Resource Allocation and Scheduling Problems in Computing Continua for Artificial Intelligence Applications

Federica Filippini

Abstract The problem of optimizing the execution of Artificial Intelligence (AI) and Deep Learning (DL) applications in the Computing Continuum gained remarkable popularity in recent years, due to both the widespread adoption of AI in real-life scenarios and the challenging environment introduced by a distributed Edge-to-Cloud paradigm. We tackled the resource selection, scheduling and placement problem both from a design-time and runtime perspective, considering, on one hand, AI inference applications characterized by complex workflows with multiple heterogeneous components and, on the other hand, resource-demanding DL training jobs executed on public or private GPU-accelerated clusters.

1 Introduction

These years witness two interconnected phenomena: on one hand, the increasing pervasiveness of Deep Learning (DL) and Artificial Intelligence (AI), widely adopted in largely heterogeneous fields (e.g., personalized healthcare, industrial applications, etc.); on the other hand, an accelerated migration towards distributed computing. Latency and privacy constraints limit the feasibility of processing in the public Cloud the large data volumes generated by Internet of Things devices, favouring on-premises resources or Edge devices. However, large datacenters featuring powerful, often GPU-accelerated, Virtual Machines (VMs) are still crucial for resource-demanding tasks, which benefit from the possibility of accessing ideally unlimited computational and storage resources according to pay-to-go pricing models. The effective management of such a complex environment, featuring heterogeneous Edge-to-Cloud resources interconnected in a so-called Computing Continuum, poses great challenges [6]. The execution of AI inference workflows including diverse components needs to be optimized at design time, selecting the most appropriate resources from the available computational layers, and then adapted at runtime, since fluctuating workload and environment conditions may determine sub-optimal scenarios

F. Filippini (✉)
Politecnico di Milano, Milan, Italy
e-mail: federica.filippini@polimi.it

S. Garatti (ed.), *Special Topics in Information Technology*,
PoliMI SpringerBriefs, https://doi.org/10.1007/978-3-031-80268-3_4

where resources are saturated or under-utilized. Similarly, DL training applications executed in private or public Cloud clusters are to be effectively scheduled on GPU-accelerated nodes to minimize the expected costs and meet the user-imposed due dates. This work addresses these three problems from different perspectives, proposing mathematical models and heuristic or Reinforcement Learning (RL)-based methods for the efficient and effective management of AI inference applications, both at design time and runtime, and of DL training jobs. Furthermore, it discusses techniques to develop analytical and Machine Learning (ML)-based performance models, crucial to accurately predict the applications response times on heterogeneous resources. The proposed tools are validated in both simulated and prototype environments, proving their effectiveness and applicability to practical scenarios.

In the following, Sect. 2 overviews our work on performance modeling based on ML; Sect. 3 presents the framework we propose for the design-time optimization of AI workflows. Sections 4 and 5 address the runtime resource management in the Computing Continuum, focusing on AI inference applications and DL training jobs, respectively. Finally, Sect. 6 draws conclusions and ideas for future works.

2 Performance Models

Predicting the execution times of computing tasks (e.g., a DL training job or a stage of a complex inference pipeline) executed on different resources is crucial to support their optimal placement, since accurate estimates allow to correctly forecast their computational requirements and the expected costs related to the Computing Continuum resources usage [17]. We addressed this problem both through analytical, white-box methods based on queuing theory, which guarantee a fast computation and the possibility of considering components interference due to co-location, and by developing black-box models based on ML. Once trained on profiling data, these allow to predict the response times of tasks in unforeseen conditions, interpolating or extrapolating over the available features, when information about the demanding times or the components throughput are not easily available.

In the next sections, we firstly present OSCAR-P [13], a tool that seamlessly supports the execution, profiling and performance modeling for serverless AI workflows running in the Computing Continuum. Then, we discuss the development of ML-based models for Distributed Deep Learning (DDL) training applications [9].

