HAL
open science

A Workflow for Fast Evaluation of Mapping Heuristics Targeting Cloud Infrastructures

Roman Ursu, Khalid Latif, David Novo, Manuel Selva, Abdoulaye Gamatié, Gilles Sassatelli, Dmitry Khabi, Alexey Cheptsov

To cite this version:

Roman Ursu, Khalid Latif, David Novo, Manuel Selva, Abdoulaye Gamatié, et al.. A Workflow for Fast Evaluation of Mapping Heuristics Targeting Cloud Infrastructures. DREAMCloud: Dynamic Resource Allocation and Management in Embedded, High Performance and Cloud Computing, Jan 2016, Prague, Czech Republic. lirmm-01265874

HAL Id: lirmm-01265874

https://hal-lirmm.ccsd.cnrs.fr/lirmm-01265874

Submitted on 1 Feb 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Workflow for Fast Evaluation of Mapping Heuristics Targeting Cloud Infrastructures

Roman Ursu*, Khalid Latif*, David Novo*, Manuel Selva*, Abdoulaye Gamatie*, Gilles Sassatelli*,
Dmitry Khabi†, Alexey Cheptsov†
*LIRMM / CNRS - University of Montpellier, France
{firstname.lastname}@lirmm.fr
†High-Performance Computing Center of Stuttgart (HLRS), Germany
{lastname}@hlrs.de

***Abstract*—Resource allocation is today an integral part of cloud infrastructures management to efficiently exploit resources. Cloud infrastructures centers generally use custom built heuristics to define the resource allocations. It is an immediate requirement for the management tools of these centers to have a fast yet reasonably accurate simulation and evaluation platform to define the resource allocation for cloud applications. This work proposes a framework allowing users to easily specify mappings for cloud applications described in the AMALTHEA format used in the context of the DreamCloud European project and to assess the quality for these mappings. The two quality metrics provided by the framework are execution time and energy consumption.**

I. Introduction

Usage of cloud infrastructures is by no means uncommon in today's scientific research. A typical cloud system is a sophisticated tool that relies on the possibility to combine computing, networking and storage capabilities of multiple compute resources to deliver a significant performance advantage over traditional desktop systems. Due to the size, cost and complexity of operation, cloud infrastructures are almost never used exclusively by a single user. Instead, a number of users try to compete for access, and thus mechanisms for effective resource sharing among multiple consumers need to be in place. A process of allocating resources efficiently and scheduling computational tasks onto these allocated resources in a proper manner is based on resource allocation and scheduling priorities. A typical scheduler targets multiple goals including maximization of throughput and minimization of response time (i.e., latency). Other optimization criteria such as CPU fairness or energy-awareness are becoming more important nowadays, too. Since some of these goals are conflicting with each other, a job of the scheduler is to figure out a reasonable allocation strategy. The approach of the DreamCloud project is to enable a closed-loop control mechanism that will allow for a dynamic resource allocation. In order to assess the quality of the resource allocation policy, the use of dedicated fast simulators is of an essential importance.

Contributions of this paper. To address the above requirement this work proposes a configurable framework for fast performance evaluation of different resource allocation strategies targeting cloud infrastructures. The implemented framework connects AMALTHEA [1] application models, used to describe application as a tasks graph in the context of the DreamCloud project (see Section III for the details of the model) to the SimGrid [2] simulation platform. It aims to be an open-source and flexible framework supporting a variety of state of the art mapping algorithms.

Outline. In the rest of the paper, Section II discusses related work. Section III introduces the AMALTHEA application modeling language. The core engine of our simulation framework allowing to simulate AMALTHEA applications on top of SimGrid is presented in Section IV. The proposed framework is introduced in Section V. Section VI presents the results of using this flow to evaluate the quality of different mappings for a scientific application case study executed on top of a typical cloud infrastructure. Finally, conclusions and perspectives are presented in Section VII.

II. Related Work

A variety of cloud application simulation frameworks has been proposed. CloudSim [3] is a Java-based software framework, which supports modeling and simulation of large scale cloud computing components and environments including data centers, virtual machines, service brokers, and provisioning policies. The features of CloudSim include the availability of a visualization engine that aids in the creation and management of multiple, independent, and co-hosted virtualized services on a data center node. Additionally, CloudSim provides flexibility to switch between space- and time-shared allocation of processing cores to virtualized services. These compelling features of CloudSim would speed up the development of new application provisioning algorithms for cloud computing. However, it is not possible to define the task level allocation policies for CloudSim.

iCanCloud [4] is an open source C++ based simulation platform which has been designed to model and simulate cloud computing systems. A user can customize the core hypervisor class, which in turn is the core of iCanCloud. The Amazon model has been integrated with the simulator. From the DreamCloud perspective, being a commercial tool, it does not provide complete access to flexibly deal with application requirements. There is no support for power consumption analysis. Additionally, the focus of iCanCloud is to define and integrate new brokering policies and minimize the user cost for public cloud infrastructure based on classical scheduling

