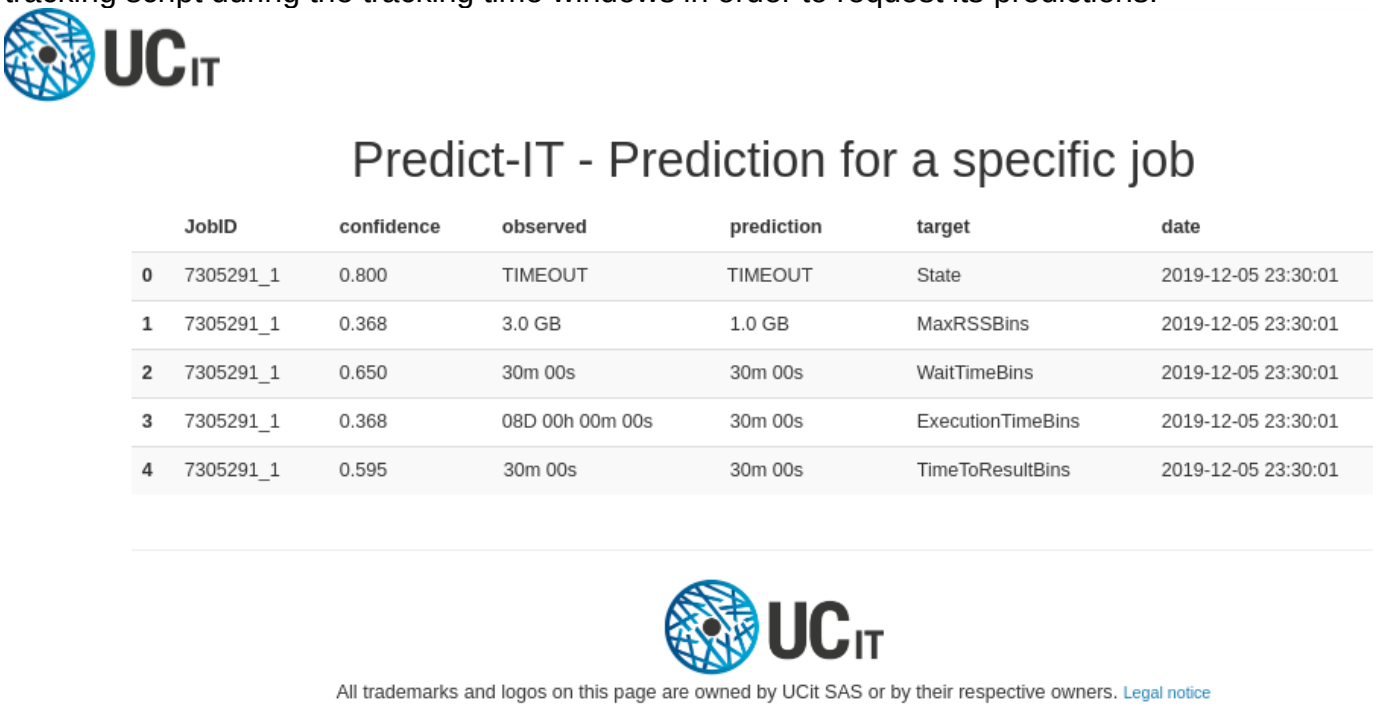

Figure 5. Predictions for a specific tracked job example

Output predictions can also be returned as json using the parameter '`json=true`'. Example: http://host:port/tracking/job?id=7305291_1&json=true.

CC BY-NC-ND UCit 2020

7.4 - Run multiple Predict-IT servers

It is possible to run multiple Predict-IT servers to test for different configurations. First you have to edit multiple server.conf files with your different configurations. Then run the servers manually by specifying the server.conf path to use using the “-c” argument.

```
./bin/predictit-server -c path/to/server.conf1
./bin/predictit-server -c path/to/server.conf2
```

Each command has to be executed in a separated environment/screen.

