e-HIGHWAY 2050

Modular Development Plan of the Pan-European Transmission System 2050

Contract number	308908	**Instrument**	Collaborative Project
Start date	1st of September 2012	**Duration**	40 months
WP 8	**Enhanced Pan-European Transmission Planning Methodology**		
D 8.6b	**Enhanced methodology for long-term grid planning: prototype**		

		Date & Visa
Written by	Jean Maeght, RTE	16/11/2015
Checked by	Patrick Panciatici, RTE	16/11/2015
Validated by	Gérald Sanchis, Nathalie Grisey, RTE	30/11/2015

Project co-funded by the European Commission within the Seventh Framework Programme		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Document information

General purpose

This document is the deliverable D8.6b of the e-Highway2050 project. It describes the global prototype developed in WP8 to implement the methodology presented in D8.6a "Detailed enhanced methodology for long-term grid planning".

Change status

Revision	Date	Changes description	Authors
V1.0	16/11/2015	Initial version	Jean Maeght
V1.1	26/11/2015	Addition of TEPES section	Jean Maeght
V1.2	30/11/2015	Remarks from Comillas, Sintef, PSE	Jean Maeght

TABLE OF CONTENT

List of Figures

1. Introduction

This document intends to give a global description of the main prototype developed in WP8 to implement the Enhanced methodology for long-term grid planning. The methodology is described in deliverable D8.6a [10], whileD8.6a relies on documents D8.2a [2], D8.3a [4], D8.4a [6] and D8.5a [8], which describe the steps in Figure 1 below.

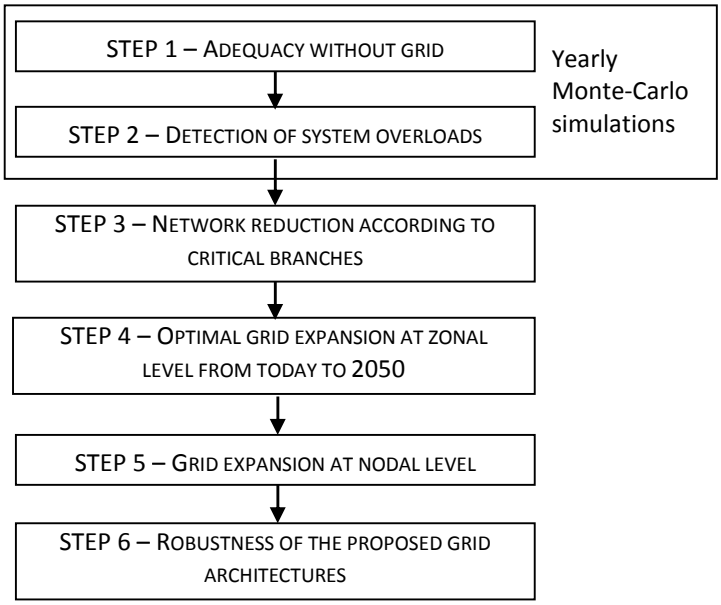

***Figure 1** – Methodology proposed in e-Highway2050, WP8*

Similarly to deliverable D8.6a [10], this document relies on prototype descriptions in deliverables D8.2b [3], D8.3b [5], D8.4b [7] and D8.5b [9].

The use of medium-sized cloud computing facilities has been investigated for the European test case presented in D8.6a [10]. This document will try to document the way in which parallel computing was used.

In former deliverables, modules generally have functional names; from a source-code point of view, scripts and code source files have other names, which are not always consistent with functional names or descriptions. Prototype is a prototype, so there was no necessity to force participants to use strict naming rules. In the next sections, the main titles are based on the Step1 – Step4 naming in Figure 1 above; references to former deliverables and functional information are also mentioned and names of source code files are also given in order to facilitate their use in the future.

In the next section, hardware architecture used for the test case is presented. In the following sections, steps 1 to 4 are presented, with explanations on the way they are parallelized.

