Platform Calibration for Load Balancing of Large Simulations: TLM Case

Cristian Ruiz, Mihai Alexandru, Olivier Richard, Thierry Monteil, Hervé Aubert

Platform calibration for load balancing of large simulations: TLM case

Cristian Ruiz*, Mihai Alexandru†‡, Olivier Richard*, Thierry Monteil†‡, Hervé Aubert†‡
*INRIA MESCAL, LIG, 655 avenue de l'Europe, 38330 Monbonnot Saint Martin, France
†CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
‡Univ de Toulouse, INP, INSA, LAAS, F-31400 Toulouse, France
Cristian.Ruiz@imag.fr, Mihai.Alexandru@laas.fr

***Abstract*—The heterogeneous nature of distributed platforms such as computational Grids is one of the main barriers to effectively deploy tightly-coupled applications. For those applications, one common problem that appears due to the hardware heterogeneity is the load imbalance which slows down the application to the pace of the slower processor. One solution is to distribute the load adequately taking into account hardware capacities. To do so, an estimation of the hardware capacities for running the application has to be obtained. In this paper, we present a static load balancing for iterative tightly-coupled applications based on a profile prediction model. This technique is presented as a successful example of the interaction between experiment management tools and parallel applications. The experiment management tool *Expo* is used that enabled to: (1) provide a general, lightweight and descriptive way to capture the tuning and deployment of a parallel application in a computing infrastructure, (2) perform the tuning of the application efficiently in terms of human effort and resources needed. This paper reports the costs for carrying out the tuning of a large electromagnetic simulation based on TLM for the platform Grid'5000 and the improvements obtained on the total execution time of the application.**

***Keywords*—Load balancing, Experiment methodology, Large scale system, Grid computing, High performance computing, Transmission-line matrix methods.**

I. Introduction

High Performance Computing (HPC) strives to achieve the maximum performance of a given machine. The increasing complexity of computing hardware architectures nowadays, makes rise the number of variables to take into account to achieve this maximum performance and it is even worse when considering heterogeneous infrastructures as computational Grids. A common problem is the computation imbalance present in tightly-coupled applications that run in Grid infrastructures which is due to the unawareness of the underlying infrastructure characteristics. One of the best options to get the maximum performance is to tune the application code for a given architecture. This approach is used by ATLAS[1] which gets its speed by specializing itself for a given platform. Architecture aware tools such as hwloc[2] are now available in high performance runtime environments of parallel applications. Therefore, a deep knowledge of the underlying infrastructure and application is the evident trend to achieve the best performance. For some regular scientific codes, it is possible to derive a performance model and the tuning of the application can be guided based on this performance model[3]. This performance model can be constructed either from a detailed understanding of the application execution or by analyzing multiple runs. A multiple-runs approach is simpler because it takes into account the complex interaction between the application and for instance the memory hierarchy. To do so, several tools such as profilers, tracers, statistical engines, runtime environments have to be linked together in order to carry out the task of automating the generation, collection and treatment of performance information and provide the appropriate data to create the model.

In this paper, it is shown how parallel applications can take advantage from experiment management tools. A technique of load balancing for large simulations codes based on a prediction model is analyzed. This technique relies on the interaction between experimental management tools and parallel applications. The technique is applied to a large electromagnetic simulation code based on Transmission-Line Matrix (TLM) numerical method [4], deployed in a heterogeneous Grid infrastructure. This technique is classified as a *Static* load balancing which is well adapted to highly regular applications. It requires few changes to the application code compared to adopting a new programming model and given the high memory requirements of the application, a dynamic approach would generate a considerable overhead. The improvements done to the experimental management tool *Expo*[5] are shown as well. This enabled us to manage the modeling workflow where the execution of big campaigns of application runs are needed and the orchestration of different tools that could participate in the process of creation of the performance model. Doing this task efficiently is important in order to not delay the execution of the real application, reduce the perturbation of the results and provide in a short period of time valuable information to the application.

The contribution of this paper is twofold:

- Show the importance of experiment management tools in helping users to manage the complexity of distributed infrastructures, to automate several tasks and to make efficient use of computational resources.
- A load balancing technique for regular scientific codes based on the calibration of the platform and a prediction model. The approach is not expensive in terms of code source modification, user intervention and presents almost no overhead. An average improvement of 36% in the execution time is achieved.

The rest of the paper is organized as follows: Section II presents related work in load balancing techniques for parallel applications and experiment management tools, which is followed by a focus on the improvements done to our tool, enabling it to manage an experiment workflow in a flexible and efficient way (Section III). Section IV describes the approach to achieve load balancing of large simulation codes. Finally, the results are given in Section V and the conclusions in Section VI.

II. Related work

The related work is organized into two parts: the load balancing techniques in parallel applications and the different techniques to carry out such a task. The second part presents the state of the art of experiment management tools and works related to the benchmarking of Grid platforms.