2.1 OSCAR-P: Serverless Applications Performance

OSCAR-P is an auto-profiling tool we built around OSCAR [21], a state-of-the-art, open-source platform that supports the execution of serverless applications in Cloud and Edge environments. OSCAR-P allows to automatically test the application under study on heterogeneous resources and in different hardware configurations, gathering

information on the components response times. Individual ML-based performance models for each component-resource pair are built through *aMLLibrary*, an open-source tool for the automatic generation of ML-based regression models [15]. They are combined to predict the response time of the entire workflows, thus supporting both their design-time optimization [23] and runtime management [18].

The models generated during the respective experimental validation, which considered several multi-component AI workflows deployed in the Computing Continuum, achieved good performance, with a Mean Absolute Percentage Error (MAPE) lower than 10% on interpolation and 20% on extrapolation in the analyzed scenarios.

2.2 Distributed Deep Learning on Ray

In addition to the performance of AI inference pipelines, we worked on estimating the execution time of DDL training applications, focusing on jobs developed and executed on Ray clusters. Ray [20] is an increasingly popular open-source framework that supports the scaling of general-purpose and AI workflows. The DDL paradigm [4] was developed to accelerate the training of DL jobs by subdividing it in multiple tasks executed in parallel. To the best of our knowledge, no previous studies on the performance of DDL applications executed on Ray clusters are available.

We considered two Convolutional Neural Network (CNN) models for image classification as target training jobs. The limited size of the target networks allowed to evaluate both GPU and CPU scalability; moreover, the considered scenario is particularly challenging for the development of accurate ML-based performance models, since the profiling data collected during the training are more noisy due to the communications overhead. Jobs were deployed on a cluster including several worker nodes hosted on AWS [1]. We profiled jobs collecting the execution time of several stages (from a single training epoch to the loop including inter-workers communications) and exploited aMLLibrary to train ML models using as base features the number of used resources and the dataset size. Our models perform well for both interpolation and extrapolation, achieving a MAPE between 3 and 15%.

3 Design-Time Optimization

The deployment and execution of AI applications in the Computing Continuum is challenging, and careful Resource Selection and Component Placement (RS-CP) decisions are crucial to find a mapping between the applications components and the infrastructure that minimizes the expected costs while meeting hardware, network, privacy and QoS requirements [6] (see Fig. 1). At design time, according to the expected value of input workload, application owners need to: (i) select the best resource among the available alternatives in each computational layer; (ii) identify the optimal deployment for Deep Neural Network (DNN) components that

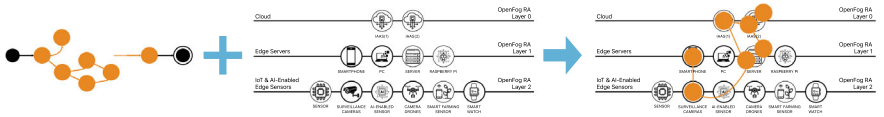

Fig. 1 The placement problem: mapping an application on the infrastructure

can be partitioned at different layers [16] to facilitate offloading; (iii) determine the minimum-cost placement of all component partitions on the selected resources, under the users-imposed constraints. A good-quality RS-CP solution is crucial also to facilitate runtime management tools that adapt the design-time choices to varying workload conditions (see Sect. 4). Indeed, a wrong resource selection at design-time may prevent such methods from suitably identifying a feasible reconfiguration (particularly for Edge devices, which have limited capacity and usually become the performance bottleneck if the input workload increases).

To the best of our knowledge, our work represents one of the first attempts to effectively address the issue of resource contention, by adopting queuing theory to model application components response times [24], while existing design-time tools usually focus on the execution of a single application instance. Similarly, considering multiple candidate DNN partitioning points for our application components and identifying the optimal deployment as part of the design-time optimization represents a novel contribution with respect to the existing literature [23].

