Hyperparameter Optimization using Agents for Large Scale Machine Learning

Pere Vergés Boncompte*†‡ Vladimir Vlassov‡, Rosa M. Badia*
*Barcelona Supercomputing Center, Barcelona, Spain
†Universitat Politècnica de Catalunya, Barcelona, Spain
‡KTH Royal Institute of Technology, Stockholm, Sweden
E-mail: {pere.verges, rosa.m.badia}@bsc.es, vladv@kth.es

Keywords—Machine Learning, Scalable Hyperparameter Search, Distributed Systems, High-performance computing, Task-based Workflow.

I. Extended Abstract

Machine learning (ML) has become an essential tool for humans to get rational predictions in different aspects of their lives. Hyperparameter algorithms are a tool for creating better ML models. The hyperparameter algorithms are an iterative execution of trial sets. Usually, the trials tend to have a different execution time.

In this paper we are optimizing the grid and random search with cross-validation from the Dislib [1] an ML library for distributed computing built on top of PyCOMPSs[2] programming model, inspired by the Maggy [3], an open-source framework based on Spark. This optimization will use agents and avoid the trials to wait for each other, achieving a speed-up of over x2.5 compared to the previous implementation.

A. Background

1) COMPSs and PyCOMPSs: The COMP Superscalar (COMPSs) is a task-based programming model that provides a programming interface for developing applications and a runtime system that exploits the inherent parallelism of applications at execution time. The main features of COMPSs are: sequential programming, programmers do not need to deal with parallel and distributed paradigms, agnostic of the computing infrastructure, offers single memory and storage space. COMPSs is a programming environment for developing complex workflows for parallelization. At runtime generates a task-dependency graph that encodes the existing parallelism of the application workflow.

2) Dislib: The Distributed Computing Library (dislib) is a machine learning library built on top of PyCOMPSs. This library has a very similar interface to scikit-learn. Not only, it can scale to large data, but also it is easy to use in high-performance computing clusters. These features make it easy to use for non-experts.

The main data structure of Dislib is the distributed arrays (ds-array). The ds-array is a matrix divided into blocks that will be stored remotely. All operations performed on the ds-arrays are parallelized using PyCOMPSs. The degree of parallelization is controlled by the size of the blocks in which the ds-array is being split.

It has implemented several ML algorithms, that range from Classification, Clustering, Regression, Decomposition, Pre-Processing, Neighbouring, and Model Selection, including the Grid and Random Search.

3) Agents and Nesting: The agent implementation in COMPSs [4] aims at enabling the offload the execution of functions of the embedded host to other nodes on Cloud-Edge Continuum. The application will request the execution of functions to the runtime. To do so, will show the execution logic, dependencies, resource requirements, data locations. After having received the request, it will invoke the runtime system. This runtime will handle the task execution asynchronously from the resource pool.

Having the agent implementation allows us to have nested tasks. Therefore, we can make a task invoke new tasks. This means that we have a parent task that invokes children. The parent will have to wait for the child to finish before finishing itself.

4) Hyperparameter Algorithms: Several algorithms are being used for hyperparameter [5] tuning, and also there are different approaches for doing parallel hyperparameter optimization that one can take [6]. However, we will focus on the ones implemented in the Dislib ML library. The two algorithms that we have are: Grid Search with Cross-Validation (CV), this algorithm evaluates exhaustively all the combinations of all parameters that have been given to it, and Random Search CV, in this case only a random subset of all the possible combinations will be executed. The core code of both algorithms is shared. We will refer to it as Base Search CV. The Base Search CV is in charge of the trial execution, which consists of a loop that will launch all the trials that have to be evaluated.

TABLE I. Execution time of the Grid Search CV with a different number of trials comparing the Initial Version with the New Version.

Number of trials	Initial Version time(s)	New Version time(s)
4	23.12	34.66
9	40.90	39.80
16	76.53	40.51
25	192.65	72.02
36	244.69	126.34

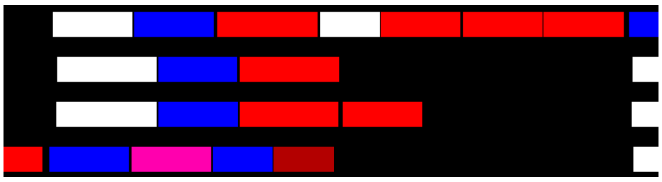

Fig. 1. Task view trace showing one iteration of the Grid Search CV algorithm, using the initial implementation and four threads.

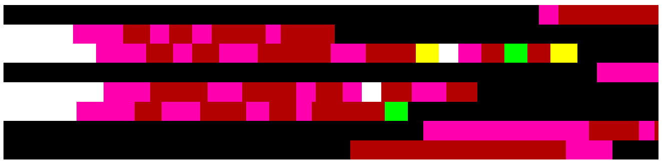

Fig. 2. Task view trace showing one iteration of the Grid Search CV algorithm, using the new implementation and four threads

B. Implementation

1) Initial Version: The initial version had a problem which was a synchronization at the end of the trails. The threads had to wait for each other, hence making them idle, which wasted time and resources. In Figure 1, we see one iteration of the Grid Search algorithm. We can observe four threads that are executing the trials. We see how the three last threads are waiting for the first one to finish its trial to start the following iteration.