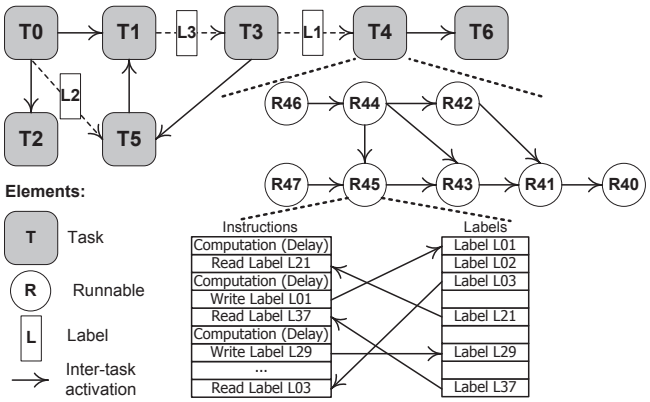


Fig. 1: An illustrative representation of an application as a task graph and task as a runnable graph, with inter-task activation and communication.

heuristic. While the main focus of the DreamCloud project is dynamic resource allocation.

GreenCloud [5] is a sophisticated packet-level cloud simulator with focus on energy consumption for cloud communications. It is a C++/OTcl based tool, with underlying NS-2 platform. GreenCloud provides the fine-grained modeling of energy consumption for the IT equipment in a data center, such as computer servers, network switches and communication links. From the DreamCloud perspective, GreenCloud does not provide the detailed timing analysis which is an important requirement for evaluating dynamic resource allocations. To introduce the detailed timing analysis, a designer needs to define timing models in NS2 environment, which is quite outdated and very complex tool.

SimGrid [2] is an open-source and mature C-based tool to study the behavior of large- scale distributed systems such as grids, clouds, HPC or P2P systems. It can be used to evaluate heuristics, prototype applications or even assess legacy MPI applications. Task level scheduling is supported by SimGrid. The power consumption analysis tool has been integrated to the latest version of SimGrid. Additionally, the developers are very reactive. Due to these facts, and making the basic study, SimGrid was selected for the integration to our design framework for performance evaluation of applications executed on top of cloud infrastructures.

III. Application Model

In the context of the DreamCloud project addressing resource allocation issues in future multicore and distributed systems. From the applications perspective, the project focuses both on automotive and on cloud computing domains. The AMALTHEA [1] application model has been chosen to capture application specifications from these two domains because it provides clean abstractions allowing to capture both of them. AMALTHEA is based on three main entities: **labels** representing data elements, **runnables** denoting the smallest units of code schedulable by an OS, and **tasks** defined as graphs of runnables.

In AMALTHEA, an application model is represented as a directed task graph, where vertices represent tasks, while directed edges represent either inter-task activation or communication. Figure 1 represents an arbitrary AMALTHEA application model with seven tasks and three labels. To specify the inter-task communication, it can be observed that task T1 communicates with task T3 via label L3. For this purpose, T1 will perform a write operation on L3 while T3 will perform the read operation on L3. The size of L3 represents the exchanged data volume. At lower granularity level, a task is composed of runnables as illustrated in Figure 1. Task 'T4' is composed of eight runnables. The runnable graph also provides information about precedence relationship between runnables.

A runnable is a set of "abstract" instructions. Such instructions in AMALTHEA application model are categorized as computation and communication instructions. Computation instructions feature operations such as arithmetic or logical operations. For quick simulations, the actual computational instructions are abstracted away by representing them with associated platform-dependent execution time in terms of clock cycles. Communication instructions comprise read/write accesses to labels, which can be further classified into two categories: local and remote accesses. Local accesses take place, when the target label is mapped at the same node as the requesting runnable. Remote accesses take place, when the target label is mapped on a different node, compared to that of the requesting runnable. In this case, the runnable goes to the wait state to wait for the read/write response and makes the core available for other runnables to execute their instructions. Figure 1 details the instructions of runnable R45 from the runnable graph of task T4. The computation instructions are represented with their execution delay values. The communication instructions are represented as read/write accesses to their corresponding labels. Their simulation consists of the actual transfer of data size corresponding that of accessed labels in read/write requests.

IV. From SimGrid to AMALTHEA

We now review how we implemented the simulation of AMALTHEA on top of SimGrid. Before digging into the details, we describe the basic architecture of SimGrid.

