A Methodology to Build Decision Analysis Tools Applied to Distributed Reinforcement Learning

Cédric Prigent, Loïc Cudennec, Alexandru Costan, Gabriel Antoniu

A Methodology to Build Decision Analysis Tools Applied to Distributed Reinforcement Learning

Cédric Prigent*, Loïc Cudennec†, Alexandru Costan*, Gabriel Antoniu*
*University of Rennes, Inria, CNRS, IRISA - Rennes, France
{cedric.prigent, alexandru.costan, gabriel.antoniu}@irisa.fr
†DGA Maîtrise de l'Information, Rennes, France
{loic.cudennec}@intradef.gouv.fr

***Abstract*—As Artificial Intelligence-based applications become more and more complex, speeding up the learning phase (which is typically computation-intensive) becomes more and more necessary. Distributed machine learning (ML) appears adequate to address this problem. Unfortunately, ML also brings new development frameworks, methodologies and high-level programming languages that do not fit to the regular high-performance computing design flow. This paper introduces a methodology to build a decision making tool that allows ML experts to arbitrate between different frameworks and deployment configurations, in order to fulfill project objectives such as the accuracy of the resulting model, the computing speed or the energy consumption of the learning computation. The proposed methodology is applied to an industrial-grade case study in which reinforcement learning is used to train an autonomous steering model for a cargo airdrop system. Results are presented within a Pareto front that lets ML experts choose an appropriate solution, a framework and a deployment configuration, based on the current operational situation. While the proposed approach can effortlessly be applied to other machine learning problems, as for many decision making systems, the selected solutions involve a trade-off between several antagonist evaluation criteria and require experts from different domains to pick the most efficient solution from the short list. Nevertheless, this methodology speeds up the development process by clearly discarding, or, on the contrary, including combinations of frameworks and configurations, which has a significant impact for time and budget-constrained projects.**

***Index Terms*—Machine Learning, Distributed Computing, Decision Making**

I. Introduction

Nowadays, machine learning (ML) has turned into a critical approach to ensure the success of operational research projects and the design of complex systems. Beyond the classical engineering steps that lead to a piece of software, ML requires new methods, data processing and even dedicated hardware to run the ever-demanding computing needs of learning algorithms. This requires to commit hardware resources and provision enough time to run the computations. Therefore, there is a huge challenge to streamline machine learning within the development of operational projects in order to stick to the budget.

In this context, a significant effort is made by researchers to find an efficient trade-off between the accuracy of the results, the computing time and the energy consumption. One of the most promising approach is to distribute the learning among computing nodes in order to benefit from the aggregated capacity of processing elements and physical memories. Distributed computing is a well-studied problem since the late eighties in the field of high-performance computing (HPC) and has lead to major established methodologies (*e.g.,* workflows) and runtimes (*e.g.,* MPI). Unfortunately, ML workloads come with their own dedicated frameworks and some new programming models, usually at a higher level of abstraction than the regular HPC applications. The early days in machine learning were not based on distributed implementations and, as a consequence, most of the architectures and system knowledge required to deploy efficient HPC code have to be adapted, re-invented and transparently managed for ML experts.

Numerous frameworks have been recently proposed [1]–[4] to parallelize and distribute learning algorithms over GPUs and computing nodes. With the diversity of frameworks and the countless configuration possibilities ML experts have to face many choices, each of them leading to software design constraints, different scalability capabilities at deployment and even different accuracies when retrieving the results.

One of the main contribution of this paper is to propose **a methodology to facilitate decision making within ML projects.** These decisions occur from the early stage of the project and down to the operational deployment. This includes: choosing a distributed framework, choosing the learning algorithms, exploring the resource sizing and mapping. While all these steps cannot be fully automated, this methodology aims at providing decision making tools based on the evaluation and the comparison of different implementations and deployment possibilities.

A secondary contribution is based on an illustration with a real-life case study: **the proposed methodology is applied to an industrial-grade problem, and evaluated using state-of-art ML frameworks**. The motivating example is to provide autonomous steering for a cargo airdrop system used to deliver supplies in complex landing areas. We rely on a reinforcement learning (RL) project developed to build the autonomous steering inference model. A RL project consists in designing a reaction loop in which agents take actions in a specific environment and change their state based on a reward. In the context of the airdrop case study, the environment is a simulator used to compute the parachute physics along the

actions taken by the agent on the steering commands. The complexity of the computation comes from the need to explore a large number of scenarios and that the simulation is CPU-consuming. This is a classical RL problem and the returns on campaign are twofold: 1) this methodology enables to clearly highlight and discard specific combinations of frameworks and deployment parameters in a generic approach and 2) its application on a real use case allows to speed up the decision process within a running project development.

This paper is organized as follows: Section II gives some background and motivation to propose a methodology for decision making. In section III the methodology is presented from the design principles to the implementation ideas. Section IV describes the airdrop package delivery simulator and how the methodology can support this use case. In section V we report on the application of the methodology. A description of the experiments and the analysis of the collected results are given in section VI. Section VII gives some elements of discussion about the proposed approach. Finally, section VIII concludes this work and opens some perspectives.

II. Background and Motivation

A. Context: Reinforcement Learning

Reinforcement Learning (RL) is a Machine Learning paradigm in which an agent tries to improve a policy in order to maximize rewards by interacting with an environment. A RL problem may be formulated as a Markov Decision Process (S, A, T, R) where S is a set of states of the system, A is a set of actions that can be performed by the agent, T is a transition function mapping a state and an action to a new state, and R is a reward function mapping a state and an action to a reward. In other words, at each time-step the agent selects an action to perform in the environment, based on the last observation of the system. This action affects the environment and changes its internal state. Depending on the quality of the action performed in the current state, the environment returns a reward and a new observation of the system to the agent. A sequence of interactions of an agent with an environment is called an *episode* (made up of state-actions pairs ending with a terminal state).

