Oikonomos: An Opportunistic, Deep-Learning, Resource-Recommendation System for Cloud HPC

Jan-Harm Betting
Erasmus Medical Center
Rotterdam, The Netherlands
j.betting@erasmusmc.nl

Dimitrios Liakopoulos
National Technical
University of Athens
dim.liak99@gmail.com

Max Engelen
Erasmus Medical Center
Rotterdam, The Netherlands
m.engelen@erasmusmc.nl

Christos Strydis
Erasmus Medical Center
Rotterdam, The Netherlands
c.strydis@erasmusmc.nl

***Abstract*—The cloud has become a powerful environment for deploying High-Performance Computing (HPC) applications. However, the size and heterogeneity of cloud-hardware offerings poses a challenge in selecting the optimal cloud instance type. Users often lack the knowledge or time necessary to make an optimal choice. In this work, we propose Oikonomos, a data-driven, opportunistic, resource-recommendation system for HPC applications in the cloud. Oikonomos trains a Multi-layer Perceptron (MLP) to predict the performance of a given HPC application, for different input parameters and instance types. It, then, calculates the cost of executing the application on different instance types and proposes the one best-fitting the user's needs. We deployed Oikonomos on a diverse mix of HPC workloads, and found that for all applications, it approached an optimal policy. The optimal instance type was chosen in 90% of the cases for seven out of eight applications, scoring a Mean Absolute Percentage Error (MAPE) consistently below 20%. This demonstrated that Oikonomos can provide a practical, general-purpose, resource-recommendation system for cloud HPC.**

***Index Terms*—resource recommendation, high-performance computing, deep learning, cloud computing, heterogeneity**

I. Introduction

Cloud computing allows users to access computing resources on demand, without specific knowledge about the location of those resources, and often without waiting times. This makes them an attractive option for High-Performance Computing (HPC): whereas traditional HPC data centers work with workload managers which place tasks in a queue of pending jobs, each to be executed on a statically defined hardware allocation, a modern cloud environment can elastically allocate the right amount of hardware to a user almost instantaneously, for as long as the user needs it. Users pay for the time that they use the resource.

Cloud services such as Amazon EC2 and Microsoft Azure now offer a wide range of instance types, with different combinations of system memory, CPU cores, GPUs, and even FPGAs. At the time of writing, EC2 offers 637 instance types, allowing users to tailor the hardware to the requirements of their applications. This makes cloud computing an interesting option for HPC and has led to the need for efficient resource orchestration and job scheduling in the cloud. To this end, technologies such as Kubernetes, Mesos and Docker Swarm have already been developed.

There is, however, an additional challenge for cloud HPC: given that application execution time is affected in a typically hard-to-predict way by both input size and parameters, as well as the hardware platform on which it runs, what is the optimal cloud-instance type to be employed for executing a particular application workload? Answering this question is crucial for minimizing application execution time, minimizing cloud compute fees, or both of the above. Especially for the case of HPC applications, such savings have a significant impact on the perceived quality of service and the budget required to complete a given task.

In this work, we present *Oikonomos*, an execution-time predictor and cloud-instance recommender for HPC users. The Oikonomos predictor relies on the use of a Deep-Learning (DL) network, specifically a Multi-Layer Perceptron (MLP), which learns based on multiple runs of a single HPC application, for different input parameters and on different instance types. By collecting training data, Oikonomos builds an application-specific prediction model. Then, using billing data from the cloud provider, it can estimate the cost of running the application on each of the instances. This allows the user to make an informed decision regarding the choice of instance type. Oikonomos, thus, bridges the gap between HPC applications and the wide variety of cloud instances available.

Oikonomos puts forward a unique proposal in that it can operate *opportunistically*: without the need for extra runs to help speed up training. In an HPC environment, the volume of user runs suffices for building an accurate prediction model, without adding to the user's usage bill. Furthermore, it is based on a generic MLP, which can be retrained for different applications, and is therefore useful for a wide variety of HPC applications. It also assumes a purely data-driven approach, which is simple, user-friendly, and robust, steering clear of complex techniques, such as application traces, microarchitecture simulators and benchmark sampling. Lastly, it can be trained in pure *user space*, thus guaranteeing user-data privacy and maximum flexibility. With the above properties of Oikonomos in mind, the contributions of this work are as follows:

- A novel, MLP-based, instance recommender for HPC applications in heterogeneous clouds.
- Empirical evidence that a single, properly sized MLP model can perform consistently well in predicting performance across diverse HPC workloads.
- An in-depth performance analysis of Oikonomos on eight

TABLE I
RELATED WORK IN THE FIELD OF CLOUD INSTANCE TYPE SELECTION

Publication	**Name**	**Approach**	**Input**	**Mechanism**	**Applications for evaluation**
Venkataraman et al. (2016)	Ernest	Prediction-based	Number of machines (CPU cores)	Non-Negative Least-Squares fitting	Various applications, such as speech recognition
Yadwadkar et al. (2017)	PARIS	Prediction-based	Application 'fingerprint', VM configuration	Random-Forest Model	Video encoding, compression, and Serving-style latency and throughput-sensitive OLTP workloads.
Chard et al. (2017)	SCRIMP	Prediction-based	Resource usage and performance data of profiling runs	Case-by-case comparison no recommender included	FastQC and BWA ALN
Alipourfard et al. (2017)	CherryPick	Search-based	Representive workload	Bayesian optimization	Benchmark applications on Spark and Hadoop
Hsu et al. (2018a)	Scout	Search-based	Low-level metrics and historical data from other workloads	Search-space exploration through relative ordering, pairwise comparison & transfer learning	Big Data workloads on Apache Hadoop & Spark
Hsu et al. (2018b)	Arrow	Search-based	Low-level performance information, such as CPU utilization and memory and I/O pressure	Tree-based learning method (Extra-Trees algorithm)	HiBench / spark-perf
Hsu et al. (2018c)	Micky	Search-based	Execution time & operational cost	Upper Confidence Bound	30 different Spark & Hadoop applications
Samreen et al. (2019)	Tamakkon	Prediction-based	Auxiliary data and historical data	Multivariate Polynomial Regression, Support Vector Regression, and Random Forests, using transfer learning	VARD, smallpt, and ItemRecommender
Samuel et al. (2020)	A2Cloud-RF	Prediction-based	PERF traces of HPC application, Cloud traces of benchmark application	Random-Forest Classifier	Several HPC applications: LULESH, HH SNNs, Best-Feature Digital Rotoscope, Data migration scheduler
This work	Oikonomos	Prediction-based	Instance type & hardware and application parameters	Multi-Layer Perceptron DNN	ARCHER, TensorFlow and simHH

diverse HPC applications, which shows its data-driven robustness for different applications and, thus, its high reuse value by non-expert HPC users.

The paper is organized as follows: Section II summarizes related works on performance prediction in cloud HPC environments. Section III describes the design concepts behind Oikonomos and Section IV details its implementation. Section V presents a detailed evaluation of the proposed system followed by a discussion of the findings and potential improvements in Section VI. Section VII concludes the work.

II. RELATED WORK

Related works can generally be classified as either prediction-based or search-based ones; see Table I. Prediction-based algorithms use offline evaluation of data to predict performance and can quickly suggest optimal hardware configurations but require large amounts of training data. Search-based approaches evaluate different instance types in succession to find the optimal choice, which does not require a large amount of data but may require exploration and making suboptimal choices. Additionally, exploration can be complex in cases where a certain job only needs to be run once, which is common in many HPC applications.

Venkataraman et al. [1] addressed the issue of performance prediction of large-scale analytics workloads on Amazon EC2 instances. The system, ERNEST, uses a prediction-based approach which fits four values to a formula using a non-negative least-squares (NNLS) solver with a 'scale' parameter and the number of machines in the job as its input. ERNEST was evaluated on EC2 `r3` instances and achieved errors of less than 20% on most workloads, but is limited in flexibility, designed for predictable workloads, and not suitable for heterogeneous instances.

Yadwadkar et al. [2] presented PARIS, another prediction-based approach for selecting the best Virtual Machine (VM) among multiple clouds. PARIS uses benchmark workload to create a profile of the system performance of each available instance type - this is only performed once for each instance type. Then, in order to find the best instance type for a particular task, the user has to provide a representative workload, which is executed on a small number of instances in order create a 'fingerprint' of the application. This 'fingerprint' is used to select the best instance via a random-forest model. Even though running the representative task is less costly than running the full job, it still incurs additional costs, burdens the user with presenting a representative workload, and does not account for differences in input parameters that can influence the usage patterns of applications.