In the server.conf files, different “port” and “dirName” are mandatory to successfully run multiple servers in parallel. You can then access `http://host:port/metrics` by updating the host and port set in configuration files to access the interface for the desired server.

When requesting predictions from client side, you can specify the URL of the requested server using the “-u” argument. Host and port have to be updated depending on the server configuration files.

```
./bin/predictit-client state -u http://host:port -j slurm -f
path/to/file
```

As for the tracking script, “-c” argument allows to set the path to the concerned server.conf file. You have to run multiple cron with the different paths to request predictions from the different running servers.

```
30 23 * * * /opt/pituser/predictit/bin/tracking-script -t 15 -c path/to/server.conf1
30 23 * * * /opt/pituser/predictit/bin/tracking-script -t 15 -c path/to/server.conf2
```

7.5 - Data enhancers

Feature engineering is possible in Predict-IT through the use of data enhancers. This way, you can test multiple features to improve the model predictions.

Data Enhancers must be implemented in either csv (comma separated value) or python files.

1. CSV (.csv) files: These are the simplest and safest enhancers. Each csv file can add multiple features (columns) to the data processed, each csv file must contain:
    a. on the first line the name of the columns: the first one is the pivot column (and must exist in the input data), the other ones are the names of the new columns that will be created (a prefix is added to all the column names specified in the csv file).
    b. following lines must contain the values: first one is the pivot value, next ones are the newly created values
    c. format of the csv file is "free" for as long as Pandas is capable of detecting the format (you can use comas, semi-colons... as separators)
2. python (.py) files: These Data Enhancers allow more complex and dynamic data transformations, as it can either add new columns, or modify values in existing ones. All new column names must be prefixed. Python Data Enhancers receive a Python DataFrame. The class implemented in the Python file must respect the following conventions:

CC BY-NC-ND UCit 2020

a. the name of the class MUST be the same as the name of the file without the .py extension, and with an upper case first letter. For example, if your file is myDataEnhancer.py, then the class must be MyDataEnhancer.
b. the _init_ method receives a single argument: prefix, the prefix that needs to/can be prepended to any new column name.
c. a transform(self, dataframe, **kwargs) method MUST be implemented. The modifications on the DataFrame must be made in place. The list of created columns is expected as a return value.
d. a fillna(self, dataframe) method CAN be implemented to fill missing values. The method must work in-place. If not provided, a generic fillna method is used.

Examples of Data Enhancers are presented in annex at the end of this document.

Data Enhancers files must be located in “conf/enhancers” directory of Predict-IT. You can then create one or multiple symbolic links from each target subdirectory (e.g. from “conf/enhancers/State”) pointing towards the Data Enhancer files to use for this target. This way it is possible to apply different Data Enhancers depending on the target. A parameter in `server.conf` allows to apply or not the Data Enhancers (`add_all_enhanced_cols`).

CC BY-NC-ND UCit 2020

8 – Administration

Here we describe all the parameters in the configuration files `server.conf` and `client.conf`. Those can be customized depending on the user dataset and what target is supposed to be predicted.

8.1 - Environment Variables

Configuration files location can be configured through environment variables. It is a good practice to preserve the original (default) server and client configuration files the way it is. If it is necessary to modify any parameter, we recommend creating an editable copy and set the system variable to let Predict-IT know where it is:

`PIT_SERVER_CONF`. specifies the location of the server configuration file.
Example: `export PIT_SERVER_CONF=/home/usename/server.conf`

`PIT_CLIENT_CONF`: specifies the location of the client configuration file.
Example: `export PIT_CLIENT_CONF=/home/usename/client.conf`

8.2 - Extract job scheduler logs

If you do not want Predict-IT to automatically and periodically extract new data from the job scheduler, you can use the scripts provided in `<INSTALLATION_PATH>/bin/extractData`. The first argument is the name of the job scheduler, subsequent arguments may vary depending on the job scheduler (type `<INSTALLATION_PATH>/bin/extractData JOBSCHED -h`, with `JOBSCHED`: `pbs`, `torque`, `slurm` or `sge`). The data extraction relies on different methods depending on the job scheduler:

- SLURM™: relies only on `sacct` and `scontrol`.
- Torque and Altair® PBS Professional®: extracts raw data from accounting files, and from `pbsnodes` and `qstat`.
- Grid Engine: extracts raw data from accounting file

Each script creates 3 text files in the current directory:

- **historical data on jobs**: `HOSTNAME_DATE.jobs`
  - From a given date up to now
  - Retrieve all data gathered by the job scheduler (job id, requested/obtained resources, node list, submit/start/run times…)
  - For confidentiality reasons, the output is anonymized: by default, usernames and groups are not retrieved, only UID and GID are
- current **list of nodes** and their description: `HOSTNAME_DATE.nodes`
  - Node name
  - Number of cores, number of cores per socket, memory
  - Specific resources (features described in the job scheduler)
- current **list of partitions/queues** and their description: `HOSTNAME_DATE.partitions`

The execution of the script is very lightweight. No processing of the data is done.
Each script comes with its own online help, run `./extractXXXXData.sh -h` to see this help.

For Predict-IT, only the first file is currently relevant: `HOSTNAME_DATE.jobs`, you need to specify the path to this file in the `server.conf` with the `inputFile` in the `jobScheduler` section.

CC BY-NC-ND UCit 2020

8.3 - Server configuration file

The `server.conf` file is organized in the following sections.

Section	Parameters
`[cluster]`	**Optional total number of CPUs available on the cluster (default None). If not specified, totCPU will be requested to the job scheduler (SLURM only).** totCPU=None
`[server]` General parameters	**host: hostname or IP address to listen on (default is 127.0.0.1). Use 0.0.0.0 to listen on all interfaces** `host = 127.0.0.1` **port: listening port (default is 9999)** `port = 9999` **Name of the output directory to save data for this configuration file (default "default").** This is mandatory when running PIT with multiple configuration files. `dirName = default` **Optional path to license file. If not specified, the license file will be searched in license/license.lic**. `licenseFile = PATH_TO_FILE` **Optional log Level (default is INFO/20)** *Level Numeric value* *CRITICAL 50* *ERROR 40* *WARNING 30* *INFO 20* *DEBUG 10* *NOTSET 0* `logLevel = 10` **Separate logs for the different targets? (optional, default False)** *If true, then for each target, a specific file will be created for its logs.* *The files will be named with <logFile>.<target>* `separateLogs=False` **Select the target(s) for the prediction (optional)** *Currently supported targets: State, ExecutionTimeBins, MaxRSSBins, WaitTimeBins, TimeToResultBins (default)* `target=` `State,ExecutionTimeBins,MaxRSSBins,WaitTimeBins,TimeToResultBins`

CC BY-NC-ND UCit 2020

	Select the confidence level used to assess prediction on jobs (optional, default 0.8) `comp_conf_thres=0.8` **Save test values and metrics? (optional, default None)** *- single_files: 2 files per target. The files will be named with <TARGET>_test_values.csv & <TARGET>_test_metrics.csv* *- separated_files: 2 files per target, per training. The files will be named with <datetime>_<TARGET>_test_values.csv & <datetime>_<TARGET>_test_metrics.csv* `Save_test_assess=None` **Filter jobs on feature values? (optional, default None)** *Format is the following: columnName<FILTER>value or columnName<FILTER>value1,value2,....* *Multiple filters can be provided by separating them with ';'.* *FILTER can be: ==, !=, <=, >=, <, >, ~= (regex).* `filterColumn=GID==500` **Save best model and compare it to the new trained model for each training? (optional, default False)** `save_compare_model=True` **Compute additional features? (optional, default False)** *If True, then apply all data enhancers found in "conf/enhancers" directory.* `add_all_enhanced_cols=False`
`[tracking_script]`	**Compute metrics on predictions for tracked jobs? (optional, default is False)** `trackingMetrics=False` **Optional time delta in days to allow user to visualize tracking metrics:** **- from starttime to starttime + timeDelta (if endtime unset and starttime set)** **- from endtime - timeDelta to endtime (if endtime set and starttime unset)** **- from now - timeDelta to now (if endtime unset and starttime unset)** **In any case, if both starttime and endtime are set we'll retrieve predictions inbetween those two dates.** `timeDelta=5`
`[jobScheduler]` Reading job-scheduler logs	**Which job scheduler should we use? (this parameter is mandatory)** *- slurm* *- torque* *- pbs* *- swf (not a real job scheduler, just reads an swf file)*