We proposed a Mixed-Integer Non-Linear Programming (MINLP) formulation for the design-time RS-CP problem in the Computing Continuum, with objective:

$$\min \left(C_E + C_C + C_F + C_T \right), \tag{1}$$

where C_E is the cost of Edge devices, including the amortized investment cost and the expected management costs over the single run of the target application; C_C is the usage cost of Cloud VMs, which depends on the chosen provider; C_F denotes the cost of FaaS instances, which depends on the memory size, the functions duration and the total number of invocations; C_T is a state transition cost that some serverless providers charge for the message passing and coordination between successive functions. This is subject to hardware, memory, and QoS constraints imposed on single application components or complex execution paths, which make the problem $\mathcal{NP}$-hard.

We developed SPACE4AI-D (System Performance and Cost Evaluation on Cloud for AI applications Design), a tool that leverages Random Search and several heuristic methods (i.e., Local Search, Tabu Search, Simulated Annealing and Genetic Algorithms), to determine good-quality solutions in a reasonable time. SPACE4AI-D yields significant cost reductions with respect to the state of the art, between 27.6% and 58% on average against a tool proposed in [19], and the observed deviation between real and predicted costs in a prototype environment is lower than 13%.

4 Runtime Management for Inference Applications

In practical applications, the design-time optimization discussed in Sect. 3 has to be continuously reevaluated and, possibly, updated: the expected application workload is often subject to fluctuations due to, e.g., variations in the generated data volumes [23]. Therefore, the initial placement may become unfeasible or lead to resource under-utilization, dictating the need of effective monitoring and adaptation mechanisms. The most relevant difference between design-time and runtime optimization methods is that, while a design-time tool is allowed to take as much time as needed (up to several minutes) to find the initial production deployment (i.e., the initial placement), a runtime tool must provide a feasible reconfiguration in few seconds at most, due to the online running application. Therefore, runtime tools must be designed according to appropriate cost-reactivity trade-offs while tuning the solver algorithms, as well as implemented in a very efficient way to reduce the execution time. Among the families of approaches that can be leveraged to develop runtime optimization frameworks, we focused on heuristic-based methods (see Sect. 4.1) and RL-based methods (see Sect. 4.2). The former guarantee good solution quality and fast execution times, but they require a deep knowledge of the problem characteristics and sometimes restrictive assumptions on the model properties to guarantee effective implementations. On the other hand, RL-based methods can successfully adapt overtime to variations in, e.g., the workload model or the application computing requirements by relying only on observations extracted from the environment [3]. Therefore, they are often more flexible than methods based on analytical optimization models, but usually require a long time to learn an effective policy. During the training, an agent needs to explore all the possible actions, possibly incurring in severe constraints violations or overspending. Our work aims to bring together the best of the two worlds: the RL-based framework we propose accelerates the initial training by leveraging the prior knowledge coming from the design-time analysis.

4.1 SPACE4AI-R

SPACE4AI-R (System Performance and Cost Evaluation on Cloud for AI applications Runtime) is the runtime counterpart of SPACE4AI-D, and exploits an efficient Random Search combined with a Stochastic Local Search algorithm to identify a suitable reconfiguration of the running placement coping with the workload fluctuations [10]. In particular, reconfiguration decisions cannot change the selected resource type on computational layers of physical resources, while they include: (i) identifying which resource is best at each virtual and currently unused computational layer or at the FaaS layer, (ii) scaling in/out resources at the physical layers or at currently-used virtual layers, (iii) selecting the optimal DNN deployment for each component, and (iv) assigning each partition to a compatible resource.

The experimental validation proved the advantages of a dynamic system reconfiguration over a static approach where the design-time solution is kept fixed regardless the variations in the input workload, achieving an average cost reduction between 2 and 36% in the analyzed scenarios. Moreover, SPACE4AI-R yields a speed-up close to 100× with respect to its design-time counterpart, tackling problem instances involving up to 15 AI application components in less than 1.5 s on average. Finally, the comparison with a state-of-the-art method that performs scaling actions with the goal of keeping the resources utilization within a predefined interval [26] highlights cost reductions between 10% and 40% and a better capability of avoiding response-time constraints violations.