A. Load balancing of distributed applications

An important phase of the execution of parallel codes is the assignment of work to compute units. The problem of load balancing then is defined as the assignment of work to the compute units according to its performance or load. This assignment of work can occur at the startup of the application (static partitioning) or it can happen several times during the execution of the application (dynamic partitioning). Both of them will be described in the following subsections.

1) Dynamic techniques: Dynamic techniques are very popular now given the apparition of infrastructures such as cloud computing. It is the case of Charm++ runtime system [6] which through continuous estimation of processor load, it adapts to the imbalance created by known fluctuations in shared infrastructures. Another approach based on Charm++ [7] takes into account the latency existing in cross-site communications for Grid infrastructures. As it can be very cumbersome to convert applications to newer paradigms such as Charm++, AMPI was proposed in [8] which enables a bigger number of application benefits from the framework features as load balancing. These dynamic techniques were mainly created due to the large presence of high irregular load in parallel computational science and engineering. Our approach applies to highly regular codes executed on Grid infrastructures where the CPU is not shared between users. Therefore, the gain obtained with a dynamic approach would be negligible and there exist a potential overhead of context switching and migration.

2) Static techniques: In [9], a static load balancing technique for mapping iterative algorithms onto heterogeneous clusters is presented focusing on the complexity of application partitioning and on efficient heuristics for the distribution schemes. Load balancing for Grid applications is proposed as well by PaGrid[10] which proposes a partitioner to balance mesh based applications. A graph is generated for the platform where processors are weighed according to its relative performance at executing standard benchmarks. This graph is matched with the graph generated for the application. In [11] is described a resource-aware partitioning where information about a computing environment is combined with traditional partitioning algorithms. The approach collects information about the computing environment and processes it for partitioning use.

B. Experiment management tools

GrapBench[12] provides a framework to carry out a semi-automatic benchmarking process for studying application behavior in grid infrastructures. The framework controls the number of benchmarking measurements required by a given application which are managed then by its experiment engine. The work outlined here differs from this in that it provides a more general experiment engine conceived to carry out any kind of study for an application in distributed platforms. Plush[13] is a widely used tool in PlanetLab, for deploying and monitoring application execution in distributed platforms. It provides abstractions to specify the steps to deploy an application, however, a real experiment entity is not taken into account. The inflexibility of its description language makes it difficult to write parametric studies. ZENTURIO[14] enables the management of parametric studies for an application in a framework for experimenting, but their high number of modules makes it difficult to port it to different platforms.

Workflows engines are well known for their capacity for carrying out parametric studies. Vistrails[15] provides parameter exploration and comparison of different results. It improves the experimentation activity providing data provenance tracking mechanisms. One limitation of Vistrails is its inability to adapt to distributed environments. Pegasus[16] offers a mapping between tasks in a workflow and distributed infrastructures (cloud, grid, clusters). Despite the capacity of some workflow engines to use distributed infrastructures, it is difficult to use them when considering the setup of an application. This setup could incur several complex steps that need a constant supervision. The approach proposed in this paper addresses those issues giving a flexible and lightweight experiment engine. The engine is based on two abstractions *resources* and *tasks*, the latter can be combined to represent a workflow. The workflow specification describes all the experiment activity: platform access, application deployment and setup, application execution, analysis and generation of results.

C. Transmission-Line Matrix

The main idea of this application is to simulate the propagation of an electromagnetic field inside large structures such as tunnels and airplane cabins. TLM numerical method models the electromagnetic field propagation by filling the space with a network of transmission-lines fed by electrical signals whose voltage and current correspond to the electric and magnetic fields. The intersection of these lines, that have the free-space impedance, is modelled with the Symmetrical Condensed Node (SCN) [17] scheme, whose scattering matrix is derived directly from the behavior of the fields. The TLM method requires significant computing resources, but its algorithm has the advantage of being parallelizable, which makes it possible to simulate oversized structures on multiple computing machines. Using a parallel approach, large electromagnetic structures can be modeled by means of large scale computing systems such as Grid or supercomputers in a HPC scenario.

In order to avoid a heavy TLM calculation, the discretized domain is sliced into several sub-domains that are assigned to the processors where will be computed in parallel. The CPUs communicate between them to achieve the job. The parallel approach, based on Message-Passing Interface (MPI),

is designed for Single Program Multiple Data (SPMD) programming model as it is presented in [18]. In the proposed parallel TLM application, a one-dimension Cartesian topology is implemented for the partitioning process.