CC BY-NC-ND UCit 2020

	- pickleFile (not a real job scheduler, just reads a pickle file) `driver=slurm` **Optional argument to load data from a file instead of from really calling the job scheduler** *Multiple files can be provided separated by commas.* `inputFile=PATH_TO_FILE,PATH_TO_FILE` **Optional Start time to get the historical data after (YYYY-MM-DD HH:MM:SS)** `starttime=2015-01-01 00:00:00` **Optional End time to get the historical data before (YYYY-MM-DD HH:MM:SS)** `endtime=2017-12-31 07:51:34` **Optional time delta in days to allow user to retrieve data:** **- from starttime to starttime + timeDelta (if endtime unset and starttime set)** **- from endtime - timeDelta to endtime (if endtime set and starttime unset)** **- from now - timeDelta to now (if endtime unset and starttime unset)** **In any case, if both starttime and endtime are set we'll retrieve data inbetween those two dates.** `timeDelta=60` **Optional duration in days to chunk the requested period when calling the jobscheduler:** *Use this to avoid timeout when requesting large amount of data.* `chunkDuration=7` **Check job scheduler (True/False) (default True)** *If you use an input file, you should set that to False* `check=false` **Additional parameters for the job scheduler plugin.** *All additional parameters must start with 'param_'.* ***+ OAR**:* - dbhost (mandatory): hostname of the PGSQL server hosting OAR database - username (mandatory): username to connect to the database - password (mandatory): password to connect to the database - database (mandatory): database name *+ **PBS**:* - *acct_dir: path to accounting directory (default="/var/spool/pbs/server_priv/accounting/")*

CC BY-NC-ND UCit 2020

	+ ***Slurm***: - *- noalloc: Gather details about each job step or not? If noalloc is True (default), then we gather the details for all steps of the jobs, it allows to get the MaxRSS values properly.* + ***SWF***: *None* + ***Torque***: - *- acct_dir: path to accounting directory (default=/var/spool/torque/server_priv/accounting/)* `param_acct_dir=/var/spool/torque/server_priv/accounting/` `param_separator=slurmdefault`
`[XXX_algorithms]` General parameters for the algorithms that predict `XXX` (with `XXX` being `ExecutionTimeBins`, `State`, or `MaxRSSBins`. See parameter `target` in section `server`)	**Comma separated name of algorithms that must be used.** Then, you must create one section in this configuration file for each algorithm Supported algorithms: randomForest, decisionTree, extraTrees, gradientBoosting `usedAlgorithms=randomForest` **Currently only the voting metaAlgorithm is supported. Do not change!** `metaAlgorithm=voting` **Partition type (optional)** *Options are chrono (to partition based on date) or size (based on size - default option). The former should be used with the split_date parameter (see below), and the latter with trainSize (see below).* `partition_type=size` **Split (optional)** *String representing the last chunk of data (chronologically) in the log which should be reserved for testing. You can provide it in terms of x months ("xm"), weeks ("xw"), days ("xd"), hours ("xH"), minutes ("xM"), or seconds ("xS"). For example, if you want to save the last month for testing (while using the rest of the data for training), use split="1m" (default is 1 month "1m").* `split=1m` **Train size should be a float value within (0, 1).** *For example, 0.999. Default = None* `trainSize=0.95` **Remove features that do not have a significant number of samples (default 0.5)** `significant_n_samples = 0.5`