Steps 5 and 6 are not presented in this document since they did not need to be integrated in the common platform, so we simply refer to former deliverables:

- Step 5 - Grid expansion at nodal level is undertaken using the software TEPES [20], presented in section 6 of Deliverable D8.3b [5],
- Step 6 – Robustness of the proposed grid architectures is presented in Deliverable D8.5b [9].

Although TEPES was not fully integrated in the common platform, the proof-of-concept of its integration on Linux systems is presented is section 6.

2. Hardware system used for the test case

High-Performance Computing (HPC) was used in the project iTesla, which applied to the Prace consortium to get access to the HPC system Curie with more than 10,000 cores. Running computation on such large computers is a real challenge. Moreover, these HPC systems are usually mostly used for fundamental physics simulation, and barely for applied research projects. So it was a real success for the iTesla project to be granted for an access to the Prace resource Curie, and to be able to actually run iTesla software on this HPC system.

In e-Highway2050 WP8, the target was less ambitious in terms of computing. We used cloud computing facilities, which means in our case renting dedicated servers from an external provider. This provider, OVH [15], is located in north of France; it operates several large data centres in France and Canada.

For the prototyping phase a server with 16 cores was rented. When addressing the European test case, we rented 4 additional servers to develop and test parallelisation tools. Once we were ready to run large computations, we rented 6 further servers.

For the main computations on the European test case, 9 servers were running in parallel. The 2 remaining ones were used by the development team to monitor computations, make intermediate result analyses, or to run small tests. To sum up, our main computation system was made of 9 servers, 16 cores each.

Detailed characteristics of each server “MG128”, as provided by OVH, is:

- Processor Intel Xeon E5 (2 processors E5-2630v3),
- 16 cores (8 cores per processor, 2 threads per core if using hyper threading, frequency varying from 2.4 to 3.2 GHz),
- 128 GB shared memory (DDR4 ECC 1866 MHz),
- 3 hard disks of 2 TB, used with RAID3 configuration: one disk is used for redundancy, so 4 TB are available for user,
- Operating system Linux Centos 6.6,
- Rental price is 300 Euros per month per server.

Although operating system installation and monitoring facilities are available through a web interface of OVH, we had to install software (FicoXpress, Ampl, python packages, Slurm, etc...) manually for each server.

Preliminary tests on hyper threading showed that for the computations we had to run, hyper threading was not efficient. So we decided to limit the number of parallel tasks to 16 on each server.

Looking at the European test case description in deliverable D8.6a [10], the reader will see 3 different scenarios and 3 time horizons. In most parts of the parallel computations, one couple (scenario, time horizon) will be run on each server (3 scenarios * 3 time horizons = 9 servers).

3. Step 1 and Step 2

From a computational point of view, Step 1 "Adequacy without grid" and Step 2 "Detection of system overloads" are integrated.

For each couple (scenario, time horizon), one server is used to run 100 Monte-Carlo simulations.

Main script (script `starter_mc_years.sh`) will run all computations for a given Monte-Carlo year:

- Step 1.1: Demand/Generation (script `getnewmc`). Method in section 3 of D8.2a [2], prototype in D8.2b [3].
- Step 1.2: Hydro scheduling (script `hydro_scheduling`). Method in section 4 of D8.2a [2], prototype in section 6 of D8.2b [3].
- For each of the 52 week, run successively:
    - Step 1.3: Computation of generation planning without grid (script `adequacy`), one optimisation problem per week. Method in section 5 of D8.2a [2], prototype in section 6 of D8.2b [3].
    - Step 2.1: Automatic mapping (script `disaggregation`). Method in section 3 of D8.3a [4], prototype in section 4 of D8.3b [5].
    - Step 2.2: DCOPF (script `wp8dcopf`), 7*24 hourly optimization problems. Method in section 4 of D8.3b [4], prototype in section 4 of D8.3b [5].
- Step 2.3: computation of overload indicators from DCOPF results (script `indicatorsCalculateYear`), see section 4.2 of D8.3b [5].