4.2 FIGARO

FIGARO (reinForcement learnInG mAnagement acRoss the computing cOntinuum) is an RL-based approach that automatically adapts the current deployment to varying and possibly unforeseen conditions [7]. To limit the negative impact of the initial exploratory phase, FIGARO undergoes an initial training that exploits imitation learning to develop a policy close to the one exploited by SPACE4AI-R. This is further trained via interactions with a simulated environment, leveraging a Deep Q-Learning approach to develop a good-quality scaling policy.

A preliminary evaluation of the FIGARO framework proved the effectiveness of our RL Agent in a simplified scenario involving only scaling actions. In particular, the initial offline training stage that exploits the design-time knowledge reduces the cumulative number and entity of response time constraints violations by 2–4 orders of magnitude, and by 25% the number of required training iterations.

5 Runtime Management for Training Applications

Training DL applications is a huge challenge these days. Jobs are very resource-demanding, despite the significant performance benefits guaranteed by GPU acceleration [25]. Moreover, high-performance servers are characterized by considerable costs, both to acquire and maintain resources (due to, e.g., energy consumption and cooling [5]) and in the public Cloud. Consequently, effectively tackling the Resource Selection and Job Scheduling (RS-JS) problem is crucial both for developers that rely on public services and for Cloud Service Providers (CSP). One one hand, selecting the most suitable resources to co-locate different DL training workloads and determining efficient schedules allows Cloud users to minimize the execution costs of their applications (see Sect. 5.1). On the other hand, CSPs need efficient methods to select resources for the execution of each job to minimize the energy consumption costs while meeting the applications due dates (see Sect. 5.2).

5.1 RS-JS Problem from a User's Perspective

We envisioned a scenario including a multi-node cluster, where servers can be provisioned on-demand and configured with heterogeneous VMs characterized by a possibly different number of GPUs, according to the provider catalog (see, e.g., [1]). The submitted jobs can be executed concurrently on the assigned nodes; all computational resources are shared except for GPUs, which are dedicated to running single jobs. Determining the optimal solution means addressing two intertwined subproblems: (i) the RS problem consists in identifying, for each submitted job, the optimal VM type and number of GPUs to minimize its execution cost while meeting the prescribed due date; (ii) the JS problem involves determining which jobs to run if the available resources are not enough to execute all of them concurrently, and assigning them to the cluster nodes effectively partitioning the selected resources. We tackle these problems jointly and in an online setting, without any information about the number and characteristics of jobs that will be submitted in the future.

We developed a MILP formulation and a heuristic algorithm based on Randomized Greedy and Path Relinking [14]. The MILP objective reads:

$$\min \sum_{j} \left(C_j^D + C_j^P\right) + \sum_{jn} C_{jn}^E + \sum_{n} C_n^{idle}. \tag{2}$$

For each job j, we pay a penalty C_j^D if it exceeds the due date, and an additional cost C_j^P if its execution is postponed: while our scheduler supports pre-emption, this term incentivizes a prompt execution to minimize starvation. Moreover, we consider the execution cost C_{jn}^E paid if job j is deployed on node n, and a final penalty C_n^{idle} that, for each cluster node n, reduces the risk of leaving idle resources.

Our heuristic scheduler proceeds by sorting jobs according to the *pressure*, i.e., the distance from the due date, so that the most critical receive resources as first. The optimal configuration for each job is assumed to be: (i) the cheapest configuration such that it is executed before its due date, if possible, or (ii) the fastest available configuration if the due date cannot be met with any setup. Jobs-to-nodes assignments are finally performed according to a best-fit approach to limit idle resources.