Several algorithms have been proposed to solve RL problems. With *on-policy learning* agents learn from their current policy, while in *off-policy learning* they learn from experience generated by other policies. Other algorithms include: value-based methods such as Q-learning [5], policy gradient methods [6] such as Proximal Policy Optimization (PPO) [7], and experience replay [8].

Distributed techniques have proved to be efficient in the context of deep learning [9] by enabling the use of more resources and accelerating the training process. Multiple architectures have been proposed for distributed RL. A common approach is to separate acting from learning. With this configuration, several actors generate experiences in parallel while a central learner uses these experiences to learn an optimal policy. Examples of architecture include A3C [10], an asynchronous version of advantage actor-critic, IMPALA [11], a highly scalable agent introducing a new off-policy algorithm called V-trace, or Ape-X [12], a synchronous learner using a distributed replay buffer to sample experiences from actors.

Numerous tools have been developed in order to help users implementing these algorithms. OpenAI Baselines [13] and Stable Baselines [14] (a Torch version of OpenAI Baselines) provide high quality implementations of RL algorithms. TF-Agents [15] provides multiple tools to implement RL algorithms. Other tools include Ray RLlib [16], a highly scalable framework with simple APIs, Fiber [17], a framework for large-scale parallel computation, Acme [18], a research-oriented framework for distributed RL which aims to enable simple descriptions of RL agents, or TorchBeast [18], a Torch implementation of IMPALA.

B. Related Work: Optimizing software configuration

Several contributions not specifically related to ML, have been proposed to help in configuring and sizing an application, in search of a trade-off between multiple evaluation criteria. For example, mARGOt [19] is an autotuning framework that is able to dynamically adapt some parts of the application. However, this approach requires to refactor the domain code in order to insert probes and triggers, called *software knobs*. In this case study [20], the authors propose to evaluate several implementations of the same application to reduce the energy consumption. These code versions are automatically generated based on a domain-specific language (DSL) which requires to re-think the application. Another approach is to rely on the job scheduler to dynamically choose the deployment configuration [21] or to rely on a complete monitoring system that can tune the software and the hardware, as proposed in the GEOPM [22] project to manage power consumption in Exascale computing architectures. In this work [23], a tool is proposed to configure a software-distributed shared memory and evaluate different mappings over heterogeneous computing resources in order to find a trade-off between the computing time and the energy consumption. Similar works [24], [25] also rely on operational research and design space exploration tools to return a Pareto front highlighting the most efficient solutions.

While the presented works have similar objectives to ours, none of them tackle the problem of *choosing the appropriate framework for developing the application and running the computation*. Furthermore, distributed machine learning runtimes are quite new compared to established HPC runtimes and several subtleties arise when dealing with probabilities and non-deterministic distributed computing. This latter point motivates *the need to explore different frameworks to deal with the accuracy and the reproducibility of the learning*.

C. Problem Statement

The advances in ML lead to the emergence of multiple frameworks, algorithms and architectures. Choosing between them and, more importantly, ensuring a high confidence of the suitability of that choice is non trivial given the specificities of each tool and each use case. Clearly, there is distinction

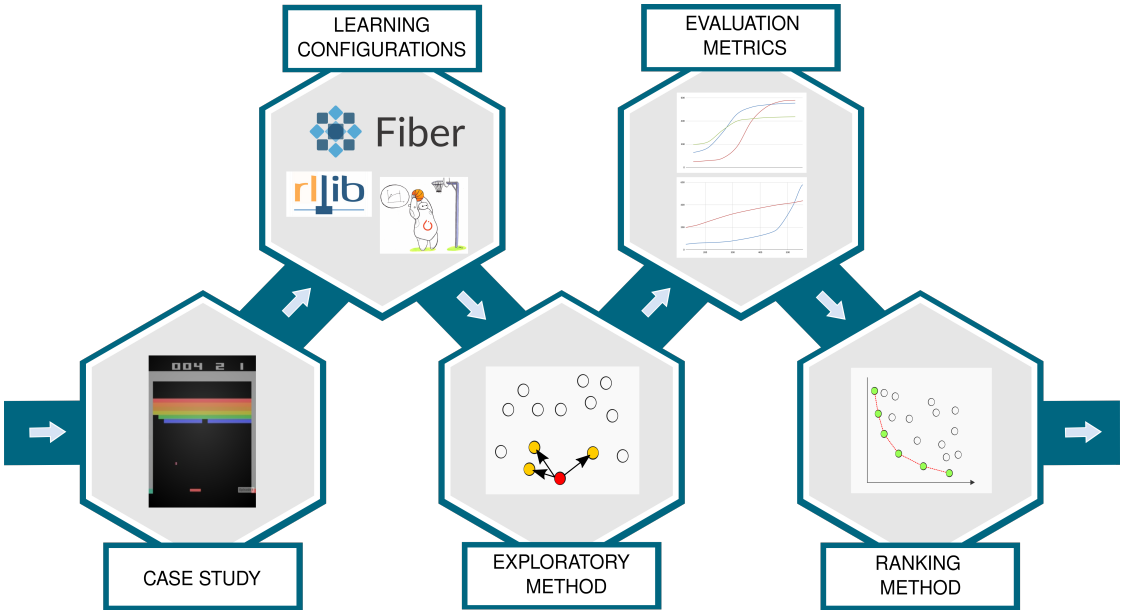


Fig. 1: Our Methodology for Decision Analysis

(in terms of application impact, performance and feasibility) between the choices made *after* the deployment, and the choices that must be made *before* implementing a solution.

In this paper, we focus on the latter (*e.g.,* choices such as frameworks and learning algorithms which have to be made before the implementation) since they are critical: they influence the applications for all the rest of their life-cycles and have impact on all dimensions of performance. Re-implementing a whole system from scratch because of some wrong initial framework choices is not suitable. A better approach is to test different sets of parameter on prototypes in order to make suitable choices.