Chard et al. [3] presented the Scalable Cost-Aware Cloud Infrastructure Management and Provision (SCRIMP), which is a multicloud middleware service for provision decisions for (scientific) applications. SCRIMP offers and interface to different cloud providers, and recognizes that the application requirements may be dependent on factors such as input data, application configuration, and execution environment. In order to describe these parameters as accurately as possible, SCRIMP introduces a profiling service. A user can submit a profiling request for their application, and receives a JSON file summarizing performance characteristics across different instance types under different configurations. This can be a good basis for a HPC resource recommender; however, SCRIMP does not provide such a recommender: it focuses on interaction with a multicloud infrastructure, not on resource recommendation.

Alipourfard et al. [4] presented CherryPick, a search-based approach that uses Bayesian optimization to build a performance model for applications. The user supplies a workload representative of the application, the objective, and the constraints, and CherryPick finds a list of candidates for the optimal hardware configuration. The system also runs benchmarking workloads on different clouds to measure the cloud noise and the Bayesian-optimization engine searches for the best configuration in an iterative manner. Even though the system finds the optimal configuration in a small number of iterations, it still requires running the workload multiple times to find the best configuration, and, like PARIS, is dependent on the selection of representative workloads by the user.

Hsu et al. presented three different search-based implementations; Scout [5], Arrow [6], and Micky [7]. Scout is a pairwise comparison algorithm that uses low-level performance information, such as memory and CPU usage, to avoid certain types of cloud configurations, while Arrow uses Bayesian optimization with a tree-based learning method, augmented with low-level data in order to reduce search costs. Micky approaches the problem of finding the best VM as a multi-armed-bandit problem and uses the strategy of Upper Confidence Bound to make choices with the highest expected rewards. Micky optimizes a batch of workloads, rather than a single workload, and can work in conjunction with Scout to search for the best configuration. However, all of these approaches require running the same workload multiple times to find the best configuration, which can be wasteful in real-life HPC scenarios where parameters vary.

More recently, two prediction-based solutions have been presented. Samreen et al. presented Tamakkon [8], which uses a transfer-learning approach to adapt a base learner (Multivariate Polynomial Regression, Support Vector Regression, and Random Forests are used) to a specific task by using profiling data from similar applications. To identify if two application-profiling data sets come from the same distribution, a Kolmogorov-Smirnov test is used. This makes the algorithm useful for different applications and hardware configurations. The systems does require the production of auxiliary data in the cloud, which entails additional costs.

Samuel et al. presented A2Cloud-RF [9], a prediction-based approach specifically aimed at HPC applications, which profiles the characteristics of the applications and cloud instances separately. The cloud instances are profiled with benchmarks for computations, main memory, and disk performance; the applications are profiled with computation and memory counters. The data is combined, and a Random-Forest Classifier (RFC) is used to make a recommendation about the instance. This decoupling eliminates the need for extensive data collection. However, it may not accurately reflect an application's needs because the features of an instance type are characterized by a predefined set of benchmarks, which can be restrictive.

Besides the aforementioned works being either too complex, too restrictive, too inflexible or too simplistic in nature, none of them takes the individual parameter values of the specific HPC job as input to predict the execution time or advice a cloud instance to the user. However, as Smaragdos et al. [10] showed, the input parameters of a HPC application can drastically affect the choice of hardware. Oikonomos aims at using the input parameters of an application to predict execution time in order to recommend an optimal instance type, without requiring knowledge about the performance structure of the application and without the need for running a specific job multiple times.

III. Oikonomos design

As mentioned above, Smaragdos et al. demonstrated the need for a decision-making middleware layer in an heterogeneous HPC system between the user and the hardware; one that selects the best-suited hardware platform among various available ones (CPUs, GPUs, dataflow FPGAs) for the task at hand [10].

The potential complexity and interdependencies of the various application parameters and the underlying execution platform call for an application-agnostic strategy for achieving performance prediction. In order to create a model that is generalizable, here we avoid performance-model construction and rely on a data-driven approach, provided that enough data is at our disposal every time. We propose the use of an MLP that takes application input parameters and available cloud-instance type data as its input to predict application execution time. This is used to advise the user on choosing an optimal instance, and giving an estimation of the computing costs they can expect.