As we want to compute 100 Monte Carlo years, the task scheduler will run 16 instances of `starter_mc_years.sh` in parallel. Then it will run the next 16 years, etc... When all 100 Monte-Carlo years are finished, results are sent to the master server.

Each of the 9 server has to run computations for each couple (scenario, time horizon), with 100 Monte-Carlo years each. So 900 Monte-Carlo years are computed in total.

Global computation time is 100/16 rounded up, so 7, to be multiplied by 5 hours for 1 run of `starter_mc_years.sh`. See computation time table in end of D8.6a [10]. The number of Monte-Carlo years computed was for some case reduced to 96 instead of 100, because the 4 additional Monte-Carlo years caused an additional 5 hours computation time.

For each Monte-Carlo year, script `starter_mc_years.sh` generates 5.2 GB of data. For 100 Monte-Carlo years, this makes roughly 520 GB. For each couple (scenario, time horizon), these 520 GB are stored on the corresponding server. From these 520 GB data, step 2.3 (overload indicators computation) computes 6.5 MB of indicators per Monte-Carlo year, so 650 MB per server. These 650 MB are sent to the master server. Total amount of overload indicators received by master server is 650 MB * 9 = 6 GB.

4. Step 3: Network Reduction according to critical branches

The methodology for this step is described in section 5 of D8.3a [4]; the corresponding prototype is presented in section 5 of D8.3b [5].

This step is not parallelized and is run on one single core. It takes a few minutes to read the 6 GB of indicators and to reduce the 1000 nodes network to a 100-zone grid.

5. Step 4: Optimal grid expansion at zonal level from today to 2050

5.1. DCOPF

After reduction to a 100-zones grid, some indicators (e.g. nodal prices) need to be calculated. The same DCOPF as in step 2 is run, but on the 100 nodes of the zonal grid instead of one the 1000 nodes in the initial grid. The methodology is described in section 4 of D8.3a [4], prototype in section 4 of D8.3b [5].

For each couple (scenario, time horizon), these computations are independent, so they are run separately on the 9 servers.

For each couple (scenario, time horizon), the script `dcopf_reduced` is run on the corresponding server. On each of the 9 servers, 16 Monte-Carlo years of the zonal DC OPF are computed in parallel. When they are finished, the task scheduler will run the next 16 and so on

To summarize: as in steps 1 and 2, a total number of 900 Monte-Carlo years of DC OPF are computed, using 9*16 cores. The computation times are included at the end of deliverable D8.6a [10].

The results of these computations stay on each server: they will be used for the zonal-price-difference computation on the same server.

5.2. Zonal-price-difference calculation

For each couple (scenario, time horizon), the zonal price difference computations are independent, so they are run separately on the 9 servers.

This computation step has to compute zonal-price-differences, using the results of the zonal DC OPF. The corresponding script is `feature_construction.py`.

For each couple (scenario, time horizon), these computations for the 100 Monte-Carlo years are independent, so they are run in parallel on the 16 cores of the corresponding server. The results of the zonal-price-difference calculation stay on the corresponding server.

5.3. Snapshot selection

For each couple (scenario, time horizon), the snapshot selection computations are independent, so they are run separately on the 9 servers.

The script name is `snapshot_selection`. Its goal is to select a small number of snapshots (e.g. 5 snapshots) among the 100*52*7*24 snapshots of the 100 Monte-Carlo years.

Clustering methods are implemented in `snapshot_selection`. The method is presented in section 2 of deliverable D8.4a [6], with complementary discussion in section 5 of D8.6a [10]; the prototype is described in section 2 of deliverable D8.4b [7]. The described method is not parallelized. Thus, on each of the 9 servers, the snapshot selection is computed on 1 core for the corresponding couple (scenario, time horizon).

The results of this snapshot selection are sent to the master server.

5.4. Candidate selection

For each couple (scenario, time horizon), the candidate selection computations are independent, so they are run separately on the 9 servers.

The script name is `starter_candidate_selection.sh`. Its goal is to select candidates to be built by the global TEP. The method is presented in section 3 of D8.4a [6] and the prototype in section 3 of D8.4b [7].