III. A Methodology to Build Decision Analysis Tools

A. Design Principles

We briefly describe the design principles of our methodology, which are largely based on current state-of-the-art practices.

Stage-based approach. Inspired from the applications life cycle, our methodology breaks down to several steps corresponding to dedicated roles and responsibilities. Examples of such roles are, from a general perspective: collecting the details about the targeted system or use case; analyzing and evaluating those details to determine what and how needs to be changed; creating the plan of actions etc. These stages work together, communicate and collaborate with one another and exchange appropriate knowledge and data to provide a holistic control decision support functionality. The methodology should not prescribe the specific implementation choices for the internal structure of each stage. Instead, the methodology should organize the internal structure into a set of capabilities or functions. These are illustrated and described in the following sections.

Support for early and intelligent decision making. When the previously mentioned functions can be automated, an intelligent decision support mechanism is achieved. Given the complexity of deploying large-scale, real-life use cases on heterogeneous infrastructures, searching the ideal deployment configuration is challenging. This breaks down to configuring a myriad of system-specific parameters and reconciling many requirements or constraints. Bad choices may result in increased financial expenses during deployment and production phases, decreased processing efficiency and poor user experience. The decision analysis tools supported by this methodology should instead enable informed, intelligent and early choices in application development and deployment, with an impact on the rest of their life cycle.

User transparency. Identifying, evaluating and ranking the relevant configurations is non-trivial due to the multiple combination possibilities, as described in the next section. Using standard interfaces enables the underlying stages of the methodology to be composed together in a manner that is transparent to the user. The methodology would eventually work as a black box from the perspective of the user, who is freed from the burden of complex empirical explorations of the parameter space.

B. Methodology Overview

Our methodology guides the user to right choices *upstream*. It gives information about the software and the system scalability. This methodology is well suited for budget- and time-constrained projects. It consists in several steps, depicted in Figure 1 and described below.

a) The case study: The first step sets the initial problem. In RL, the case study defines the environment and all the

interactions that come with it. Examples include *gym* environments [26] such as Atari Breakout or Atari Pong in which the agent has to learn how to play games and obtain the best score.

b) Learning configurations: The second step defines the set of learning configurations one wants to study. A learning configuration is a set of parameters selected for a learning task. The parameters and the search space must be well defined, otherwise the risk is to get unsuitable results at the end of the process.

The parameters may be differentiated according to whether they are related to the algorithm configuration, the system configuration or the case study configuration. For instance, a parameter related to the algorithm might be the choice of the framework, the actual algorithm used or the learning rate, a parameter related to the system might be the topology of the network, the number of nodes or the nature of the nodes, and a parameter related to the case study could be the wind setting if any wind is simulated in the environment.

c) Exploratory method: If the search space is continuous or it is a large set containing many configurations, or if the expected experimentation time is long, then a better strategy than trying all the possibilities is to partially explore the search space. So the next step of our methodology is to choose an exploratory method.

This step sets the strategy for the exploration of the configuration pool. For instance, Random Search takes random combinations of hyper parameters to define configurations. By leveraging random combinations, the system might propose configurations which were not considered initially, and potentially find new, interesting solutions.

d) Evaluation metrics: The fourth step of the methodology defines the evaluation metrics. They are collected for each tested configuration and they provide different results throughout each learning process. These metrics sets the main objective of the study. Power consumption or bandwidth usage are examples of evaluation metrics.

e) Ranking method: The last step of the methodology consists in setting a ranking method. This method classifies the different solutions by building a hierarchy between them. It helps understanding the strengths and weaknesses of each solution. This ranking method might take the form of a graph or be textual. This last step provides the decision analysis tool to help the user in the decision making. Pareto front or sorted arrays are examples of ranking methods.

C. Implementation Ideas

In this subsection, we explore implementation ideas for our methodology.

The first approach, detailed in section V, is the one used for our experimental study. In this approach, the learning configurations are manually selected by picking parameters that seem relevant for the use case. The exploratory method is Random Search. Evaluation metrics such as Reward, Computation Time and Power Consumption are selected as important for the study. The chosen ranking method is Pareto Front, which provides a simple-to-interpret graph for the user.

Another interesting approach is to implement the methodology by using a hyperparameter optimization framework such as Optuna [27] or Hyperopt [28]. Optuna provides multiple optimization algorithms including sampling algorithms such as Random Search or Grid Search, and pruning algorithms which automatically stop unpromising trials. Hyperopt provides Random Search, Tree of Parzen Estimators (TPE) and Adaptive TPE as optimization algorithms. Both frameworks allow distributed hyperparameter search which is interesting when working with large clusters.

Algorithm 1 Execution of an episode of the Airdrop Package Delivery Simulator

Inputs:
$args \leftarrow wind,\ gust,\ gust\_probability,\ altitude\_limits$
Initialize:
$env \leftarrow gym.make('simulator', args)$
$agent \leftarrow init\_agent(env)$
$observation \leftarrow env.drop\_package()$
while $env.current\_altitude() > 0$ **do**
$action \leftarrow agent.select_{action}(observation)$
$observation \leftarrow env.compute_{dynamics}(action)$
end while
$return\ env.reward()$

IV. Motivating Use Case: Airdrop Package Delivery Simulator

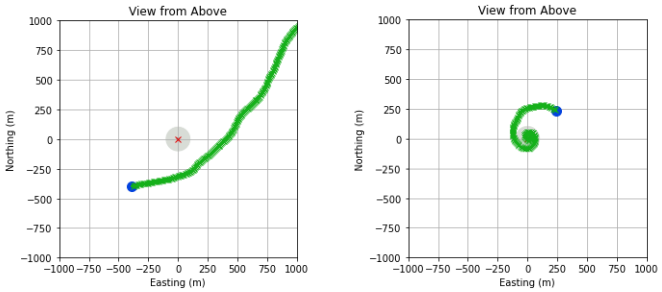
We illustrate the use of the proposed methodology with a real life use case: an Airdrop Package Delivery Simulator, that we describe in this section.