The MLP is the most commonly used architecture in DL applications. MLPs are used to solve classification and pattern-recognition problems, but are also very suitable for regression analysis. Hence, they can be used for estimating numerical values. They are considered the most general-purpose types of artificial neural networks. There are more specialized DL architectures, such as Convolutional Neural Networks (CNNs), which are widely used in the field of computer vision, and Recurrent Neural Networks (RNNs), which are good at processing input sequences. However, since our data set consists of individual data points, an MLP is the most logical choice. An MLP learns through the process of backpropagation, in which the weights of the connections between different neurons of different layers are adjusted in order to minimize a predefined error function. Training is performed in a supervised way, with a training set in which the correct outcomes (in our case, the application execution times) are known. To prevent overfitting, an additional set (the validation set) is used to assess the prediction quality after each training epoch. After training, the network's functionality is assessed with a test set. It is important that these three data sets be independent from one another, in order to get a

TABLE II
DESCRIPTION OF THE APPLICATIONS USED TO EVALUATE OIKONOMOS

Application name	**Type**	**Source**	**No. of input parameters**	**Description**
BENCHIO	Synthetic	ARCHER	2	Processor write-bandwidth evaluation
GROMACS	Realistic	ARCHER	2	Molecular-dynamics simulation
CP2K	Realistic	ARCHER	313	Quantum-chemistry and solid-state-physics simulations
DL_POLY	Realistic	ARCHER	63	General-purpose classical molecular-dynamics simulation
HPCC	Synthetic	ARCHER	3	Benchmark collection for measuring the range of memory-access patterns
MNIST (MLP)	Realistic	Google TensorFlow	5	Training a deep MLP to recognize handwritten digits
CIFAR-10 (CNN)	Realistic	Google TensorFlow	5	Training a deep CNN to classify color images
simHH	Realistic	In-house	3	Extended Hodgkin-Huxley, biologically realistic, neural-model simulator

good indication of the quality of the trained network. Choosing optimal MLP hyperparameters, such as the number of layers, the type of activation functions and the learning rate, is a subject of research, for which different solutions have been proposed [11], but in many cases, remains a process of trial and error. However, we expect a single, general-purpose MLP to be robust enough to achieve performance prediction across a wide range of HPC applications.

A. *Oikonomos training mode*

For each HPC application, Oikonomos silently monitors multiple, user-initiated application runs, each with its own unique set of input parameters and records tuples of the form *(application, parameters, cloud instance, execution time)*. An assumption central to this work is that a sufficiently large number of runs takes place for Oikonomos to learn properly. This is usually the case in scientific or production HPC contexts where a single or multiple users are mounting HPC experiments via the same application for a prolonged period of time. A secondary assumption is that the targeted HPC application is implemented in such a way that it can be optimally run in at least two different platforms. For modern HPC applications, such an assumption is safe. Oikonomos treats the HPC applications as black boxes, which allows it to compare the behavior of different implementations of the same HPC application.

For each application run, the application parameters along with the various cloud-instance characteristics, such as the number of vCPUs, the type of CPU, amount of RAM, and information about the available GPU, comprise the Oikonomos training data to be fed as input into the MLP during the training phase. The MLP output consists of execution times, one per application run. Training starts when a sufficient amount of data for the creation of training, validation and testing sets of the MLPs has been collected. The process is illustrated in Figure 1.

B. *Oikonomos prediction mode*

After training the network to predict execution times, the performance predictions can be used to calculate the expected cost of running an application with a particular set of parameter values and hardware configuration. After all, cloud providers usually bill the use of instances per

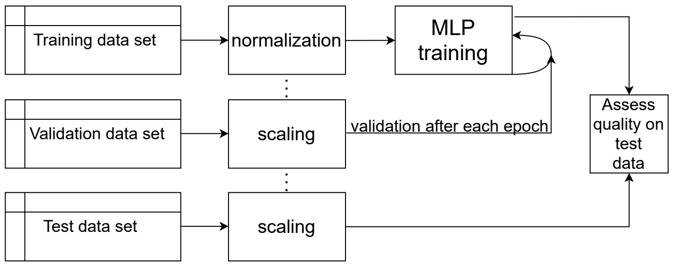

Fig. 1. *Oikonomos training mode:* The labelled dataset, which consists of the application parameter values and cloud-instance hardware specifications, combined with the measured execution time for each job, is divided into training, validation, and test sets. The training set is normalized and used for training. The validation set is scaled in the same way as the training set, and used to prevent overfitting. The test set is used to assess the trained MLP post-training.