The described method is not parallelized. Therefore, on each of the 9 servers, the candidate selection is computed on 1 core for the corresponding couple (scenario, time horizon). However, inside the candidate selection, the MIP solver FicoXpress [12] is used. FicoXpress is able to run in parallel on several cores, in order to launch simultaneously different linear-solver options (namely dual simplex, primal simplex and interior point method). In the case of candidate selection, our experiments showed that these features could lead to improvements, so we allowed FicoXpress to use all available cores for the optimization problems solved as a part of candidate selection.

The results of candidate selection are sent to the master server.

5.5. Transmission Expansion Planning optimization

The transmission expansion planning optimization step is run only on the master server. All selected snapshots and candidates are sent to the master server. The script `starter_TEP.sh` takes all these data and creates the main optimization problem for zonal TEP, containing all selected snapshots and candidates, for all scenarios and time horizons. This problem is solved using FicoXpress. Only one server is used at this stage, but all 16 cores of this server are used by FicoXpress. The method is presented in section 4 of D8.4a [6], and the prototype in section 4 of D8.4b [7].

6. Step 5: Grid expansion at nodal level with TEPES

The results of expansion planning optimization (at the end of Step 4) are investment decisions for the first time horizon, together with compatible investment decisions for each scenario for the two last time horizons.

The latter investment decisions are not used in Step 5: only investment decisions for the first time horizon are kept for Steps 5 and 6. These investment decisions were computed at zonal level, corresponding to corridors-capacity modifications. In Step 5, the software TEPES [20] is used to implement these decisions at nodal level.

TEPES is developed for Windows platforms and uses GAMS [21] as an optimization software (GAMS is similar to AMPL [11]). All input data, including nodal existing network description, generation, demand scenarios and zonal capacity investment decisions are inserted in a dedicated Excel file. From this Excel file, GAMS execution is run and results loaded in the Excel file.

More precisely, the following parts are subsequently executed by TEPES:

1. Input data extraction from Excel to text files
2. Text files upload into GAMS model
3. Computation of the optimal nodal-expansion solution
4. Results are written by GAMS in GAMS' internal format named GDX format
5. Upload of results from GDX files to initial Excel file.

Parts 1 and 5 are optional.

Text files created in Part 1 have standard formats and can be easily created directly from AMPL software or through Python routines on Linux Server.

Starting directly from the text files created in Part 1 on a Windows platform, we were able to successfully run parts 2 to 4 on the project's Linux server. It has also been checked that optimization was correctly computed. The results were written by GAMS in its GDX format.

GAMS software had been installed beforehand on a Linux server with standard installation procedures. A utility executable named GDXDUMP is provided by GAMS. This GDXDUMP allows extracting results from GDX files to standard text files.

This proves that TEPES could successfully be integrated on a Linux server within an automatic process.

7. Software

7.1. Task scheduler

The role of a task scheduler is to launch all scripts on all servers, offering a tool to organize and monitor scripts dispatching and sequencing.

The Slurm open source software [16] was selected. Authentication tool used inside Slurm is Munge [22].

After installing Slurm on all 9 servers, a master node has to be selected. After the development of all appropriate scripts, all tasks may be launched from the master server:

- Source code, binaries and input data are deployed from the master server to all other 8 servers,
- Steps 1 to 4 are launched successively, using 9 servers for all parts, except for the network reduction and the last TEP optimization, which are run only on the master server.

7.2. Other external software and languages

For the modelling and creation of optimization problems, AMPL [11] is used. Since this program is very easy to use and has a syntax very similar to mathematical equations, it is commonly used by optimization practitionners.

For solving linear and MIP (Mixed-Integer Linear Programming) problems, FicoXpress [12] is used.

For time-series analysis, Monte-Carlo sampling and network reduction Matlab [17] is used. The compiler module in Matlab allows compiling Matlab source code and deploying resulting binaries on all computation servers.