The experimental validation compares Path Relinking, simpler heuristic variants based on Randomized Greedy, a previously-proposed method based on a hierarchical problem decomposition [8], and a dynamic programming-based method adapted from the literature [22]. Even in the most complex settings, we achieve an average cost reduction between 23 and 97% and tackle instances including up to 100 cluster nodes and 450 concurrent jobs in less than 7 s. The deviation between real and predicted costs in a prototype environment is lower than 5%.

5.2 RS-JS Problem from a Cloud Provider Perspective

In this scenario, we assume that jobs may conclude the training before the predefined number of epochs, since DL developers usually rely on termination criteria based on, e.g., the accuracy level; therefore, we consider stochastic execution profiles. Guided by a solution model proposed in [2], which identifies the optimal schedule for a single job exploiting information about the probability distribution of the epochs required to complete the training and the power consumption model, we proposed a MINLP formulation for the minimization of the expected energy costs and designed a STochastic Scheduler (STS) to select the best GPU type and amount of resources to execute each application, incorporating GPU sharing [11]. STS exploits the assumption that energy costs increase linearly with the number of used GPUs, which can be observed in practice from Cloud providers pricing models (see, e.g., [1]), and designs the resource profile for each job starting the execution with a low-power configuration, which is usually less expensive, and accelerating the training process by progressively increase the resources as the due date approaches.

STS outperforms both state-of-the-art approaches [22] and an adapted version of our Randomized Greedy algorithm [12], achieving cost reductions between 38 and 80% on average. Finally, the adoption of GPU sharing guarantees additional advantages, further reducing the STS costs by 19–29% on average.

6 Conclusions

The approaches proposed in this work focus on optimizing the execution of AI and DL applications in the Computing Continuum. We initially worked on the automatic development of ML-based models for performance prediction, and then tackled the resource selection, scheduling and placement problems both from a design-time and runtime perspective. The frameworks we developed target, on one hand, AI inference applications characterized by complex workflows with multiple heterogeneous components and, on the other hand, resource-demanding DL training jobs executed on public or private GPU-accelerated clusters.

All the proposed approaches proved their effectiveness and applicability to practical scenarios, achieving significant cost reductions against the state of the art with minimal deviations between real and predicted costs when evaluated in prototype environments. Future works will further investigate the potential of RL for the runtime applications management, exploring federated and multi-agent approaches.