III. EXPO EXPERIMENT MANAGEMENT TOOL

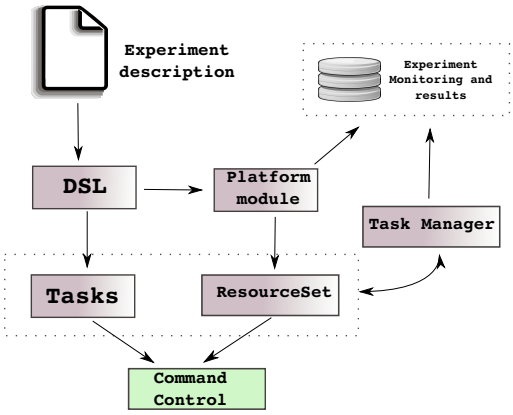

Fig. 1: Expo architecture

Expo is an experiment management tool designed to simplify and automate the conduction of experiments in distributed platforms. All the experimental plan is captured (i.e., access to the platform, experiment setup, experiment execution, results analysis, etc.) in a workflow where sequences of commands are grouped together in tasks and dependencies. This facilitates the recreation of the experiment setup and in turn, it will make easier the replay of experiments. Replayability of a computational experiment is the first step towards experiment reproducibility. The workflow tells how all the different tasks have to be called in order to get the results of the experiment. It comprehends tasks that can be executed sequentially, in parallel, asynchronously, etc. *Expo* strives to simplify the description of an experiment by providing a concise and readable way to describe it, specially when dealing with a big amount of nodes. It relies on parallel command executors as TakTuk[19] which makes it scale with a big amount of nodes.

Expo architecture is described in Figure 1, which mainly consists in six components: a Domain-Specific Language (*DSL*) module features a flexible description language built on top of Ruby[1] that enables to exploit all its richness in available libraries and mainly its descriptiveness. The *DSL* flexibility and scalability relies on two abstractions: *ResourceSet* and *Tasks*. Those components interact together in order to provide the necessary information to the *Command Control* and help it in translating the experimental plan into commands. The platform dependent module enables the interaction with different platforms such as: Grid'5000, PlanetLab[2], cloud computing infrastructures, computing clusters, etc. This module works as an interface for the *DSL* module, making an experiment description independent from the platform. *Expo* makes few assumptions about the resources to manage, relying on common system utilities suchs as: scp, ssh, unix commands, TakTuk which can deploy itself. It only requires to run a Ruby interpreter and few ruby libraries as described in its website [3]. Thus, *Expo* architecture is very simple and ligthweight. The schedule of the experimental workflow is done by the *Task manager* which is in charge of the results collection and experiment monitoring.

A. Expo ResourceSet

A *ResourceSet* is an abstract view of the resources and their organization in distributed computational infrastructures such as Grids. This abstraction was conceived in order to provide to the user a concise way to express actions that have to be carried out for a set of resources. Resources can be any computing unit: processor cores, processors, nodes, clusters, sites, etc. In Table I are shown some operators which gives to *Expo* a high flexibility against another approaches in the description language[5]. At the same time, this abstract view enables the generation of efficient parallel topology aware commands.

B. Expo Tasks

Expo adopts the notion of task, already exploited in workflow management tools as [20] and Rake[4] as well as web application deployment frameworks such as Capistrano[5]. A *Task* describes what to do and the *ResourceSet* tells the experiment management where to execute the task. Tasks can be triggered by events (e.g, availability of jobs in the infrastructure). Therefore, a complete unattended experiment campaign can be carried out. In Listing 1, an example of a definition of a task is shown. The compilation of a source code instrumentation package is performed. This task is executed on a *ResourceSet* which is represented by the variable *resources*. For this case a parallel command will be generated that will carry out the task for every machine represented in the *ResourceSet*. This task could be useful when compiling a program for different architectures.

Listing 1: Task abstraction

```
task :compile, :target => resources do
  run("cd ~/Test_profiling/; tar -xf pdt.tgz")
  run("cd ~/Test_profiling/pdtoolkit-3.17/; ./configure")
  run("cd ~/Test_profiling/pdtoolkit-3.17/; make install")
end
```

C. Expo experiment mapping

Workflow engines map scientific workflows to distributed platforms in an automatic form. Their mapping decisions are driven by minimizing the time to run the workflow. Given that the objective of a workflow is to perform a big computation, it is more flexible when mapping the workflow into the computing platform. In contrast, an experimenting workflow aims at performing tests. Some tests are targeted to a certain machine architecture and it is important to take this into account when performing the mapping of the workflow. Consequently, a way to control the underlying infrastructure has to be provided. There is a trade-off between descriptiveness and scalability (efficient mapping). In Figure 2 is explained the procedure to map an experiment description into a distributed platform,

[1] https://www.ruby-lang.org
[2] https://www.planet-lab.org/
[3] http://expo.gforge.inria.fr/
[4] http://rake.rubyforge.org/
[5] http://www.capistranorb.com

runs the command in parallel for all the nodes of the cluster 1	`run("make lu NPROCS=8 CLASS=A MPIF77=tau_f90.sh",:target => resources[:cluster_1])`
runs the command hostname for each node sequentially	`resources.each{ |node| run("hostname",:target => node) }`

runs the command for different set of resources, the length of the sets generated are powers of two.	`resources.each_slice_power2 do |nodes|`
	`  run("mpirun -np 2 --machinefile #{nodes.nodefile} ./app",:target => nodes.first)`
	`end`

selects the resources of a specific cluster, it keeps the topology of the *ResourceSet* in order to generate the right parallel command.	`fast_cluster = resources.select(:cluster){ |cluster|`
	`  cluster.properties["clock_speed"]>1700000000`
	`}`
	`run("~/benchmarks/NPB3.2-OMP/bin/BT.A_out.4",:target => fast_cluster)`