A. Application Framework

The goal of this simulator is to teach an autonomous agent to pilot a parachute canopy for a precision landing. The simulator is provided as a *gym* environment (*gym* is a library from openAI that provides the tools to develop customized RL environments).

Algorithm 1 gives a simplified description of the execution workflow of an episode, as follows.

1) The package is dropped from a random altitude included in a defined interval.
2) For each time-step until package landing:
   a) The simulator computes the dynamic of the parachute canopy and provides the agent with an observation of its new state. The observation includes rotation, position, orientation and velocity vectors of the airdrop package. A schematic representation of the package is provided in Figure 3.
   b) Taking into account the last observation, the agent selects a rotation direction for the parachute canopy.
3) The agent gets a reward depending on how close the package landed from the target point.



(a) Random policy (b) Trained PPO agent

Fig. 2: Airdrop package trajectory throughout an episode

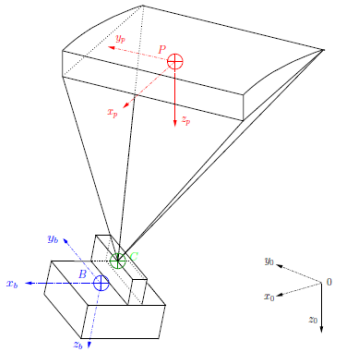


Fig. 3: Rotation axes of the airdrop package

.

Top views of trajectories taken by the package during two episodes using different policies are represented in Figure 2.

The simulator was developed by taking into account the dynamics of the parachute canopy, the drop altitude, the wind and the gusts of wind in order to simulate real conditions. Although it was designed to be used in the context of military operations, this use case is in fact representative for many applications including package delivery services by drone such as Amazon PrimeAir (*e.g.,* wind and other environmental factors), airdrop of rescue kits in emergency situations (*e.g.,* likely to use parachute). More generally, all simulations leveraging a decomposition of the environment at different granularities and the computation of some metrics (*e.g.,* temperature, humidity in climate contexts; plane speed, drag in aviation contexts etc.) share the same episode principles.

B. Configurable Environment

The simulator provides multiple parameters for the configuration of the environment (*i.e.,* **environment-specific** parameters):

- *Activation and deactivation of the wind*. This parameter may need to be activated depending on the environment context in which one wants the agent to learn.
- *Activation and deactivation of the gusts of wind*. This parameter adds some randomness to the environmental events.
- *The gust probability*. This parameter sets the occurrence probability of the gusts of wind.
- *The drop altitude limits*. The package will be dropped in this interval of altitudes.

The learning difficulty varies according to the settings of each of these parameters.

Another important parameter for the study is the **Runge-Kutta methods order**. Runge-Kutta methods are iterative methods that provide approximated solutions of differential equations. These methods are used by the simulator in order to compute the dynamic of the parachute canopy. As these methods are iterative, we may vary the order used for the computation of the dynamics. This will impact the accuracy of the observation but also the computation time. In general, if the Runge-Kutta order is lower, then the computation time will be lower but the accuracy of the solution will also be lower. Instead, if the Runge-Kutta order is higher, then the accuracy will be higher but the computation time will also be higher.

To illustrate this, we refer to the results in Table I, summarizing different performance metrics according to various parameter settings from our experimental evaluation. For instance, solutions 4 and 7 have the same parameter values except for the Runge-Kutta order. Solution 4 used 3rd order Runge-Kutta methods and solution 7 used 5th order Runge-Kutta methods. The results show that solution 4 with lower order Runge-Kutta methods gets lower reward and better computation time, which is coherent since the approximated solutions provided by 3rd order Runge-Kutta are less accurate but also need less time to be computed. On the other hand, solution 7 with higher order Runge-Kutta methods gets higher reward and takes more computation time.

C. How Does the Methodology Help Solving the Use Case?

The simulator was designed to be used on constrained and shared computing resources, hence an important objective is to rationalize the available resources while at the same time speeding-up the learning. Moreover, since the learning phase might get reproduced in the future, it would be better to set a good learning configuration beforehand. For instance, if the characteristics of the parachute canopy are changed, an agent has to be retrained from scratch. In this case, it would be better to already have a good configuration to learn from.

Our methodology provides the right tools for it. If the right parameter search space is set from the beginning and if the appropriate metrics are fixed, then potentially better trade-offs should be returned. For instance, power consumption is an important metric for constrained devices. With our methodology, we are able to find solutions that minimize this metric. Another example is the use of the computing platform by several operational projects at the same time, hence making the processing units (CPUs, GPUs) a *disputed* resource. In that case, our methodology allows to find solutions that best fit to the number of available resources at the moment. More detailed on the application of the methodology to this use case are given in the next section.

TABLE I: Configuration settings and results of the experimental evaluation.

	Configuration					Results		
	Environment-dependent	Environment-independent						
	Runge-Kutta order	*Framework*	*Algorithm*	*Nodes*	*CPUs per node*	*Reward*	*Computation time (minutes)*	*Power consumption (kJ)*
1	3	RLlib	PPO	2	2	-1.30	57	286
2	3	RLlib	PPO	2	4	-1.19	**46**	201
3	3	RLlib	SAC	1	2	-14.14	147	307
4	5	RLlib	PPO	1	4	-1.01	51	125
5	5	RLlib	PPO	2	4	-0.96	49	201
6	5	RLlib	SAC	1	4	-13.63	59	151
7	8	RLlib	PPO	1	4	-0.52	85	220
8	8	RLlib	PPO	2	4	-0.73	51	249
9	3	TF-Agents	SAC	1	4	-0.47	218	390
10	3	TF-Agents	PPO	1	2	-1.10	72	143
11	3	TF-Agents	PPO	1	4	-0.72	49	**120**
12	8	TF-Agents	SAC	1	4	-0.78	337	614
13	8	TF-Agents	PPO	1	4	-1.00	58	152
14	3	Stable Baselines	PPO	1	2	-0.47	85	172
15	3	Stable Baselines	SAC	1	4	-1.30	278	604
16	8	Stable Baselines	PPO	1	4	**-0.45**	65	176
17	8	Stable Baselines	SAC	1	4	-0.97	299	675
18	8	Stable Baselines	PPO	1	2	-0.48	108	222