2) New Version: The new version overcomes this limitation by using nested tasks. In this version, we are creating a task for each trial. Each trial will be able to invoke a new task with the fit and score tasks of the model being evaluated. This implementation will avoid the synchronization between iterations. Hence, we will optimize the resource and time utilization. In Figure 2, we have the first iteration of the Grid Search CV execution using the new version. It is a bit hard to visualize since we have that every new task will be scheduled in a new virtual thread (but at any point in time, we only have four). However, we can perfectly see that there are no threads idle waiting for other threads to finish.

C. Execution Environment

The evaluation of this implementation is done in the MareNostrum 4 Supercomputer. The configuration selected has been two nodes, where one acts as a master and the other as a worker. Each worker uses 48 CPUs.

D. Results

We have executed the Grid Search CV algorithm for the Cascade Support Vector Machines model. We have done several runs with a different number of trials to execute by the algorithm. In Figure 3 we see that for a small number of trials the new version does not improve. However, once we start to have a bigger number of trials the new versions start performing better, reaching a speedup of over x2.5. In Table I, we reflect the exact times for both executions.

E. Conclusion and Future Work

In this study, we have been able to demonstrate that the use of nesting has been successful at solving the problem

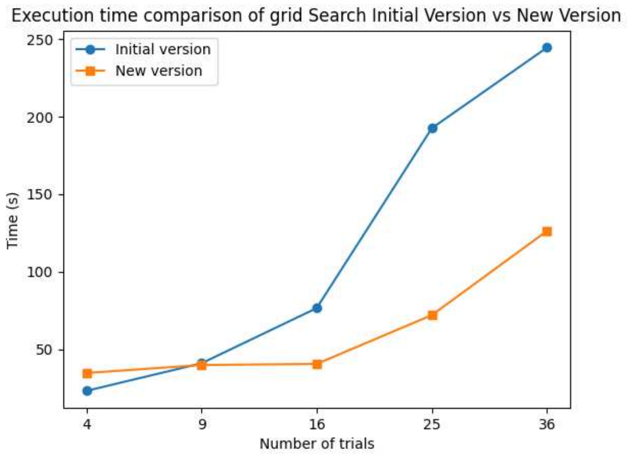

Fig. 3. Execution time of the Grid Search CV with different number of trials comparing the Initial Version against the new Version.

of parallel trial execution on the grid and random search algorithms. The solution has been able to achieve a speedup over x2.5. This speedup is due to avoiding the synchronization that was present in the initial version. This allowed us to take advantage of the time and resources previously being wasted.

As for future work, we have also started and implementation of early stopping criteria, using COMPSsExceptions [7] to emulate a wait for any call, which will also reduce the time execution and resource utilization even further.

References

[1] J. Álvarez Cid-Fuentes *et al.*, "dislib: Large Scale High Performance Machine Learning in Python," in *Proceedings of the 15th International Conference on eScience*, 2019, pp. 96–105.

[2] e. A. Lordan, F., "ServiceSs: an interoperable programming framework for the Cloud," in *Journal of Grid Computing*, 2014, p. 67–91.

[3] M. Meister, "Maggy: open-source asynchronous distributed hyperparameter optimization based on apache spark," in *FOSDEM 20*, 2019.

[4] D. L. F. Lordan and R. M. Badia, "Colony: Parallel Functions as a Service on the Cloud-Edge Continuum," in *Euro-Par 2021*, 2021.

[5] T. Yu and H. Zhu, "Hyper-parameter optimization: A review of algorithms and applications," *CoRR*, vol. abs/2003.05689, 2020. [Online]. Available: https://arxiv.org/abs/2003.05689

[6] L. Li *et al.*, "Massively parallel hyperparameter tuning," *CoRR*, vol. abs/1810.05934, 2018. [Online]. Available: http://arxiv.org/abs/1810.05934

[7] e. A. Ejarque J, "Managing Failures in Task-Based Parallel Workflows in Distributed Computing Environments," in *Euro-Par 2020: Parallel Processing*, 2020.

Pere Vergés Boncompte received his BSc degree in Computer Engineering from Universitat Politècnica de Catalunya (UPC), Barcelona, Spain in 2020, coursing one of the semesters at the University of Edinburgh. After that, he started his MSc degree in Innovation and Research in Informatics in Advanced Computing, at UPC and also started working at Workflows and Distributed System group of Barcelona Supercomputing Center (BSC), where he also coursed one semester at KTH Royal Institute of Technology, Stockholm, Sweden and now is finishing his Master Thesis at University of California Irvine (UCI).