TABLE I: ResourceSet operations

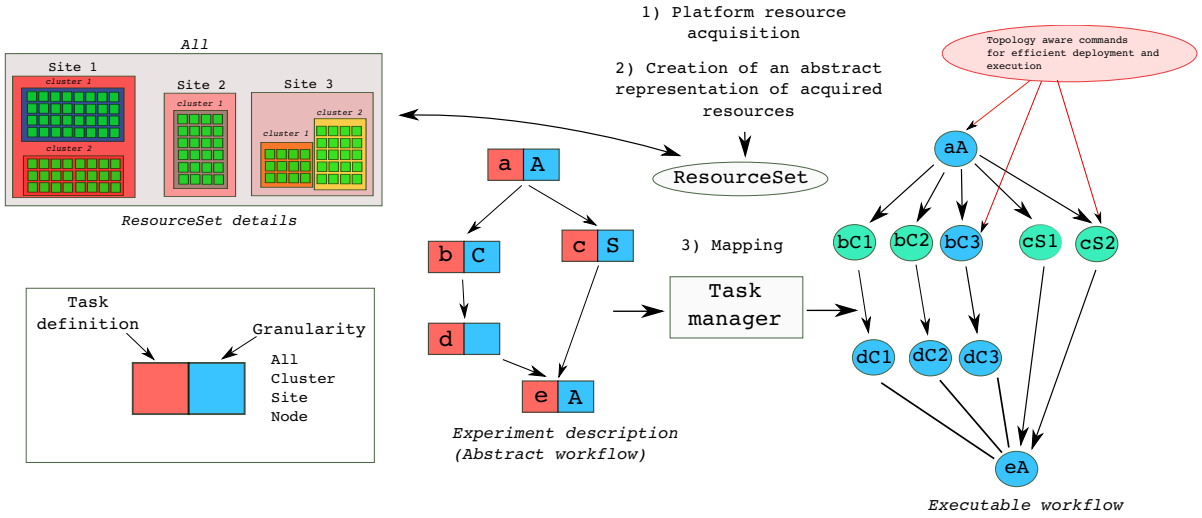

Fig. 2: Expo workflow mapping. Tasks are split according to the granularity of execution, generating sub-tasks for the executable workflow. In the Figure, tasks are generated for 3 different clusters and 2 sites. The *Task manager* uses the information provided by the *ResourceSet* to generate the topology aware commands

in this particular case a Grid computing infrastructure. 1) The experiment management tool contacts the platform in order to ask for resources, this step is known as *resource acquisition*. 2) Once the resources are available a *ResourceSet* is created which is an abstract representation of the resources that will be used for the experiment. As already said, an experiment is described as a workflow composed of tasks and dependencies between them. This initial workflow is known as abstract workflow which main goal is to capture the experiment activity. Two important information are: the body of the task which is simply all the sequence of commands to execute and the granularity of execution. For the example shown, this granularity can be: all resources, site, cluster, node, etc. The task manager will be in charge of taking this abstract workflow and map it into the infrastructure. It uses the information provided by the granularity of execution in order to generate the executable workflow. This is an expanded version of the abstract workflow, where tasks have been split according to the granularity of execution. This enables to choose the best type of execution (parallel, asynchronous, parallel-asynchronous, etc) and the less expensive in terms of number of connections with the remote machines and threads created for control the experiment. The tasks created at this level guarantees the generation of topology aware commands with TakTuk for an efficient deployment and execution. The scalability of commands execution was already shown in a previous publication[5].

IV. Load Balancing Approach

Here, the technique of load balancing applied to the TLM application is described. Considering a fully heterogeneous infrastructure, such as Grid'5000, a Grid computing with many clusters geographically distributed composed of different hardware configurations. The application needs to assign an adequate workload for each node in order to fully exploit the infrastructure capacities. Given that the application is highly regular as shown in [18], a static load balancing technique is chosen, where all the work is divided and distributed at the beginning. The amount of work assigned to each processor depends on the relative performance of the application on such processor. As this relative performance can be difficult to get from processor characteristics, a prediction model is used in order to have a more accurate indicator. It was already shown that the expected runtime of the computation part of the application scales linearly with the number of TLM cells N_x, N_y, N_z on the three Cartesian directions, y being the partitioning direction. Thus, a simple linear function given in [18] is used to model the performance:

$$T_{calc} = c_1 + c_2 N_x N_y N_z t, \tag{1}$$

where $c_{1,2}$ are the time coefficients corresponding to different blocks of the TLM application and t represents the number of computing iterations. The prediction model, given in (1), takes into consideration the algorithm to be executed and the processor architecture performing the computation. They represent the processor architecture information inside the prediction model. This model takes into account the effects of cache misses, according to the problem size. The first term may be neglected as it is very small compared to the second one. Lets consider that the partitioning procedure gives the length of the computing sub-domain assigned to the process *i*, as:

$$l_i = \alpha_i N_y, \tag{2}$$

with

$$\sum_{i=1}^{p} \alpha_i = 1$$

for all *p* processes the structure is computed by. Consequently, the amount of work is distributed according to the fact that the computation time has to be the same for each process *i*:

$$T_{calc_i} = c_i N_x l_i N_z t, \forall i \in [1, p] \tag{3}$$

where c_i is the second coefficient from (1) corresponding to the process *i*. This leads to describe (2) by:

$$l_j = \frac{N_y}{c_j \sum_{i=1}^{p} \frac{1}{c_i}}, \tag{4}$$

where l_j is the work assigned to the process j. Therefore, a construction of a prediction model of the application for each different computing hardware available on the Grid infrastructure has to be performed. In order to have a good prediction model, a given set of chosen simulations have to be run and analyzed for each different machine. *Expo* is used to automate the task of conducting this big number of executions. This process will be called *calibration*. The module used to this end is described in Section IV-A. The load-balancing approach implemented in this work considers the communication between different clusters being homogeneous. The communication capabilities of the computing environment are not taken into account. Not all resources have to be involved especially when the structure to be computed is not so large, because the communications due to an excess of processors may slow down the entire simulation, despite the increased accumulated speed.

The execution of the application will be wrapped in two *Expo* modules, which will automate all the process in the platform chosen for testing (Grid'5000).

- Calibration of the platform. This module runs once, it can contact the platform in order to know if there has been a change in the hardware configuration and deploys the necessary calibration.
- Deployment of the application. Generation of a file that contains platform fitness information for the application and carry out the load balancing at application level.

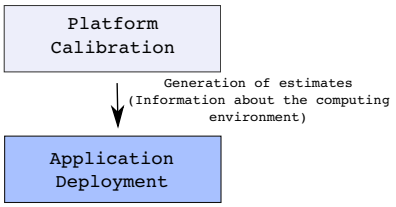

Fig. 3: Expo Modules: the calibration modules is executed once

A. *Expo calibration module*

All the procedure of platform calibration was captured using *Expo* tasks abstractions. The following tasks were defined:

1) run reservation: make a request to the computing platform in order to reserve the resources needed.

2) transferring code to each site on the grid: The code is sent from one chosen site to every site in Grid'5000.

3) extracting and compiling the code: The code is extracted and compiled with the right configuration.

4) calibration: It comprehends the execution of several simulations with different parameters. Two types of calibration are performed in order to take into account the cache effects.

5) compute coefficients: The statistical engine R[6] is used in order to process the files generated by the calibration and perform a linear regression in order to calculate the coefficients of the model.

6) free resources: It makes a request to the platform in order to free the resources used by the calibration.

These tasks were described using *Expo* DSL using 180 lines. An extract of the description is shown in Listing 2 and the different execution times of each task for different clusters are shown in Table II. It is important to note that the time to execute the whole module for a particular cluster mainly depends on the execution time of the simulations. There is an almost negligible overhead in the execution time with *Expo*, which was already shown in [5].

In Figure 4 is shown the executable workflow generated from the abstract calibration experiment definition. Here, the level of execution is the job. The system submits a job into the infrastructure for every different (different architecture) cluster in Grid'5000. Thereby, every task defined in the abstract representation is mapped into a cluster and managed asynchronously. Several machines were used per cluster in order to lower the time to get the results. The simulation were deployed in parallel for this case using TakTuk which enable us to maintain a low number of ssh connections to control the experiment. In Figure 5, it is shown the heterogeneity of Grid'5000 in terms of coefficients of the prediction model. This figure was generated using the results obtained by the calibration module.

Advantages of using Expo:

- It helps to deploy efficiently the simulations used for the calibration part, making independent from the

[6]http://www.r-project.org/

Task name	Execution time [sec] per cluster									
	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Transfert site	15.09	13.31	16.32	14.06	26.76	42.55	10.26	10.46	11.92	35.03
Compiling code	21.84	24.35	30.14	22.38	23.49	27.10	20.56	21.36	29.94	20.28
Calibration	1770.14	4860.31	3630.55	1770.47	4660.67	7590.81	1640.23	1600.83	3430.70	1620.87
Free resources	1.76	1.62	2.20	1.25	1.33	1.54	1.42	1.77	1.06	1.55

TABLE II: Execution time of the different tasks that compose the calibration module.