A. SimGrid Architecture

SimGrid has been an active project for more than 10 years. Since then, it has been evolving and taking new systems into consideration. At start, it was dedicated to grid simulation and its focus was to study centralized scheduling simulations (e.g., off-line scheduling of a Directed Acyclic Graph (DAG) on a heterogeneous set of distributed computing nodes). Nowadays, SimGrid deals with a variety of distributed systems ranging from grids to peer-to-peer systems, and arbitrarily defined hybrid systems combining different computing systems. The latest, stable and publicly available version of SimGrid (3.2) implements the architecture shown in Figure 2.

SimGrid provides three APIs, all based on a common kernel, allowing users to write custom simulations. SURF is this

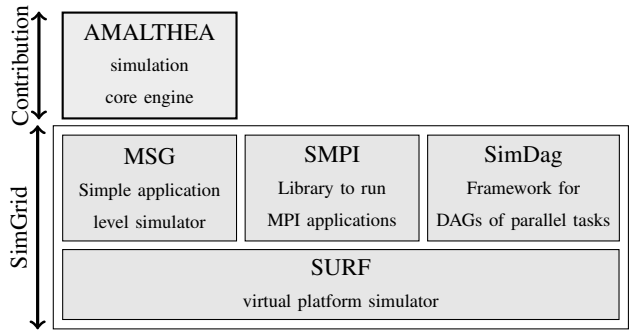


Fig. 2: SimGrid Architecture and the contribution of this work. SimGrid provides three different APIs allowing user code to write custom simulations. This work relies on the MSG API.

internal kernel. It provides the core functionalities to simulate a virtual platform. It has been implemented at very low level and is not intended to be used by the end users. On top of the SURF layer, SimDag, MSG, and SMPI APIs are implemented. MSG is a simple programming environment, which was the first distributed programming environment provided within SimGrid. It is still the most commonly used programming environment and can be used to build rather realistic simulations. The SimDag API provides specific programming environment for DAG applications while the SMPI one provides support for simulating MPI applications. To simulate the rich set of abstractions provided by the AMALTHEA model described in the previous section, we rely on the MSG API.

B. AMALTHEA Execution on SimGrid

To execute AMALTHEA applications on top of SimGrid, we implemented a simulation core engine that reads the in-memory object representation of an AMALTHEA model and converts it to SimGrid function calls. Each runnable of the initial application is converted into a SimGrid process. Runnables activation dependencies and communications through labels are then implemented using SimGrid messages.

In order to fit AMALTHEA applications into SimGrid, we simplified the AMALTHEA semantic by gathering for each runnable all the read instructions together, all the computing instructions together and all the write instructions together. From that, each runnable process first waits for receiving activation messages from all the runnables it depends on. Then, the runnable process sends messages corresponding to the read label instructions it contains. The computational load of the process required by SimGrid is then computed by adding the execution time of all the compute instructions of the runnable. Finally, the runnable process performs write label operations and activates all subsequent runnables.

V. Workflow

In order to be able to simulate AMALTHEA applications on top of cloud infrastructure we propose the workflow depicted on Figure 3. This workflow comprises the 4 following steps:

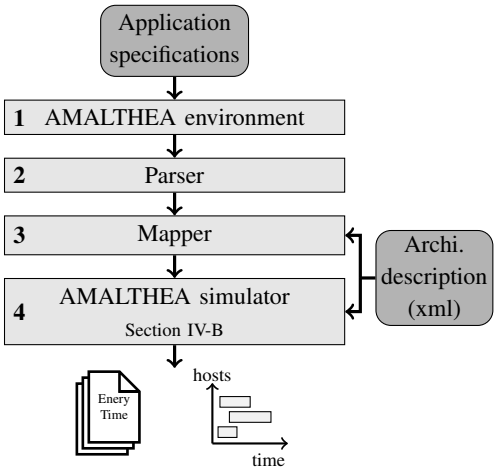


Fig. 3: Simulation workflow from applications specifications to simulation results

- **Step 1.** The application specifications are modeled in AMALTHEA as discussed in the previous section.
- **Step 2.** The AMALTHEA model is fed to a parser. The job of the parser is to translate the AMALTHEA model into a format that can then be easily simulated using SimGrid. This parser has been written in C++ to be easily integrated with SimGrid.
- **Step 3.** The mapper then executes the mapping algorithms by allocating tasks and labels onto the simulation platform. The description of the platform is provided to the mapper. Because this work focuses on providing a modular workflow to evaluate mapping strategies, the proposed workflow does not provide elaborated mapping strategies but rather a clean and simple interface to let users implement any mapping algorithm. Nevertheless, to illustrate the complete workflow, basic mapping strategies are proposed by default.
- **Step 4.** At this stage, the AMALTHEA runnables are executed on the simulation platform as described in Section IV-B. Ensuring the AMALTHEA execution semantic on top of the SimGrid MSG API was the biggest implementation challenge of the workflow.