V. Application of the Methodology

In this section, we present the configuration we selected to apply our methodology to the Airdrop Package Delivery Simulator. We further detail the choices made at each step of the methodology.

a) Configuration of the case study: Previous experiments on the simulator have shown that learning for 200,000 time-steps is sufficient to provide significant results, hence we chose this configuration. The drop altitude, which is not the main focus of our study, is left to its basic configuration, which is an interval between 30 and 1,000 units. Similarly, since the wind impact is not of concern for our study, we chose to disable it in order to get significant results faster.

b) Learning configurations: We have extracted 5 important parameters to study for our learning configurations:

- The **Runge-Kutta methods order**. This parameter is environment-specific. It allows to configure the computation accuracy for the dynamic of the parachute canopy. We selected 3rd, 5th and 8th orders which correspond to the values provided by the SciPy library [29].
- The **Framework** for implementing the algorithm. We selected 3 frameworks providing tools to implement RL. The first one, Ray RLlib [16], is a library for distributed RL built on top of the Ray framework [30]. The second framework, Stable Baselines [14], is another RL library which provides parallelized environments through vectorization. The third framework, TF-Agents [15], is a TensorFlow library for Contextual Bandits and RL.
- The **RL algorithm**. For this parameter we selected two popular algorithms, Proximal Policy Optimization (PPO) [7] and Soft Actor-Critic (SAC) [31].
- The **number of nodes**. As the number of available devices is limited to two machines, the number of nodes will vary between one and two nodes. Distributed training on 2 nodes is available with RLlib framework. TF-Agents and Stable-Baselines implementations parallelize the training on a single node, using multiple CPUs.
- The **number of CPU cores** used on each nodes. For this parameter, the number of CPU cores on each node is the same and it varies between 2 and 4 cores.

c) Exploratory method: For parameter search, we selected Random Search. This method takes random combination of parameters and has turned out to be effective for hyper-parameter optimization [32].

d) Evaluation metrics: We decided to study 3 metrics:

- **Reward** is the initial metric for any RL problem. In this case study, it defines how good the landing was (*i.e.,* if it was close enough to the target point).
- **Computation Time** measures the application running time, an important metric for time-constrained systems. The measure takes into account the whole process, from the launch time of the first actor until the last stop.
- **Power Consumption** is crucial for constrained devices using batteries. Since the process is mainly CPU intensive, we based the measurement on the CPU usage, computed as an equivalence with a consumption curve of the CPU.

e) Ranking method: We leverage Pareto Fronts, which provide an easy-to-understand graph and present the results as trade-offs between metrics.

VI. Experimental Evaluation

In this section we report on the experimental results of applying the methodology to the simulator use case. The objectives of this experimentation campaign is to assess whether:

- the methodology provides a decision analysis tool that answers the needs of the user;
- the different steps of the methodology are specific enough to provide suitable results.

To this purpose, the experiments are held on a small cluster of 2 nodes. These nodes are physical machines equipped with

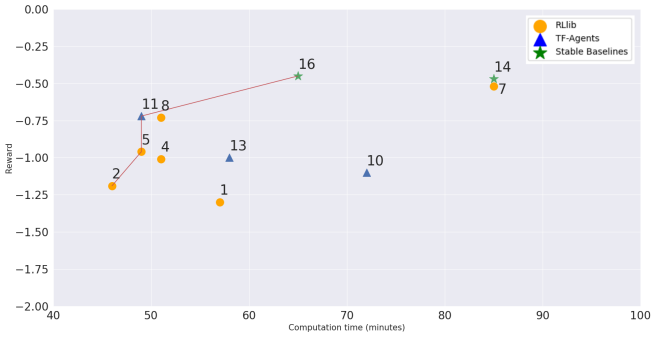


Fig. 4: Reward vs. Computation Time trade-off.

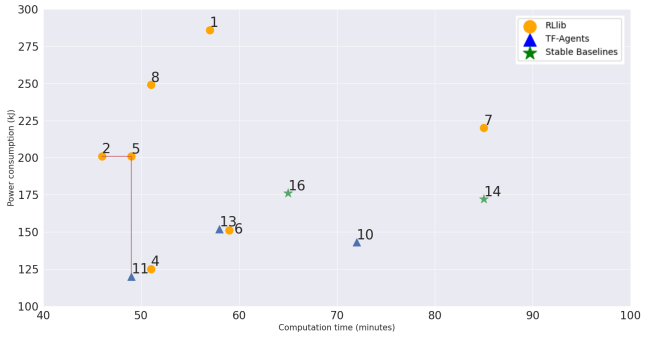


Fig. 5: Power Consumption vs. Computation Time trade-off.

Intel Xeon W-2102 CPU, Nvidia GeForce 1080 Ti GPU, Nvidia Quadro P400 GPU and 16GB of memory each. They are connected with a 1Gbps Ethernet Switch.

Three Pareto Fronts generated by applying our methodology are represented in Figures 4- 6, and discussed in the following subsections. For each of them, we refer to the configurations presented in Table I.

A. *Trade-off between Reward and Computation Time*

A first Pareto Front is represented on Figure 4. It highlights the best trade-offs between reward and computation time. The four non-dominated solutions are solutions 2, 5, 11 and 16.