time unit. In order to recommend an optimal instance type, Oikonomos concatenates the specific parameter values with the hardware configuration values of each of the instance types, and predicts the execution time for each possible instance. The cost predictor, then, calculates the predicted costs of running this job on each of the instances by multiplying the predicted time by the cost per second. In this way, Oikonomos gives a recommendation for an optimal instance choice. The prediction algorithm is summarized in Figure 2. If Oikonomos is used for multiple applications, each application needs its own trained MLP.

IV. IMPLEMENTATION

A. *Applications and benchmarks*

For evaluating the efficiency of Oikonomos, in this work we employ a mix of synthetic and realistic HPC applications, as shown in Table II. We selected five prominent benchmarks (BENCHIO, GROMACS, CP2K, DL_POLY, HPCC) taken from the ARCHER Hybrid Benchmark Suite, which was designed to be representative of the workloads of ARCHER, the UK National Supercomputing Service [12]. The five benchmarks that were available to us represent a balanced mix of both synthetic and realistic workloads. Apart from these benchmarks, we selected three additional, real HPC applications: simHH, MNIST (MLP), and CIFAR-10 (CNN).

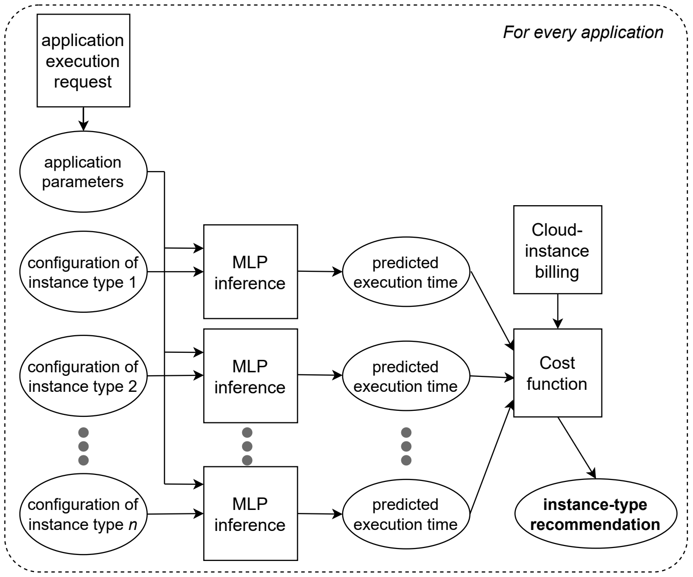

Fig. 2. *Oikonomos prediction mode:* On each new application-run request, the system takes as input the application-run parameter values, combines them with the configuration details of all targeted instance types and with their hyperscaler-provided usage costs, and outputs an instance recommendation to the user. Depending on the cost function selected, Oikonomos will recommend the fastest or cheapest instance to do the particular run on.

BENCHIO [13] is a synthetic parallel I/O benchmark, which evaluates the write bandwidth to a shared file per processor. The benchmark was designed by the Edinburgh Parallel Computing Centre (EPCC). It performs each test ten times, and evaluates the minimum, maximum, and average bandwidth returned. The code is written in Fortran 90, and writes the array in two ways: according to a 'single file, single writer' patterns, and in a 'single file, collective writer' pattern. The application writes in serial and parallel to three subdirectories. There are no available GPU or FPGA implementations of BENCHIO, but the number of MPI processes can be varied.

GROMACS [14] is a package for molecular-dynamics simulations, which is written in C/C++ with MPI and OpenMP parallelism. The package is free software, and it is widely used in scientific research: thousands of publications use GROMACS every year. The application employs multi-level parallelism, distributing computational work across different cores on each domain. Application performance will differ dependent on the hardware and the number of MPI-processes used. The application has support for GPGPU (CUDA) and Xeon Phi (KNL).

CP2K [15] is a quantum chemistry and solid-state physics software package, that performs atomic simulations of several kinds of systems. The package contains a large number of computational methods and simulation approaches, combining algorithms with good parallel scalability in order to exploit HPC architectures. The package was written in Fortran with MPI and OpenMP parallelism, and also supports CUDA, but only for specific GPUs.