References

1. Amazon EC2 Instance Types (2023). https://aws.amazon.com/ec2/instance-types. Accessed 11 Jul. 2024
2. Anselmi, J., Gaujal, B.: Energy optimal activation of processors for the execution of a single task with unknown size. In: MASCOTS, pp. 65–72 (2022). https://doi.org/10.1109/MASCOTS56607.2022.00017
3. Barrett, E., Howley, E., et al.: Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. Concurr. Comput.: Pract. Exp. **25**(12), 1656–1674 (2013). https://doi.org/10.1002/cpe.2864
4. Ben-Nun, T., Hoefler, T.: Demystifying parallel and distributed deep learning: an in-depth concurrency analysis. ACM Comput. Surv. **52**(4) (2019). https://doi.org/10.1145/3320060
5. Bharany, S., Sharma, S., et al.: A systematic survey on energy-efficient techniques in sustainable cloud computing. Sustainability **14**(10) (2022). https://doi.org/10.3390/su14106256
6. Costa, B., Bachiega, J., et al.: Orchestration in fog computing: a comprehensive survey. ACM Comput. Surv. **55**(2) (2022). https://doi.org/10.1145/3486221
7. Filippini, F., Cavadini, R., et al.: FIGARO: reinForcement learnInG mAnagement acRoss computing continua. In: IEEE/ACM UCC Proceedings (2023). https://doi.org/10.1145/3603166.3632565
8. Filippini, F., Lattuada, M., et al.: Hierarchical scheduling in on-demand GPU-as-a-service systems. In: SYNASC, pp. 125–132. IEEE (2020). https://doi.org/10.1109/SYNASC51798.2020.00030
9. Filippini, F., Lublinsky, B., et al.: Performance models for distributed deep learning training jobs on ray. In: SEAA, pp. 30–35 (2023). https://doi.org/10.1109/SEAA60479.2023.00014
10. Filippini, F., Sedghani, H., Ardagna, D.: SPACE4AI-R: a runtime management tool for AI applications component placement and resource scaling in computing continua. In: IEEE/ACM UCC Proceedings (2023). https://doi.org/10.1145/3603166.3632560
11. Filippini, F., Anselmi, J., et al.: A stochastic approach for scheduling AI training jobs in GPU-based systems. IEEE Trans. Cloud Comput. **12**(01), 53–69 (2024). https://doi.org/10.1109/TCC.2023.3336540
12. Filippini, F., Lattuada, M., et al.: A path relinking method for the joint online scheduling and capacity allocation of DL training workloads in GPU as a service systems. IEEE Trans. Serv. Comput. **16**(3), 1630–1646 (2023). https://doi.org/10.1109/TSC.2022.3188440
13. Galimberti, E., Guindani, B., et al.: OSCAR-P and AMLLibrary: performance profiling and prediction of computing continua applications. In: ACM/SPEC ICPE Companion, pp. 139–146 (2023). https://doi.org/10.1145/3578245.3584941
14. Gendreau, M., Potvin, J.Y. (eds.): Handbook of Metaheuristics. International Series in Operations Research & Management Science. Springer International Publishing (2019)
15. Guindani, B., Lattuada, M., Ardagna, D.: aMLLibrary: an AutoML approach for performance prediction. In: ECMS Proceedings, pp. 215–221 (2023). https://doi.org/10.7148/2023-0215
16. Kang, Y., Hauswald, J., et al.: Neurosurgeon: collaborative intelligence between the cloud and mobile edge. In: ASPLOS Proceedings, pp. 615–629 (2017). https://doi.org/10.1145/3037697.3037698
17. Lattuada, M., Gianniti, E., et al.: Performance prediction of deep learning applications training in GPU as a service systems. Clust. Comput. **25**(2), 1279–1302 (2022)
18. Li, E., Zeng, L., et al.: Edge AI: on-demand accelerating deep neural network inference via edge computing. IEEE Trans. Wirel. Commun. **19**, 447–457 (2019)
19. Lin, C., Khazaei, H.: Modeling and optimization of performance and cost of serverless applications. IEEE Trans. Parallel Distrib. Syst. **32**(3), 615–632 (2021)
20. Ray (2023). https://www.ray.io. Accessed 05 Jul. 2024
21. Risco, S., Moltó, G., et al.: Serverless workflows for containerised applications in the cloud continuum. J. Grid Comput. **19**(3), 1–18 (2021)

22. Saxena, V., Jayaram, K.R., et al.: Effective elastic scaling of deep learning workloads. In: MASCOTS, pp. 1–8 (2020). https://doi.org/10.1109/MASCOTS50786.2020.9285954
23. Sedghani, H., Filippini, F., Ardagna, D.: A random greedy based design time tool for AI applications component placement and resource selection in computing continua. In: IEEE EDGE, pp. 32–40 (2021). https://doi.org/10.1109/EDGE53862.2021.00014
24. Sedghani, H., Filippini, F., Ardagna, D.: A randomized greedy method for AI applications component placement and resource selection in computing continua. In: IEEE JCC, pp. 65–70 (2021). https://doi.org/10.1109/JCC53141.2021.00022
25. Shawi, R.E., Wahab, A., et al.: DLBench: a comprehensive experimental evaluation of deep learning frameworks. Clust. Comput. **24**(3), 2017–2038 (2021)
26. Zhu, X., Young, D., et al.: 1000 islands: an integrated approach to resource management for virtualized data centers. Clust. Comput. **12**, 45–57 (2009)