CC BY-NC-ND UCit 2020

refreshModel= <second> <minute> <hour> <day> <month> <year>
The following table lists all the available expressions for use in the fields from year to second. Multiple expression can be given in a single field, separated by commas.
Expression Field Description
** any Fire on every value*
**/ any anyFire every a values, starting from the minimum*
a- any anyFire on any value within the a-b range (a must be smaller than b)
a-b/ any anyFire every c values within the a-b range
x,y,z any Fire on any matching expression; can combine any number of any of the above expressions""""
example to refresh every 5 hours:
`refreshModel=0 0 */5 * * *`

trainOnce (True,False) (default False)
train the model only once, and then pauses the training task
`trainOnce=True`

Switch on/off (True/False) data scaling (normalizing) (default False)
`scale=False`

Switch on/off (True/False) data Balancing (default False)
`balance=False`

Switch on/off (True/False) automatic computation of 20 balanced bins (default False)
This does not apply to State target.
`balanced_bins =False`

Parameters for cross validation.
cv={True,False}: to switch on/off cross validation. (default True)
Turning it off might be useful to reduce overall calculation time.
fold_strategy={kfold,stratified} (default kfold)
scoring_multiclass={f1_micro, f1_macro} (default f1_micro)
n_splits: integer, number of folds for kfolds (default 3)
```
cv=True
fold_strategy=kfold
scoring_multiclass=f1_micro
n_splits=4
```

Select features to use as explanatory variables (default is all features).
Currently supported features: Requested_CPUS, Requested_Memory_Per_CPU, Requested_Nodes, UID, GID, Hour_Submit, Weekday_Submit, Day_Submit,

CC BY-NC-ND UCit 2020

	Month_Submit, Year_Submit, JobName, Partition, Account, QOS, Timelimit, Cluster_Load_Running , Cluster_Load_Running_Relative, Cluster_Load_Queued , Cluster_Load_Queued_Relative, Number_Queued_Jobs, Number_Queued_Jobs_Relative + features added using data enhancers. `feature_list=Requested_CPUS,Requested_Memory_Per_CPU,Requested_Nodes` **Choose bins to categorize the target.** *Bins format varies depending on the target (time-related target or MaxRSS). This does not apply to State target.* `Bins_list=60s,10m 00s,30m 00s,01h 00m 00s,02h 30m 00s,03h 20m 30s,04h 00m 00s,07D 00h 00m 00s`
`[XXX_randomForest]` Specific parameters for the initialization of random forest algorithm for `XXX` predictions (with `XXX` being `ExecutionTimeBins`, `State`, or `MaxRSSBins`. See parameter `target` in section `server`)	**Parameters for the initialization of the algorithm.** *All these parameters must start with 'param_'.* *Parameters and default values:* *max_depth = None* *max_features=auto* *min_samples_split = 2* *random_state = 0* *n_estimators = 10* *criterion = gini* *class_weight=None* *param_oob_score=True*
`[voting]`	*Parameters for the initialization of the voting algorithm.* *All these parameters must start with 'param_'.* *Parameters and default values:* *voting_case={hard,soft} (default hard)* *weight_algorithms = None (array-like, shape = [n_classifiers]) (default None)* *n_jobs: integer, number of threads to use (-1 means no restrictions, use all cores) (default -1)*
Other sections: `[decisionTree]` `[extraTrees]` `[gradientBoosting]`	Templates for other algorithms. See `server.conf` for a complete list of parameters.

CC BY-NC-ND UCit 2020

8.4 - Client configuration file

The `client.conf` file contains only one section described below:

Section	Parameters
`[server]`	URL used by the client side to contact the server. *Format is* `http://IP_OR_HOSTNAME:PORT/` `baseURL=http://127.0.0.1:9999/` Warning: the IP address indicated in the baseURL variable must be the IP indicated in server.conf server-side.

CC BY-NC-ND UCit 2020

9 - Troubleshooting

Here we provide a list with known issues and the actions that you can take (if necessary) to circumvent them.