As results, the simulation platform provides information about execution time, execution steps and the energy consumption. All these results are provided in the form of text files. The energy consumption results are provided both globally and per host. Moreover, the workflow provides a timeline view of runnable processes execution on the different hosts of the platform. This timeline view, provided as a standard timeline trace file, also includes runnables dependencies and labels reading and writing information. This timeline view is provided as a trace file that could be visualized using the Vite[1] open source tool.

[1] http://vite.gforge.inria.fr

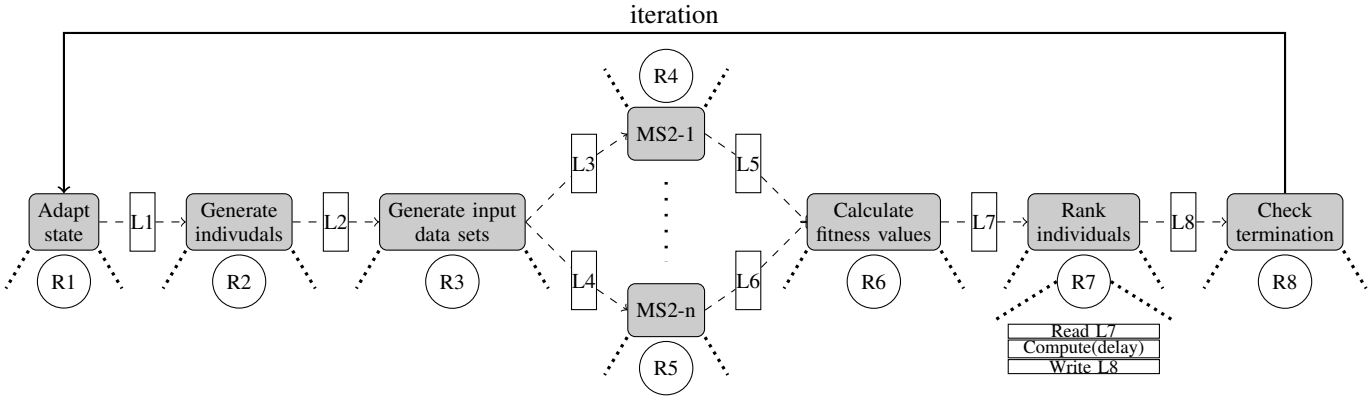


Fig. 4: The eScience case study application

VI. Experimental Results

We now show how the proposed workflow can be used to evaluate an application model and a cloud infrastructure provided by the High Performance Computing Center Stuttgart (HLRS) involved in the DreamCloud project. In all the following experiments, to illustrate the workflow, we use a mapping algorithm that randomly allocate the runnables and the labels on the underlying hardware infrastructure.

A. eScience Application

The application use case is based on an eScience application MS2 genetic algorithm [6] that performs molecular dynamic simulation. The goal of the simulation is to predict thermo-physical properties of condensed fluids which is required for the design and optimization of chemical engineering processes. The eScience workflow as an AMALTHEA model is depicted in Figure 4. MS2 is used to optimize the algorithm that tries to fit the parameters of an existing model to data collected through experiments. The algorithm is executed until a good enough fit is found, a time limit is hit or a fixed number of iterations have been executed. This in handled by the task called *Check termination*. In this application model, each AMALTHEA task contains a single runnable as depicted on Figure 4. As shown for the runnable $R7$, each of this runnable first reads input data from an input label, then perform a specific number of computations instructions and finally write output data to an output label. In all the following experiments, 32 *MS2* tasks are considered.

B. Cloud Infrastructure

The cloud infrastructure model we use in the following experiments, also provided by HLRS, is shown in Figure 5. It contains two nodes, named node 0 and node 1. Node 0 contains three hosts (node0,0 node0,1 and node0,2) while node 1 contains two hosts (node1,0 and node1,1). The configuration details for each host can be observed. The Frontend host acts as the master, and resource allocations are performed by this host. The bandwidth requirements for each interconnection link have also been provided.