First, solution 2 (corresponding to configuration 2 in Table I) is the fastest solution regarding the computation time with a total of 46 minutes. This result is the consequence of using all available resources and running the lowest Runge-Kutta order. As we can see on the generated graph, other solutions (4, 5, 8 and 11) provide close computation times. Again, these configurations are also using all available resources (except for solution 11 running on a single machine), with different order for Runge-Kutta methods. We highlight that Ray RLlib (configurations 2, 4, 5, 8) provides quite good solutions with respect to the computation time. Solutions 5 (RLlib) and 11 (TF-Agents) provide a good trade-off between the solution with the best computation time (configuration 2) and the solution with the best reward (configuration 16).

Finally, solution 16 (Stable Baselines) provides the best reward with -0.45. It is not as fast as the previous solutions, taking 65 minutes, but the results are still interesting since this configuration uses a single machine and leverages 8th order Runge-Kutta methods (with a strong impact on computation time and accuracy). Solutions 14 (Stable Baselines) and 7 (RLlib) also provide good rewards, -0.47 and -0.52 respectively, but take longer to compute, up to 85 minutes.

Note that all the presented solutions for this trade-off are using PPO. SAC solutions didn't perform well for these metrics and could not be displayed in the graph because of the scale and for the sake of clarity.

The best trade-offs from the tested configurations include 2 solutions from RLlib, 1 from TF-Agents and 1 from the Stable Baselines framework. All configurations are using the PPO algorithm. Regarding the RLlib setup, both configurations are distributed on 2 nodes, each node using 4 CPU cores. Runge-Kutta is the selective parameter, giving the best computation time when set to order 3 and the better reward when set to order 5. Configuration 11 using TF-Agents provides a good trade-off between reward and computation time. This solution using Runge-Kutta of order 3 prioritizing the computation time and is parallelized on one node using 4 CPU cores. Finally, configuration 16 using Stable Baselines provides the best reward with -0.45. This solution uses 4 CPU cores on one node and Runge-Kutta of order 8, therefore prioritizing the reward over the computation time.

B. *Power Consumption and Computation Time Trade-offs*

The second Pareto Front, represented on Figure 5, highlights the best trade-offs between Power Consumption and Computation Time. Solutions 2, 5 and 11 are highlighted as best trade-offs. Note that each of these solutions were already highlighted as good trade-offs in the previous experiment.

Solution 11 (TF-Agents), besides being a good trade-off between reward and computation time, is also the less power consuming solution with a total of 120 kJ. This is likely due to a cost-effective use of the CPUs by the framework, coupled with working on a single node.

Solution 4, a RLlib configuration on a single node, while not highlighted as one of the best trade-off by the Pareto Front, is quite close to solution 11 (125 kJ for 51 minutes, compared to 120 kJ for 49 minutes respectively). Moreover, solution 11 used 3rd order Runge-Kutta methods while solution 4 used 5th order Runge-Kutta methods. Hence, a RLlib configuration on a single node using 3rd order Runge-Kutta methods would potentially be closer-or-better than solution 11.

Solution 2, as seen on the previous Pareto Front, is the best solution with respect to the computation time. It takes less time to compute but provides less reward and consumes more power than solution 11.

Solution 5, although highlighted in this Pareto Front, is less interesting as this solution consumes as much as solution 2 (both consume 201 kJ) and takes as long to compute as solution 11 (both taking 49 minutes). This difference may be explained by the Runge-Kutta order (3rd order for solutions 2 and 11, and 5th order for solution 5).

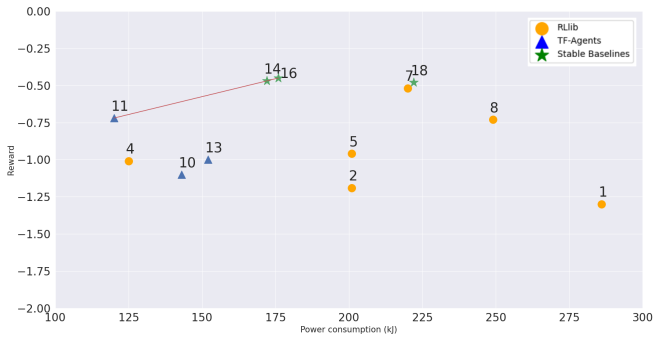


Fig. 6: Reward vs. Power Consumption trade-off.

No Stable Baselines solutions appear to offer a good trade-off between power consumption and computation time. All selected solutions use the PPO algorithm as well as all the 4 available CPUs cores. However, the number of nodes may vary: for this use case, and with a limited number of computing nodes, the intra-node parallelism is a more efficient choice than distributing the computation among the nodes.

C. Trade-off between Reward and Power Consumption

The last Pareto Front, represented on Figure 6, highlights the best trade-offs between reward and power consumption. Solutions 11, 14 and 16 are highlighted as non-dominated configurations.

Solution 11 (TF-Agents) and 16 (Stable Baselines) were already described and discussed in the previous experiments. Solution 14, implementing PPO with Stable Baselines, provides a good trade-off between reward and power consumption, that is close to solution 16. While it uses a low Runge-Kutta order (3), it still provides high accuracy -0.47 (in comparison with -0.45 for solution 16). Our analysis is that it is the consequence of using environments that are less vectorized, as one vectorized environment is used per CPU core and solution 14 uses only 2 CPU cores.

D. Impact of the Methodology

According to our experiments, each framework has proven efficient in different situations. It appears that RLlib is a good candidate to deal with the computation time, as it supports distributed training on multiple nodes. A second observation is that TF-Agents with PPO offers the lowest power consumption, as a consequence of running efficient parallel computations on a single node. Stable Baselines offers the best accuracy for the objective and obtained the best rewards. Regarding the learning algorithm, PPO provided accurate results with rather short computing times. On the contrary, SAC was inefficient and obtained poor results, either taking too much time for computation and consuming too much power, or failing in learning tasks and collecting low rewards.