Value error
The following lines are printed on the terminal prompt during the execution of the tool:

```
ValueError: Buffer dtype mismatch, expected 'Python object' but got 'long'
Exception ignored in: 'pandas._libs.lib.is_bool_array'
ValueError: Buffer dtype mismatch, expected 'Python object' but got 'long'
ValueError: Buffer dtype mismatch, expected 'Python object' but got 'long'
Exception ignored in: 'pandas._libs.lib.is_bool_array'
ValueError: Buffer dtype mismatch, expected 'Python object' but got 'long'
```

Such “error” is actually a warning about the data type that is being used in the conversion of the log data (usually strings and numbers) into categorical data (used by the embedded algorithm that performs the prediction). This does not affect prediction performance and should be removed in a coming release.

Selection of parameter class_weight in `[XXX_randomForest]`
The parameter class_weight allows for equalizing the relevance of the classes. It is especially useful when in State prediction the number of TIMEOUT jobs is much lower than the number of COMPLETED, providing the user with another way of balancing the dataset. Four options are available for class_weight: 'None', 'balanced', 'balanced_subsample', and '{0: weight_class_0, 1: weight_class_1, ...}', where weight_class_j for j=0,1,… can have any value greater or equal to zero.

In State prediction, if the user chooses class_weight = {0: weight_class_0, 1: weight_class_1} while keeping cv=True (calculation of cross validation in [State_algorithms]), the execution breaks and no prediction model is generated. This way, if using a dictionary like the one above to define custom weights for each class, make sure to turn off the calculation of cross validation. This issue will be fixed in the next release.

Error when executing tracking script using cron
Empty output files when calling the tracking script using cron are due to missing environmental variables. This issue can be fixed by sourcing the profile directory before calling the tracking script in crontab file, e.g.:

```
30 23 * * * source /etc/profile;/predictit/bin/tracking-script
```


CC BY-NC-ND UCit 2020

Annex

Data Enhancer – CSV example

The following example creates two new columns (`FirstName` and `Name`) from the `UID` of the user.

```
UID, FirstName, Name
10001, Luke, Skywalker
10002, Anakin, Skywalker
0, Yoda, Yoda
22, R2, D2
```

Data Enhancer – Python example (testTransform.py)

The following example creates a new column (`_category`) and modifies the `Account` of the jobs.

```
class TestTransform():
    """ This is a simple data enhancer example for Predict-IT"""
    def __init__(self, prefix):
        self.new_cols = []  # List of columns added using this data enhancer
        self.prefix = prefix
        self.default_na_values = {}  # Store values used to replace NaN

    def transform(self, dataframe, **kwargs):
        # Modifications must be done in place.
        # You CAN  make a different treatment between training vs. Asking
        # for predictions.
        # For example, if you are computing a value during training that has
        # to be used when getting predictions.
        if kwargs["training"] is True:  # During training
            # Creation of a new column
            newcol = self.prefix + "category"
            dataframe[newcol] = dataframe["Allocated_CPUS"].apply(lambda x:
"mono" if x == 1 else "multi")
            self.new_cols.append(newcol)

            # Modification of an existing column
            dataframe["Account"] = "Grouped accounts"

        else:  # When asking for prediction
            newcol = self.new_cols[0]

            # Creation of a new column
            dataframe[newcol] = dataframe["Allocated_CPUS"].apply(lambda x:
"mono" if x == 1 else "multi")

            # Modification of an existing column
            dataframe["Account"] = "Grouped accounts"

        # Return the list of newly created columns
        return self.new_cols
```


CC BY-NC-ND UCit 2020

```
    def fillna(self, dataframe):
        """Fills na values. Modifies dataframe in place."""
        for col in self.new_cols:
            if col in dataframe.columns:
                if col not in self.default_na_values.keys():
                    self.default_na_values[col] = dataframe[col].mode()
                dataframe.loc[:, col].fillna(self.default_na_values[col],
inplace=True)
```


CC BY-NC-ND UCit 2020