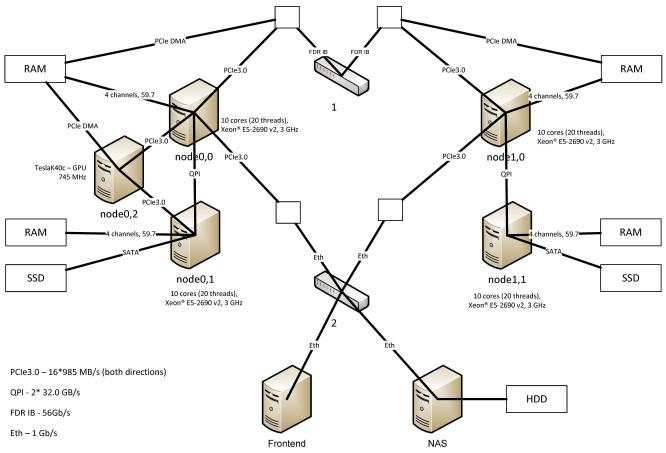


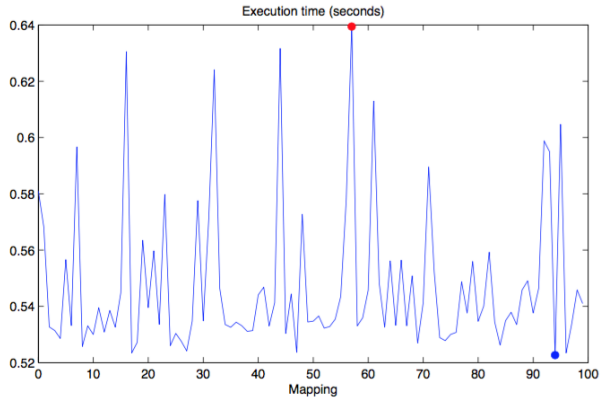
Fig. 5: The HLRS cloud platform

C. Results

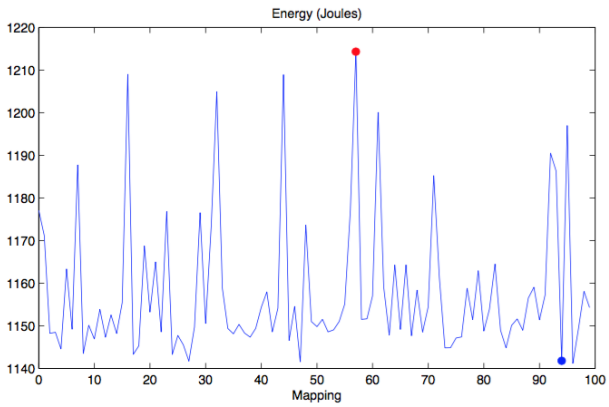
1) Preliminary mapping evaluation: We simulate the execution of the eScience application on the cloud infrastructure described previously. During each simulation instance, we trigger only one run of the application, i.e. the execution of the application completes as soon as the task named *Check termination* finishes. In a more general execution setting, the eScience application could possibly iterate several times until data fit the model (see Section VI-A).

On the other hand, we consider two variants of the cloud infrastructure: heterogeneous and homogeneous platform versions. The former is depicted in Figure 5, while the latter is obtained by removing from the Heterogeneous platform the host HOST_0_2, which is a GPU.

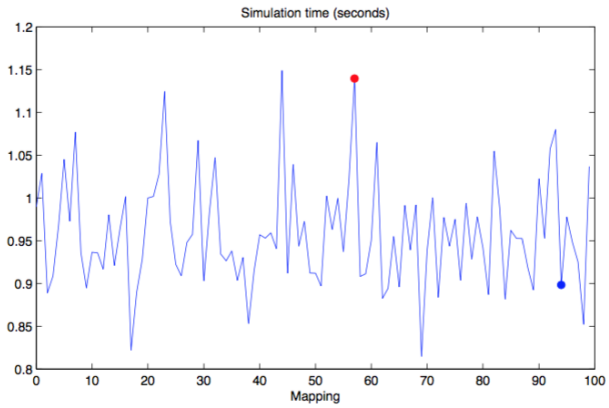
We conducted two sets of experiments. In the first experiment, we simulated the execution of 100 random static mappings of the eScience application on the homogeneous platform. The characterization of each mapping in terms of execution time, energy dissipation and simulation time is represented in Figure 6. The red and blue dots in the Figure, point to the worst and best mappings in terms of the execution time.