DL_POLY [16] is general-purpose classical molecular-dynamics simulation software. It has been developed since 1994 and has been described as the first public, general-purpose molecular-dynamics package written specifically for parallel computers. The application employs different parallelization strategies in order to speed up the simulations. The application was written in Fortran with MPI and OpenMP parallelism and, like CP2K, supports CUDA.

HPCC [17] is a prominent collection of synthetic benchmarks that measure the range of memory-access patterns with MPI and OpenMP support. The HPCC benchmark consists of seven individual benchmarks (HPL, DGEMM, STREAM, PTRANS, RandomAccess, FFT, and Communications Bandwidth and latency). There are no available GPU or FPGA implementations of HPCC, but the number of MPI processes can be varied.

Apart from the ARCHER benchmarks, we employed three HPC applications. The first one is a versatile neurosimulator called simHH and developed at the Erasmus Medical Center, Rotterdam [18]. It can simulate a wide range of extended Hodgkin-Huxley neural models, which is a family of highly biologically plausible, conductance-based models. From a computational perspective, such networks typically are non-embarrassingly parallel workloads and their high simulation detail calls for invoking basic solvers operating on short time intervals.

Our second HPC application involves the training of an MLP Deep Neural Network, consisting of two layers and built with Google TensorFlow. We employed the publicly scrutinized MNIST database [20]. It contains images of a large number of handwritten digits and is often used to test AI classification applications. MNIST is a standard dataset in Google's TensorFlow library. There are several training

TABLE III
HPC APPLICATIONS USED FOR THE OIKONOMOS EVALUATION, AND THEIR PARAMETER RANGES

(A) BENCHIO

parameter	range
MPI-procs	1-8
Multiplier	1-10

(B) GROMACS

parameter	range
MPI-procs	1-8
Multiplier	1-10

(C) CP2K

parameter	number
No. of different bench-tests	313

(D) DL_POLY

parameter	number
No. of different bench-tests	63

(E) HPCC

parameter	range
MPI-procs	4-8
N	1000 - 25000
NB	100 - 10000

(F) simHH

parameter	range
time-steps	1 - 300,000
connectivity	0.2 - 1.0
neurons	20 - 20,000

(G) MNIST (MLP)

parameter	range
epochs	1 - 1000
Training batch size	1 - 5000
Test batch size	1 - 5000
Layer 1 size	1 - 5000
Layer 2 size	1 - 5000

(H) CIFAR-10 (CNN)

parameter	range
epochs	1 - 1000
Training batch size	1 - 5000
Test batch size	1 - 5000
Layer 1 size	1 - 5000
Layer 2 size	1 - 4

hyperparameters that can be varied, and the time it takes to run the training algorithm varies with those parameters.

For our third HPC application, we used the training of a Convolutional Neural Network (CNN), also built with TensorFlow. As a dataset, we used the CIFAR-10 database, which consists of thousands of color images from ten classes. The goal is to classify the images correctly. Like MNIST, the CIFAR-10 dataset is a standard dataset in Google's TensorFlow library. As variable parameters, the number of convolutional layers (1, 2, or 3), the size of the fully-connected layer, the number epochs, and the test and training minibatch sizes were chosen. These parameters influence the training time.

In our situation, the application parameters that we vary in order to create the datasets for the applications are more or less uniformly distributed. In a real-life scenario, this will not be the case: users may choose some parameter-value combinations more often than others. However, data biasing is a well-studied problem in modern AI and various techniques exist for dealing with the problem [19]. In our evaluation of Oikonomos, we use uniform distributions directly since debiasing techniques such as the above can be routinely combined with Oikonomos in a real setting.

B. MLP-based resource recommendation

The prediction and recommendation algorithm was implemented in PyTorch, an open-source, machine-learning framework which allows for development and implementation of DL models in Python. The datasets were divided into a training set (70% of data), a validation set (15% of data), and a test set (remaining 15%). The input parameters were normalized so that each parameter in the training set was distributed with $\mu = 0$ and $\sigma = 1$. The same was done for the output values, which consist of a single value per sample: the execution time.