As expected, when decreasing the Runge-Kutta order, the result accuracy decrease (as the reward is lower) and in the same time the computation time and the power consumption are enhanced. Instead, when increasing the Runge-Kutta order, the reward is higher but the computation takes longer, together with a higher power consumption. However, there are some exceptions: solutions 14 and 16 located at both extreme ranges of Runge-Kutta order result in almost the same performance for reward and power consumption. This shows that the number of vectorized environments has major consequences on the results.

For the system part, using all the available CPU cores speeds-up the training and seems to decrease the power consumption, as seen with solutions 10 and 11, while at the same time preserving the accuracy of the landing. On the RLlib side, using more nodes speeds-up the process but working on a single node seems to result in better reward. For instance, solutions 7 and 8 using the same configuration except for the number of nodes, do not provide the same reward. Solution 7 using a single node obtained a score of -0.52, while solution 8 working with two nodes obtained a score of -0.73. Distributing the learning to speed up the computation comes with uncertainties and a lack of reproducibility regarding the accuracy. This is quite similar to some simulation codes in which the order of computing events is not deterministic (due to the lack of strong causal dependencies), each ordering leading to a different -but hopefully converging- result.

To conclude this section, while some parameters appear at first sight easy to determine, the others are less so. For instance, using all the available resources (all nodes and all CPU cores) and setting the lowest Runge-Kutta order is an intuitive strategy to lower the computation time. However, when dealing with multiple objectives, this approach requires a more in-depth analysis. With our methodology, we built a decision analysis tool that helps users in finding a suitable learning configuration for optimizing accuracy, computation time and power consumption. Applied to the context of the Airdrop Package Delivery Simulator, multiple solutions with different configurations provide interesting trade-offs highlighted by Pareto Fronts.

VII. Discussion

Generality. We provide a tool to help users in their decisions to determine a suitable solution depending on the objectives set. This applies to some set of parameters, algorithmic- system- or environment- dependent. The environment parameters used in this paper are specific to our motivating use-case, however we claim that *our methodology is applicable to any other use case for optimizing algorithmic- and system-parameters.*

Scale of the experiments. In this paper, a first evaluation of our methodology was done on a small cluster by applying it to a real use case. It served as a successful proof of concept and it provided tools to better visualize the performance of each solution. One future direction we plan to explore is *to scale up the experiments, potentially using a large-scale distributed testbed* such as Grid'5000 [33] and an automatic experimentation framework like E2Clab [34].

Abstract vs. concrete methods. Our methodology provides the tools to help users in their decision making, however it remains rather abstract. Another interesting direction is *to provide more specific tools to define which exploratory method or which ranking method is the most adapted* depending on the use case.

VIII. Conclusions

Machine learning has become a critical step within the life-cycle management of many applications, carried out by the use of AI-based methods in every sector of activity prone to automation. The success of this ML step directly depends on using the appropriate development tools and an efficient sizing of the computation.

This paper introduces a methodology to help in making such decisions. This methodology allows to compare different ML runtimes, different algorithms and different strategies to collocate or to distribute the computation on remote nodes. This results in a decision making tool that highlights relevant combinations of runtimes and deployment parameters, offering a trade-off between multiple criteria such as the computing time and the energy consumption. This methodology has been applied to an industrial-grade case study for which it has demonstrated to be efficient in deciding between the accuracy of the computation, the computing time and the energy consumption.

As for many systems intended to relieve the developer from arduous and repetitive tasks, a fully automated toolchain remains illusory: the expertise of ML developers is still needed to implement some computing kernels and to arbitrate within a short list of solutions. While not providing a baked product on the table, this paper advocates for undertaking methodologies and tools that lead to a more efficient and wiser use of (super-)computing resources in the context of distributed machine learning at the age of green computing.

Acknowledgment

The authors would like to thank Guillaume Berthé and Samuel Hangouët (DGA MI) for their invaluable help in using the airdrop package delivery simulator and the application of reinforcement learning methods on this use case. Parts of this work were conducted while the first author was at DGA Maîtrise de l'Information and at the University of Western Brittany.

References

[1] A. Sergeev and M. D. Balso, "Horovod: fast and easy distributed deep learning in tensorflow," *CoRR*, vol. abs/1802.05799, 2018.

[2] M. Baines, S. Bhosale, V. Caggiano, N. Goyal, S. Goyal, M. Ott, B. Lefaudeux, V. Liptchinsky, M. Rabbat, S. Sheiffer, A. Sridhar, and M. Xu, "Fairscale: A general purpose modular pytorch library for high performance and large scale training." https://github.com/facebookresearch/fairscale, 2021.

[3] N. Shazeer, Y. Cheng, N. Parmar, D. Tran, A. Vaswani, P. Koanantakool, P. Hawkins, H. Lee, M. Hong, C. Young, R. Sepassi, and B. A. Hechtman, "Mesh-tensorflow: Deep learning for supercomputers," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada* (S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), pp. 10435–10444, 2018.

[4] J. J. Dai, Y. Wang, X. Qiu, D. Ding, Y. Zhang, Y. Wang, X. Jia, C. L. Zhang, Y. Wan, Z. Li, J. Wang, S. Huang, Z. Wu, Y. Wang, Y. Yang, B. She, D. Shi, Q. Lu, K. Huang, and G. Song, "Bigdl: A distributed deep learning framework for big data," in *Proceedings of the ACM Symposium on Cloud Computing, SoCC 2019, Santa Cruz, CA, USA, November 20-23, 2019*, pp. 50–60, ACM, 2019.

[5] C. J. C. H. Watkins and P. Dayan, "Technical note q-learning.," *Mach. Learn.*, vol. 8, pp. 279–292, 1992.

[6] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, (Cambridge, MA, USA), p. 1057–1063, MIT Press, 1999.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[9] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, "Large scale distributed deep networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, (Red Hook, NY, USA), p. 1223–1231, Curran Associates Inc., 2012.