(a) Execution time

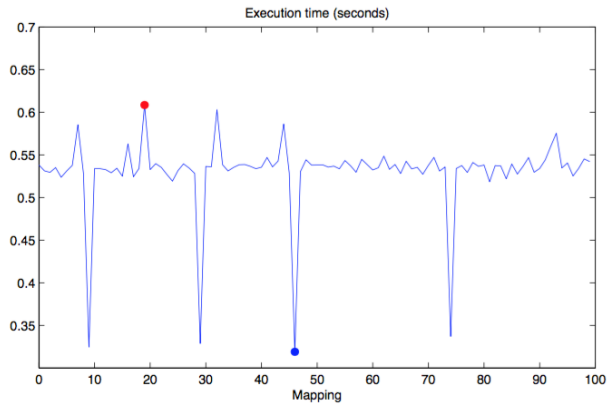


(b) Energy consumption

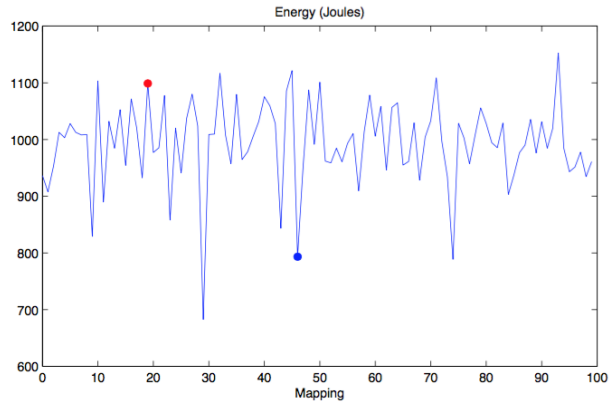


(c) Simulation time

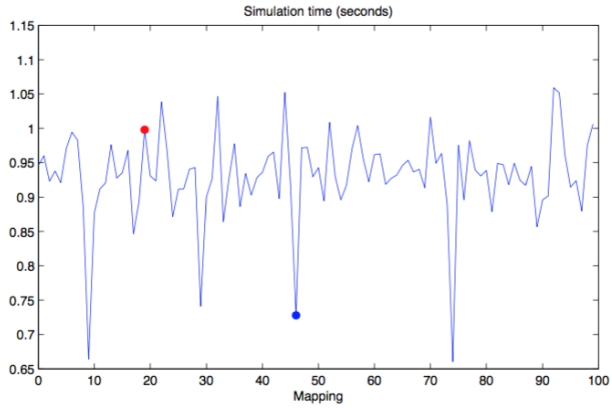
Fig. 6: Simulation on homogeneous platform