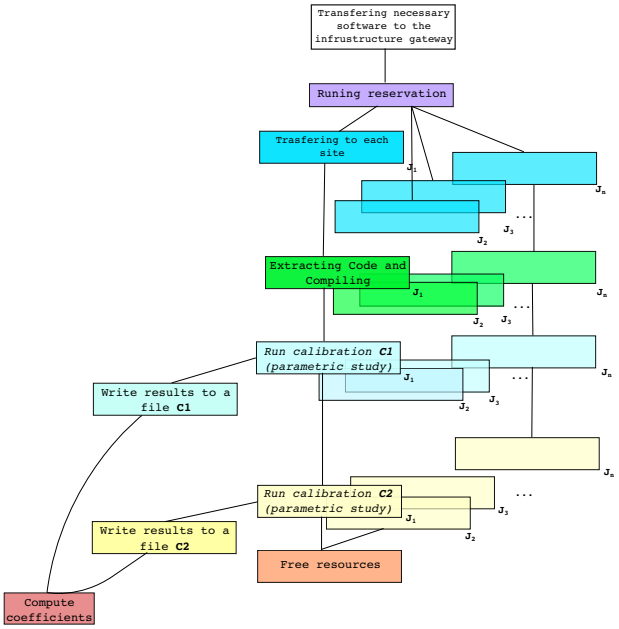

Fig. 4: Experiment calibration executable workflow

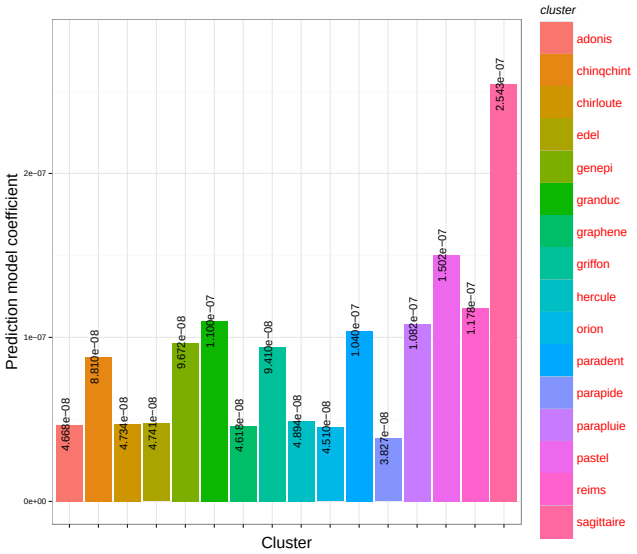

Fig. 5: Heterogeneity of Grid'5000

platform. More than 1359 simulations were necessary to get data for the prediction model.

- Makes all the procedure more reproducible and repeatable.
- Frees the application from implementing this functionality. Relying on more flexible languages for this task.

Listing 2: Extract of calibration module

```
task :transfering_tlm, :target => resources.gw do
    put("~/TLM/tlm_v1.tar","/tmp/tlm_test.tar",:method => "scp")
end

task :run_reservation, :depends => [:transfering_tlm] do
  reserv.run!
end

task :transfert_site, :target => resources, :depends => [:run_reservation] do
  options_put = {:method => "scp", :nfs => :site}
  run("mkdir -p ~/Exp_tlm")
  put("/tmp/tlm_test.tar","~/Exp_tlm/tlm_test.tar",options_put)
end

task :compiling, :target => resources, :depends => [:transfert_site] do

  check("ls ~/Exp_tlm/TLMME/") then
    run("cd ~/Exp_tlm/; tar -xf tlm_test.tar")
    run("make -C ~/Exp_tlm/TLMME/tlm/")
  end

end

task :calibration_c2, :target => resources, :depends => [:compiling] do

 params_c2.each_with_index{ |par,index|
    number_sim = 1
    RUNS.times do
      tag = {:parameters => par,:size => size_c2[index] }
      commands =["cd ~/Exp_tlm/TLMME/tlm/;./run 1 #{par} matched"]
      run(commands, :ins_per_machine => number_sim,:log => tag)
    end
    puts "Finishing parameter #{par}"
  }
end
```

V. Results

A. Experimental platform

The simulations were performed on Grid'5000 platform[21]. For performance reasons, only two processes are executed on grid nodes, each one on a different processor. The architectures of the computing nodes from Grid'5000 are different from cluster to cluster. The same clusters where used in order to keep the homogeneity between the experiment results concerning the simulation time. These clusters are geographically distributed in two sites. These sites are connected by RENATER, the French network for research and teaching. All *Expo* description files used two run the experiments are available in[7].