As for the MLP-based predictor, we expected that the MLP depth, activation function and training batch size ought to have a bearing in performance. We explored a range of diverse MLP architectures. In this work, we chose to present the performance of two specific ones, deep enough to learn the complexity of realistic HPC workloads. The first MLP architecture (*MLP 1*) consists of seven fully connected layers. After the first four layers, the `tanh` activation function was used. The subsequent two hidden layers are followed by ReLU activation functions. The second MLP architecture (*MLP 2*) consists of nine fully connected layers, the first five followed by `tanh` activation functions, and the subsequent three layers by ReLU activation functions. All but the last two layers were followed by dropout layers, which improves learning.

Each MLP was trained with mini-batches of size 8. As the loss function, several options were considered, such as the L1 (Least Absolute Deviations) Loss, Mean Absolute Percentage Error (MAPE), the Mean Squared Error (MSE), and the Root Mean Squared Error (RMSE). The L1 error was used, which was observed to work well for training the MLP. The number of epochs was based on the validation loss: after each epoch, the training and validation losses were calculated, and the weights associated with the lowest validation loss were retained. Because the data in the validation set is separate from the data in the training set, this method prevents overfitting.

V. Evaluation

A. Experimental setup

The instance types that were used for this work are shown in Table IV. The data was generated using eight different types of instances in Amazon EC2. Of the eight instance types, two have a GPU available. The number of vCPUs ranges from 8 to 48. The instances have been picked in order to present diverse hardware options, but we also chose multiple instances from the same family to test if Oikonomos can discern the best among those, too. Four of the instances were general-purpose instances (`t2.2xlarge`, `m5a.4xlarge`, `m5a.8xlarge`, and `m5a.12xlarge`), two were compute-optimized (`c5a.4xlarge` and `c5a.8xlarge`), and two were accelerated-computing instances (`g3.4xlarge` and `g4dn.4xlarge`). We collected the required data by running the applications on these instances, with different parameter values. The parameter ranges that were chosen for all the applications are displayed in Table III.

After generating the datasets by running the applications in the cloud with the relevant parameters and expanding the dataset using our insights in the applications' behaviors, the data set was divided into training, validation, and test sets, as discussed in Section III. The training data was standardized, with its mean set to 0 and its standard deviation set to 1, as is commonly done for training data for MLPs. The validation and test sets were scaled using the same parameters as the training set.

The instance type was also included as an input parameter to Oikonomos, employing one-hot encoding. The same was done for hardware types, such as a specific GPU, being present in a given instance or not. Also quantifiable hardware aspects, such as the number of vCPUs, GPUs, FPGAs and the memory available were presented as input parameters. In this way, the MLP is able to train for multiple instance types. The MLP was trained, and converged rapidly from the first few epochs on. The training process was stopped when the validation loss stopped decreasing, to prevent overfitting.

B. Results

The evaluation of the trained MLP for each application was done on the test sets. This gives a good indication as to how well the network performs when a new job request comes in. We present the prediction error as the Mean Percentage Error (MPE) and the Mean Absolute Percentage Error (MAPE), in order to show the error in proportion to the actual execution time.

However, since the main purpose of Oikonomos is to accurately choose the best instance type (either the fastest or the most cost-efficient option), a better way to assess the quality of the predictions is to compare the resource selection of Oikonomos to an optimal policy and a random policy.

The data from [21] were used to determine the cost of Linux, On-Demand instances in Amazon. These types of

TABLE IV
AMAZON EC2 INSTANCE TYPES USED FOR THE OIKONOMOS EVALUATION. TYPES OF HIGH DIVERSITY, WITH VARIOUS NUMBERS OF CPU CORES AND DIFFERENT TYPES OF GPUS, AS WELL AS SIMILAR TYPES FROM THE SAME FAMILY, HAVE BEEN SELECTED ON PURPOSE.

Instance type	**CPU type**	**vCPU no.**	**Memory (GiB)**	**GPU type**	**GPU mem. (GiB)**
`t2.2xlarge`	Intel Xeon Family @ 3.3 GHz	8	32	–	–
`c5a.4xlarge`	AMD EPYC 7R32 @ 2.8 GHz	16	32	–	–
`m5a.4xlarge`	AMD EPYC 7571 @ 2.5 GHz	16	64	–	–
`m5a.8xlarge`	AMD EPYC 7571 @ 2.5 GHz	32	128	–	–
`c5a.8xlarge`	AMD EPYC 7R32 @ 2.8 GHz	32	64	–	–
`m5a.12xlarge`	AMD EPYC 7571 @ 2.5 GHz	48	192	–	–
`g3.4xlarge`	Intel Xeon E5-2686 v4 @ 2.3 GHz	16	122	Tesla M60	8.0
`g4dn.4xlarge`	Intel Xeon Family @ 2.5 GHz	16	64	Tesla T4	16.0