(a) Execution time



(b) Energy consumption

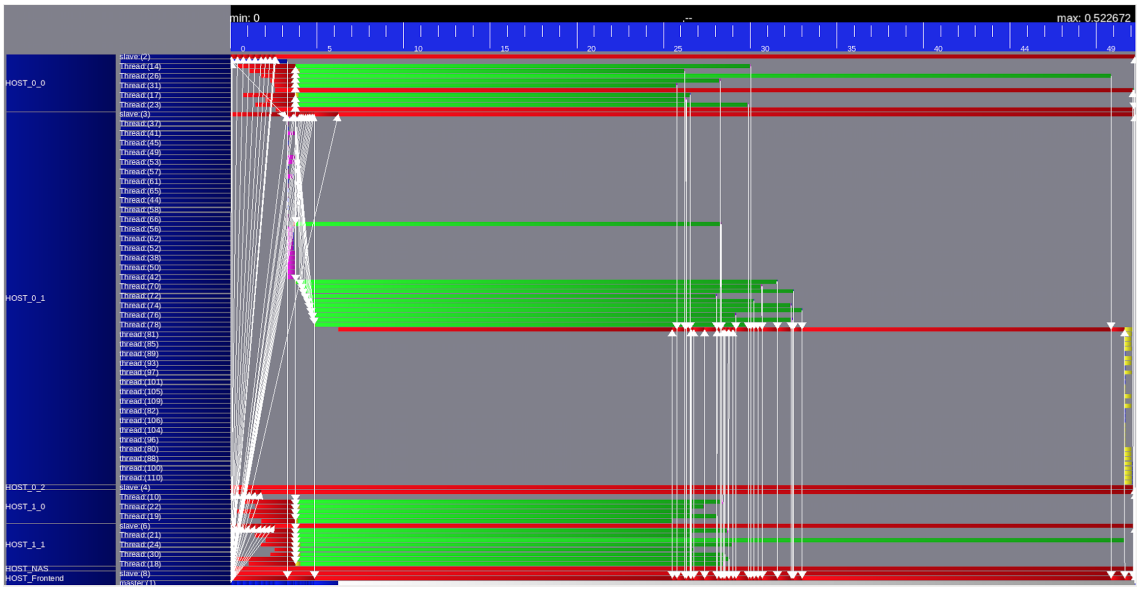


(c) Simulation time

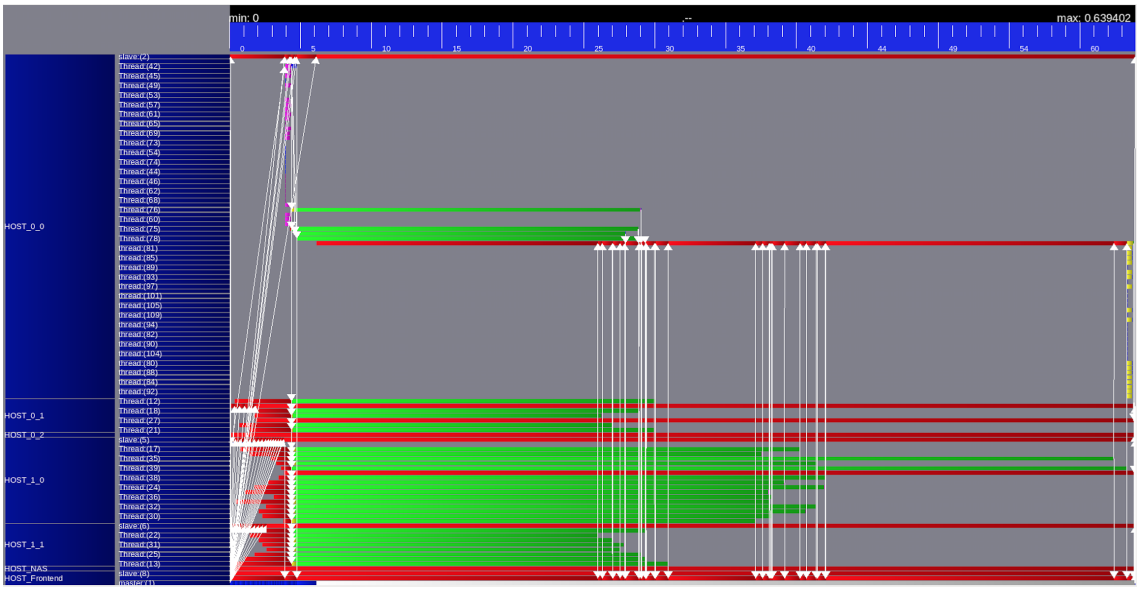
Fig. 7: Simulation on heterogeneous platform

The second experiment is similar to the first one with the difference that the heterogeneous platform is considered. The corresponding results are shown in Figure 7. Here, for the mappings with the shortest and longest execution time, the detailed temporal evolution of the execution is depicted in Figures 8(a) and 8(b) respectively. We can notice that in this heterogeneous case, the mapping with the shortest execution time may not be the most efficient one from the energy point of view. Indeed, the mapping number 28 may be a good candidate, as it provides an execution time close to the shortest one while using around 15 percent less of energy.

The execution time in Figure 8(a) is shorter than in Figure 8(b) because in the former case the tasks *MS2* are more evenly distributed on the hosts. Indeed, in Figure 8(b), the host HOST_0_1 executes 15 *MS2* tasks and the host HOST_0_0 only 4 tasks. Thus, in Figure 8(b) the host HOST_0_1 is overloaded compared to the other hosts. In that case, the host HOST_0_1 has to provide computational ressources to more



(a) Best mapping scenario



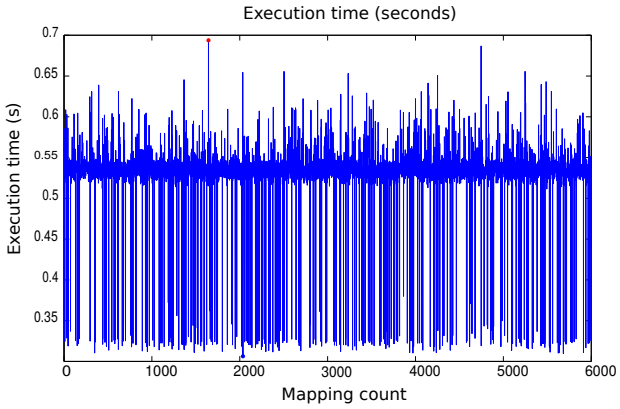
(b) Worst mapping scenario

Fig. 8: Zoom into the best and worst scenarios in simulating the execution on the homogeneous platform

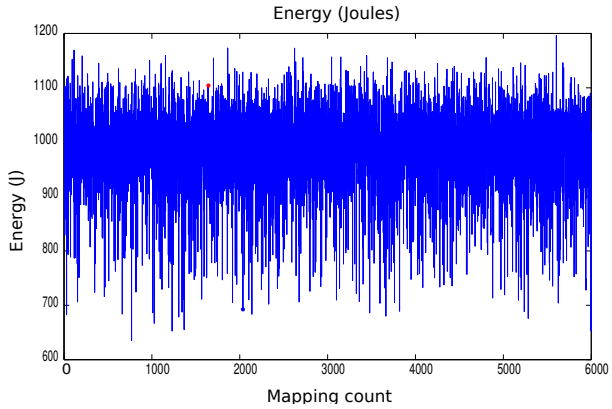
tasks than the other hosts. Consequently, the tasks on the host HOST_0_1 need more time to execute. Indeed, in Figure 8(b) the tasks executing on the host HOST_0_1 need 50% more time to finish compared to the tasks executing on the other hosts.

2) Large-scale mapping evaluation: We extend the previous evaluation to a larger number of mappings in order to get a better insight of the mapping impact on execution time. Figure 9 shows the evaluation of 6000 mappings of the eScience application on the heterogeneous platform. As for the experiments in the previous section, these mappings are randomly choosen.