B. Using different configurations

Here, it was evaluated the performance gain obtained using load balance under different hardware configurations. In order to show the improvement in performance for large simulations, we opted for using different simulation sizes proportional to the number of nodes. This enabled to maintain a favorable rate between computation and communication. The results are shown in the Fig. 6. A maximum gain of 42.84% was obtained

[7] http://expo.gforge.inria.fr/

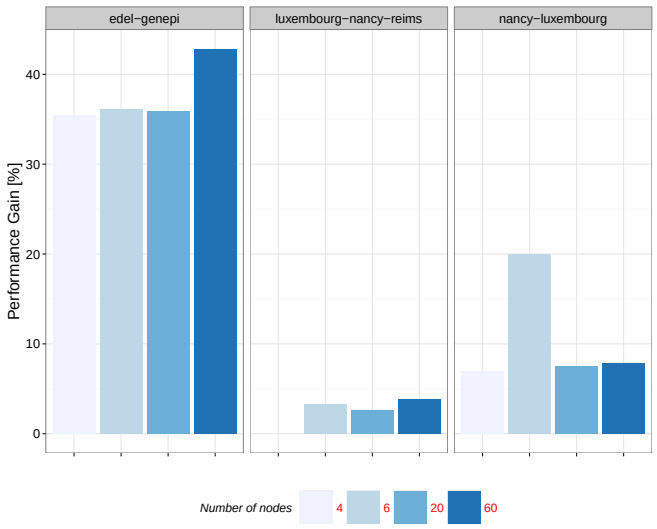

Fig. 6: Using different heterogeneous configurations. First tests used cluster located in the same site (*edel-genepi*). The other two series of test used different geographically distributed sites (*luxembourg, nancy, reims*).

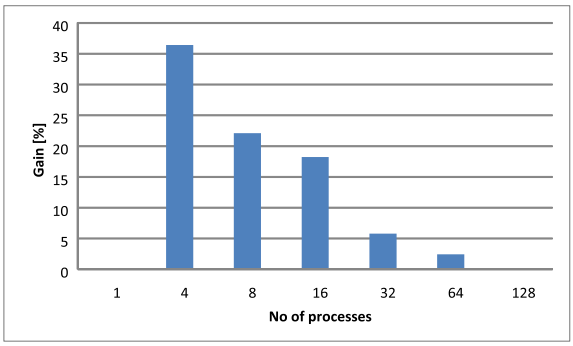

Fig. 7: Gain obtained with the same simulation parameters changing the number of nodes.

using clusters located in the same site. The gain obtained using several geographically distributed sites varies a great deal, we observed here performance gains ranging from 3.25% to 19.92%.

C. Changing the number of nodes

The experiment simulates the electromagnetic field propagation, using the TLM method, for 10000 time steps inside a waveguide structure, having the dimensions: 172 mm width, 86 mm height, 2432 mm length, a mesh step of 1 mm. In this experiment the computing nodes belong to Griffon, Chinqchint and Chirloute clusters. The simulation time values are presented in Fig. 7. The maximum gain obtained when using load-balancing approach is about 36%. The values of the simulation time when the load is balanced according to the calibration model given by *Expo* are smaller than the time values when the structure is divided identically on all MPI processes. The gain obtained by load balance approach decreases while the number of processes increases, because the computation time decreases according to communication time.

D. Large structure

In order to prove the real benefits of the grid environment for TLM large simulations, a supersized rectangular matched waveguide, discretized upon 95 million TLM cells is simulated. Its dimensions are: 345 mm width, 173 mm height, 1600 mm length and a mesh step of 1 mm.

1) Distributed experiment: In the first experiment, the simulations are performed using four nodes from Griffon and Chirloute clusters. The gain obtained by load balancing approach is about 25.5%.

2) Local experiment: A second experiment was carried out using nodes from clusters Parадent and Parapide which are localized on the same site. The gain obtained by load balancing approach is about 48.5%, much better than the distributed experiment because the communication time is much smaller between nodes on the same site.

VI. Conclusions and Future Works

This work showed the interaction between applications and experiment management tools, which is not limited to reproducibility purposes and replayability of experiments. This calibration is an example of how experiment management tools can free applications of doing cerating tasks and how can they help them to perform a tuning for a given platform. The use of tools as *Expo* serves the following purposes: it makes easy the access to complex platforms, helping non-expert users to make an efficient use of the resources. It helps to combine tools in order to capture the experimenting process.

It is difficult to perform an efficient deployment of the application using just information provided by the hardware. Performance models based on runs provide a more accurate information for using the platform resources more efficiently. At the same time, a load balancing based on a performance model gives to the application high flexibility for estimating the best work placing for a certain size given the hardware configuration.

In perspective, smarter reservation mechanisms taking into account the calibration and the availability of the platform, the different number of possible configurations for deploying and their cost represent a viable solution toward fast and automatic multidisciplinary application simulations.

VII. Acknowledgments

This research was supported by the Hemera INRIA large scale initiative.

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

REFERENCES

[1] R. C. Whaley and A. Petitet, "Minimizing development and maintenance costs in supporting persistently optimized BLAS," *Software: Practice and Experience*, vol. 35, no. 2, pp. 101–121, February 2005, http://www.cs.utsa.edu/~whaley/papers/spercw04.ps.

[2] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst, "hwloc: A generic framework for managing hardware affinities in hpc applications," in *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, 2010, pp. 180–186.