Python [18] is used as a language for the calculation of Overload indicators, Snapshot selection and Candidate selection.

BASH [19] shell is used for data gathering, general scripting and task scheduling together with Slurm.

Data security management is the following: Munge [22] authentication service is used for Slurm task scheduling; Openssh software [23] (more precisely: scp client of Openssh) is used for encryption of all data exchanged between servers.

TEPES [20] is developed with GAMS [21] software.

7.3. Source code tour

In the following table, all source code files for Step 1 to Step 4 are listed. The number of lines gives a flavour of the amount of work done. The total volume for Step 1 to Step 4 is about 28,000 lines.

# of lines	Name of source code file	Part of prototype
26	send_notif_mail.sh	Task scheduler
99	config.sh	Task scheduler
779	master_cluster.sh	Task scheduler

3431	build_scenario	Step 1
3945	getnewmc	Step 1
6125	time_series_analyzer	Step 1
252	onemcyear.sh	Step1+Step2
117	oneweek.sh	Step1+Step2
23	retry_invalid_years.sh	Step1+Step2
5	send_end_master1.sh	Step1+Step2
4	starter_gen_architecture.sh	Step1+Step2
27	starter_get_mc_years.sh	Step1+Step2
48	starter_mc_years.sh	Step1+Step2
76	concate_daemon.sh	Step1+Step2
52	gen_architecture.sh	Step1+Step2
253	hydro_scheduling.mod	Hydro Scheduling
218	hydro_scheduling.run	Hydro Scheduling
386	adequacy.mod	Adequacy
641	adequacy.run	Adequacy
251	disaggregation.run	Automatic mapping
337	wp8dcopf.mod	DC OPF
483	wp8dcopf.run	DC OPF
152	indicatorsCalculateAll.py	Olverloads indicators
177	indicatorsCalculateYear.py	Olverloads indicators
6	starter_network_reduction.sh	Network Reduction
92	network_reduction.sh	Network Reduction
3102	network_reduction	Network Reduction
12	starter_get_network_red.sh	Step 4
14	starter_mc_years_red.sh	Step 4
48	onemcyear_red.sh	Zonal DCOPF
69	oneweek_reduced.sh	Zonal DCOPF
341	dcopf_reduced.mod	Zonal DCOPF
421	dcopf_reduced.run	Zonal DCOPF
19	data_dcopf.run	Snapshot Selection
119	dcopf.mod	Snapshot Selection
24	dcopf.run	Snapshot Selection
66	accl_method.py	Snapshot Selection
130	feature_construction.py	Snapshot Selection
228	feature_construction_all.py	Snapshot Selection
130	feature_construction_F1a.py	Snapshot Selection
190	kmeans_method.py	Snapshot Selection
131	kmeans_method_old.py	Snapshot Selection
91	kmedoids_method.py	Snapshot Selection
143	plot_clustering.py	Snapshot Selection
165	snapshot_selection.py	Snapshot Selection
301	snapshot_selection_clara.py	Snapshot Selection

265	snapshot_selection_medoids.py	Snapshot Selection
132	snapshot_selection_multi.py	Snapshot Selection
53	util.py	Snapshot Selection
29	snapshot_selection.sh	Snapshot Selection
16	starter_end_snapshot_selection.sh	Snapshot Selection
32	starter_snapshot_selection.sh	Snapshot Selection
223	candidate_analysis.run	Candidate selection
300	candidate_management.run	Candidate selection
291	common.mod	Candidate selection
161	generate_feasible_cand.py	Candidate selection
134	merge_candidates.py	Candidate selection
127	unique_potential_cand.py	Candidate selection
112	unique_potential_cand_v1.py	Candidate selection
600	analysis.py	Candidate selection
61	candidate_selection.sh	Candidate selection
89	candidate_selection.sh	Candidate selection
11	candidate_selection_parallel.sh	Candidate selection
50	starter_candidate_selection.sh	Candidate selection
21	starter_merge_candidate.sh	Candidate selection
181	inv.mod	TEP
155	opex.mod	TEP
266	inv_opex	TEP
392	inv_opex.mod	TEP
138	inv_opex.run	TEP
31	starter_TEP.sh	TEP