[10] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016.

[11] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu, "IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures," *CoRR*, vol. abs/1802.01561, 2018.

[12] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, and D. Silver, "Distributed prioritized experience replay," *CoRR*, vol. abs/1803.00933, 2018.

[13] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines." https://github.com/openai/baselines, 2017.

[14] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines." https://github.com/hill-a/stable-baselines, 2018.

[15] S. Guadarrama, A. Korattikara, O. Ramirez, P. Castro, E. Holly, S. Fishman, K. Wang, E. Gonina, N. Wu, E. Kokiopoulou, L. Sbaiz, J. Smith, G. Bartók, J. Berent, C. Harris, V. Vanhoucke, and E. Brevdo, "TF-Agents: A library for reinforcement learning in tensorflow." https://github.com/tensorflow/agents, 2018.

[16] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, "RLlib: Abstractions for distributed reinforcement learning," in *Proceedings of the 35th International Conference on Machine Learning* (J. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, pp. 3053–3062, PMLR, 10–15 Jul 2018.

[17] J. Zhi, R. Wang, J. Clune, and K. O. Stanley, "Fiber: A platform for efficient development and distributed training for reinforcement learning and population-based methods," *CoRR*, vol. abs/2003.11164, 2020.

[18] M. Hoffman, B. Shahriari, J. Aslanides, G. Barth-Maron, F. Behbahani, T. Norman, A. Abdolmaleki, A. Cassirer, F. Yang, K. Baumli, S. Henderson, A. Novikov, S. G. Colmenarejo, S. Cabi, Ç. Gülçehre, T. L. Paine, A. Cowie, Z. Wang, B. Piot, and N. de Freitas, "Acme: A research framework for distributed reinforcement learning," *CoRR*, vol. abs/2006.00979, 2020.

[19] D. Gadioli, G. Palermo, and C. Silvano, "Application autotuning to support runtime adaptivity in multicore architectures," in *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS 2015, Samos, Greece, July 19-23, 2015* (D. Soudris and L. Carro, eds.), pp. 173–180, IEEE, 2015.

[20] I. Raïs, H. Coullon, L. Lefèvre, and C. Pérez, "Automatic energy efficient HPC programming: A case study," in *IEEE International Conference on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications, ISPA/IUCC/BDCloud/SocialCom/SustainCom 2018, Melbourne, Australia, December 11-13, 2018* (J. Chen and L. T. Yang, eds.), pp. 995–1002, IEEE, 2018.

[21] Y. Georgiou, E. Jeannot, G. Mercier, and A. Villiermet, "Topology-aware resource management for HPC applications," in *Proceedings of the 18th International Conference on Distributed Computing and Networking, Hyderabad, India, January 5-7, 2017*, p. 17, ACM, 2017.

[22] J. Eastep, S. Sylvester, C. Cantalupo, B. Geltz, F. Ardanaz, A. Al-Rawi, K. Livingston, F. Keceli, M. Maiterth, and S. Jana, "Global extensible open power manager: A vehicle for HPC community collaboration on co-designed energy management solutions," in *High Performance Computing - 32nd International Conference, ISC High Performance 2017, Frankfurt, Germany, June 18-22, 2017, Proceedings* (J. M. Kunkel, R. Yokota, P. Balaji, and D. E. Keyes, eds.), vol. 10266 of *Lecture Notes in Computer Science*, pp. 394–412, Springer, 2017.

[23] K. Trabelsi, L. Cudennec, and R. Bennour, "Application topology definition and tasks mapping for efficient use of heterogeneous resources," in *Euro-Par 2019: Parallel Processing Workshops - Euro-Par 2019 International Workshops, Göttingen, Germany, August 26-30, 2019, Revised Selected Papers* (U. Schwardmann, C. Boehme, D. B. Heras, V. Cardellini, E. Jeannot, A. Salis, C. Schifanella, R. R. Manumachu, D. Schwamborn, L. Ricci, O. Sangyoon, T. Gruber, L. Antonelli, and S. L. Scott, eds.), vol. 11997 of *Lecture Notes in Computer Science*, pp. 258–269, Springer, 2019.

[24] R. Friese, B. Khemka, A. A. Maciejewski, H. J. Siegel, G. A. Koenig, S. Powers, M. Hilton, J. Rambharos, G. Okonski, and S. W. Poole, "An analysis framework for investigating the trade-offs between system performance and energy consumption in a heterogeneous computing environment," in *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, Cambridge, MA, USA, May 20-24, 2013*, pp. 19–30, IEEE, 2013.

[25] L. Zaourar, M. A. Aba, D. Briand, and J. Philippe, "Task management on fully heterogeneous micro-server system: Modeling and resolution strategies," *Concurr. Comput. Pract. Exp.*, vol. 30, no. 23, 2018.

[26] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[27] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[28] J. Bergstra, D. Yamins, and D. D. Cox, "Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms," 2013.

[29] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[30] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica, "Ray: A distributed framework for emerging ai applications," 2018.

[31] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *CoRR*, vol. abs/1801.01290, 2018.

[32] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. 10, pp. 281–305, 2012.

[33] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec, "Adding virtualization capabilities to the Grid'5000 testbed," in *Cloud Computing and Services Science* (I. I. Ivanov, M. van Sinderen, F. Leymann, and T. Shan, eds.), vol. 367 of *Communications in Computer and Information Science*, pp. 3–20, Springer International Publishing, 2013.

[34] D. Rosendo, P. Silva, M. Simonin, A. Costan, and G. Antoniu, "E2clab: Exploring the computing continuum through repeatable, replicable and reproducible edge-to-cloud experiments," in *IEEE International Conference on Cluster Computing, CLUSTER 2020, Kobe, Japan, September 14-17, 2020*, pp. 176–186, IEEE, 2020.