[3] T. Hoefler, "Bridging performance analysis tools and analytic performance modeling for hpc," in *Proceedings of the 2010 conference on Parallel processing*, ser. Euro-Par 2010. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 483–491. [Online]. Available: http://dl.acm.org/citation.cfm?id=2031978.2032045

[4] W. Hoeffer, "The transmission-line matrix method–theory and applications," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 33, no. 10, pp. 882–893, oct 1985.

[5] C. C. Ruiz Sanabria, O. Richard, B. Videau, and I. Oleg, "Managing large scale experiments in distributed testbeds," in *Proceedings of the 11th IASTED International Conference*, IASTED. ACTA Press, feb 2013, pp. 628–636. [Online]. Available: http://dx.doi.org/10.2316/P.2013.795-011

[6] A. Gupta, O. Sarood, L. Kale, and D. Milojicic, "Improving hpc application performance in cloud through dynamic load balancing," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, 2013, pp. 402–409.

[7] G. Koenig and L. Kale, "Optimizing distributed application performance using dynamic grid topology-aware load balancing," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, 2007, pp. 1–10.

[8] M. Bhandarkar, L. V. Kale, E. de Sturler, and J. Hoeflinger, "Object-Based Adaptive Load Balancing for MPI Programs," in *Proceedings of the International Conference on Computational Science, San Francisco, CA, LNCS 2074*, May 2001, pp. 108–117.

[9] H. Renard, Y. Robert, and F. Vivien, "Static load-balancing techniques for iterative computations on heterogeneous clusters," in *Euro-Par 2003 Parallel Processing*, ser. Lecture Notes in Computer Science, H. Kosch, L. Bszrmnyi, and H. Hellwagner, Eds. Springer Berlin Heidelberg, 2003, vol. 2790, pp. 148–159. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-45209-6_24

[10] S. Huang, E. Aubanel, and V. Bhavsar, "Pagrid: A mesh partitioner for computational grids," *Journal of Grid Computing*, vol. 4, no. 1, pp. 71–88, 2006. [Online]. Available: http://dx.doi.org/10.1007/s10723-005-9018-0

[11] K. D. Devine, E. G. Boman, and G. Karypis, "Partitioning and load balancing for emerging parallel applications and architectures," in *Frontiers of Scientific Computing*, M. Heroux, A. Raghavan, and H. Simon, Eds. Philadelphia: SIAM, 2006.

[12] F. Nadeem, R. Prodan, T. Fahringer, and A. Iosup, "Benchmarking grid applications," in *Grid Middleware and Services*. Springer US, 2008, pp. 19–37. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-78446-5_2

[13] J. Albrecht, R. Braud, D. Dao, N. Topilski, C. Tuttle, A. C. Snoeren, and A. Vahdat, "Remote control: distributed application configuration, management, and visualization with plush," in *Proceedings of the 21st conference on Large Installation System Administration Conference*, ser. LISA'07. Berkeley, CA, USA: USENIX Association, 2007, pp. 15:1–15:19. [Online]. Available: http://dl.acm.org/citation.cfm?id=1349426.1349441

[14] R. Prodan and T. Fahringer, "Zenturio: an experiment management system for cluster and grid computing," in *Cluster Computing, 2002. Proceedings. 2002 IEEE International Conference on*, 2002, pp. 9–18.

[15] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo, "Vistrails: visualization meets data management," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '06. New York, NY, USA: ACM, 2006, pp. 745–747. [Online]. Available: http://doi.acm.org/10.1145/1142473.1142574

[16] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny, "Pegasus: Mapping scientific workflows onto the grid," in *Grid Computing*, ser. Lecture Notes in Computer Science, M. Dikaiakos, Ed. Springer Berlin Heidelberg, 2004, vol. 3165, pp. 11–20. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-28642-4_2

[17] P. Johns, "A symmetrical condensed node for the tlm method," *IEEE Trans. on Microwave Theory and Tech.*, vol. 35, no. 4, pp. 370–377, apr 1987.

[18] M. Alexandru, T. Monteil, P. Lorenz, F. Coccetti, and H. Aubert, "Large electromagnetic problem on large scale parallel computing systems," in *International Conference on High Performance Computing and Simulation*, 2012.

[19] B. Claudel, G. Huard, and O. Richard, "Taktuk, adaptive deployment of remote executions," in *Proceedings of the 18th ACM international symposium on High performance distributed computing*, ser. HPDC '09. New York, NY, USA: ACM, 2009, pp. 91–100. [Online]. Available: http://doi.acm.org/10.1145/1551609.1551629

[20] M. Tanaka and O. Tatebe, "Pwrake: a parallel and distributed flexible workflow management tool for wide-area data intensive computing," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 356–359. [Online]. Available: http://doi.acm.org/10.1145/1851476.1851529

[21] Grid5000. (2013) Grid5000:hardware. [Online]. Available: https://www.grid5000.fr/mediawiki/index.php/Special:G5KHardware