8. Conclusion

During project e-Highway2050, several teams developed prototypes to implement the global methodology drawn in Figure 1. These prototypes have been presented in deliverables D8.2b [3], D8.3b [5], D8.4b [7] and D8.5b [9]. Steps 1 to 4 have been integrated on a single medium size cloud computing system. All computation and data had to be well organized to run computations on a group of 9 servers with 16 cores each. Most parts of the computation could be efficiently parallelized. Massive computations have been run over 3 scenarios, 3 time horizons and a total amount of 900 Monte-Carlo years, leading to 47k adequacy MIP problems and 15.7M DC OPFs.

In order to extend to more than one server per couple (scenario, time horizon), special attention should be paid to Snapshot selection and Candidate selection. Indeed, these two parts of Step 4 are now running with access to all Monte-Carlo simulation results for one couple (scenario, time horizon). If these data were stored on more than one server, these two modules should be modified.

TEP optimization is the final bottleneck. It is a fundamental choice of methodology of WP8 to solve it in one single TEP the global optimization problem. The commercial solver FicoXpress allows to use all cores of a server to solve this MIP problem on a parallelized Branch-and-Bound tree. In order to enlarge the number of snapshots selected for TEP computation, cloud computing systems dedicated to parallel optimization should be used. These systems are not standard at the date of writing (2015), but are being developed or tested by main optimization companies.

REFERENCES

[1] e-Highway2050, http://www.e-highway2050.eu
Deliverables available at: http://www.e-highway2050.eu/results

[2] e-Highway2050 (2015). D8.2a: Enhanced methodology for the computation of Demand and Generation scenarios. PDF

[3] e-Highway2050 (2015). D8.2b: Demand/Generation scenarios prototype. PDF

[4] e-Highway2050 (2015). D8.3a: Enhanced methodology to define optimal grid architectures for 2050. PDF

[5] e-Highway2050 (2015). D8.3b: Prototype for the definition of optimal grid architectures for 2050. PDF

[6] e-Highway2050 (2015). D8.4a: Enhanced methodology to define the optimal modular plan. PDF

[7] e-Highway2050 (2015). D8.4b: Prototype for optimal modular plan to reach 2050 grid architectures. PDF

[8] e-Highway2050 (2015). D8.5a: Enhanced methodology to assess robustness of a grid architecture. PDF

[9] e-Highway2050 (2015). D8.5b: Prototype to assess robustness of a grid architecture. PDF

[10] e-Highway2050. D8.6a: Detailed enhanced methodology for long-term grid planning.

[11] AMPL: A Modeling Language for Mathematical Programming, by Robert Fourer, David M. Gay, and Brian W. Kernighan. http://www.ampl.com

[12] FICO Xpress Optimization Suite, FICO, http://www.fico.com/en/products/fico-xpress-optimization-suite

[13] iTesla: Innovative Tools for Electrical System Security within Large Areas, http://www.itesla-project.eu
Electronic versions of most papers and presentations of iTesla project are available online here: http://www.itesla-project.eu/publications

[14] PRACE Partnership for Advanced Computing in Europe, http://www.prace-ri.eu
Curie system: http://www.prace-ri.eu/prace-resources

[15] OVH hosting provider, http://www.ovh.com

[16] SLURM workload manager http://slurm.schedmd.com

[17] Matlab http://www.mathworks.com

[18] Python programming language http://www.python.org

[19] BASH is the GNU Project's shell http://www.gnu.org/software/bash

[20] TEPES Long-Term Transmission Expansion Planning Model for an Electric System http://www.iit.upcomillas.es/aramos/TEPES.htm

[21] GAMS General Algebraic Modeling System http://www.gams.com

[22] MUNGE authentication service http://dun.github.io/munge

[23] OPENSSH free version of SSH encryption service http://www.openssh.com