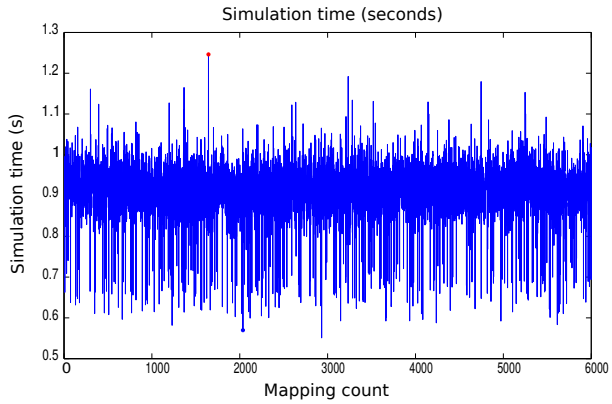
The shortest execution time is approximately 0.3064 seconds and the longest execution time is 0.6938 seconds. The reason for this difference is that in the first case, 34.4% of the *MS2* tasks are mapped on the host HOST_0_2 which is almost 100 times more powerful than the other hosts. In the second case, only 18.8% of the *MS2* tasks are mapped on the host HOST_0_2. Thus the execution for the first mapping finishes sooner than for the second mapping. Table I shows the execution times (E_time(s)) in seconds of different mappings. Each line corresponds to one mapping. For each mapping, the percentage of the *MS2* tasks that were mapped on the different hosts is specified. The mapping m_good is the best mapping



(a) Execution time



(b) Energy consumption



(c) Simulation time

Fig. 9: Large scale simulation on heterogeneous platform

in terms of the execution time among the 6000 mappings depicted in Figure 9. The m_bad is the worst mapping among those 6000 mappings. For the mapping m_best, all the tasks of the eScience application were mapped to the most powerful host HOST_0_2. In case of m_worst, all the tasks were mapped on the host HOST_0_1. When mapping all the tasks on the most powerful host, the execution time is almost 10 times shorter than if all the tasks are mapped on another host. This result was expected. Indeed, among the 5 hosts of our platform, 4 hosts have identical performances and 1 host provides 100 times more flops compared to any other host.

	H_0_0	H_0_1	H_0_2	H_1_0	H_1_1	E_time(s)
m_bad	25%	12.5%	34.4%	9.4%	18.7%	0.6938
m_good	53%	6.3%	18.8%	9.4%	12.5%	0.3064
m_worst	0	100%	0	0	0	1.0270
m_best	0	0	100%	0	0	0.1181

TABLE I: Execution times and *MS2* tasks mappings for different mappings

VII. Conclusion and Perspectives

This work proposes an integrated simulation workflow allowing to quickly assess the quality of mapping algorithms targeting cloud infrastructures and according to different criterion (i.e. energy efficiency or execution time). This simulation workflow takes as input applications specifications expressed in the AMALTHEA model chosen as the application model to be used in the context of the DreamCloud project. The specification of custom mappings is handled by implementing a clean interface whose main role is to specify where AMALTHEA entities should be located on the targeted cloud architecture.

We plan to extend the mapping design space exploration introduced in the last section with optimization techniques such as simulated annealing to allow end users to identify what are (and why) the best mapping strategies for particular applications. We will also improve the quality of the provided results by taking into account more architectural parameters.

Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under the DreamCloud Project: http://www.dreamcloud-project.org, grant agreement no. 611411. The authors would like also to thank the SimGrid team for the provided support.

References

[1] "AMALTHEA : Model Based Open Source Development Environment for Automotive Multi-Core Systems," http://www.amalthea-project.org/.

[2] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, Jun. 2014. [Online]. Available: http://hal.inria.fr/hal-01017319

[3] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities," *CoRR*, vol. abs/0907.4878, 2009. [Online]. Available: http://arxiv.org/abs/0907.4878

[4] A. Nez, J. Vzquez-Poletti, A. Caminero, G. Casta, J. Carretero, and I. Llorente, "icancloud: A flexible and scalable cloud infrastructure simulator," *Journal of Grid Computing*, vol. 10, no. 1, pp. 185–209, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10723-012-9208-5

[5] D. Kliazovich, P. Bouvry, Y. Audzevich, and S. Khan, "Greencloud: A packet-level simulator of energy-aware cloud computing data centers," in *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, Dec 2010, pp. 1–5.

[6] C. W. Glass, S. Reiser, G. Rutkai, S. Deublein, A. Kster, G. Guevara-Carrion, A. Wafai, M. Horsch, M. Bernreuther, T. Windmann, H. Hasse, and J. Vrabec, "A molecular simulation tool for thermodynamic properties, new version release," *Computer Physics Communications*, vol. 185, no. 12, pp. 3302 – 3306, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010465514002550