TABLE V
OIKONOMOS EVALUATION (MLP 1 / MLP 2)

	BENCHIO	GROMACS	CP2K	DL_POLY
Mean Percentage Error (MPE)	2.35% / 1.06%	0.67% / 1.79%	0.01% / -0.11%	-0.03% / -0.02%
Mean Absolute Percentage Error (MAPE)	4.50% / 4.68%	2.32% / 3.28%	1.94% / 1.74%	0.52% / 0.53%
Time regret %	3.99% / 3.88%	1.85% / 4.10%	0.00% / 0.00%	0.01% / 0.02%
Cost regret %	5.36% / 10.53%	0.00% / 0.00%	0.01% / 0.01%	0.00% / 0.00%
Time/cost regret of random policy	35.00% / 152.64%	50.76% / 280.45%	52.80% / 124.12%	20.67% / 150.13%
Correct instance type chosen (time)	60.30% / 52.10%	83.10% / 71.70%	99.90% / 99.90%	99.40% / 99.10%
Correct instance type chosen (cost)	96.80% / 85.30%	100.00% / 100.00%	99.90% / 99.90%	100.00% / 100.00%
	HPCC	simHH	MNIST	CIFAR-10
Mean Percentage Error (MPE)	0.51% / 1.96%	3.68% / 0.61%	16.83% / 14.81%	-2.69% / 7.46%
Mean Absolute Percentage Error (MAPE)	4.16% / 5.40%	12.84% / 13.23%	19.63% / 16.67%	7.93% / 10.94%
Time regret %	0.65% / 0.27%	196.76% / 111.80%	6.00% / 0.04%	3.02% / 0.26%
Cost regret %	0.36% / 0.38%	196.09% / 63.14%	6.45% / 4.64%	2.24% / 0.84%
Time/cost regret of random policy	72.05% / 403.08%	744.95% / 389.90%	284.63% / 537.12%	1000.45% / 666.95%
Correct instance type chosen (time)	39.10% / 73.70%	22.91% / 65.74%	98.86% / 99.97%	99.46% / 98.65%
Correct instance type chosen (cost)	98.90% / 98.70%	24.21% / 73.37%	94.28% / 94.89%	98.92% / 95.95%

instances are billed per second, from the time of launch until their termination. For the purpose of this research, it was assumed that each instance is launched for one single application run, after which it is immediately terminated.

Table V summarizes the results for all eight applications. Both the Mean Percentage Error and the Mean Absolute Percentage Error per application are shown. We also present the *regret* (the difference between the Oikonomos policy and an optimal policy) as a percentage of the optimal policy. This shows how much more time-consuming or costly the Oikonomos policy is compared to an optimal policy. For comparison, we also added the regret percentages for a random policy. Lastly, we assess in which percentage of cases the choice of the Oikonomos policy matches those of the best policy.

For all applications with either of the two MLPs, Oikonomos performs much better than a random policy. It can also be seen that the errors for the five benchmarking applications are usually lower than those of the HPC applications. This is expected, since for these applications, the relationship between parameters and execution time is more complex than for the ARCHER applications.

In general, there is little difference between the results for the two MLP architectures. For all eight applications and each MLP, the Oikonomos policy far outperforms a random policy. The choice of MLP matters in some cases: for HPCC, simHH, and MNIST, MLP 2 performs much better than MLP 1. Interestingly, higher error percentages do not always correlate with higher regret: selection of the correct instance is more important than an accurate prediction.

The graphs in Figure 3A show a visualization of the cost estimations of the Oikonomos system for the MNIST MLP training with particular parameter configurations. The figures show that different parameter configurations can lead to different cost-optimal configurations. This confirms what Smaragdos et al. [10] have described in their work: the optimal hardware configuration of an HPC application strongly depends on its parameter values.

Figure 3B shows confusion matrices for the optimal instance choices for the MNIST MLP training, both regarding time and cost, for a test set of jobs that were executed on all the instances. The diagonal shows the count of jobs where the option recommended by Oikonomos corresponds to the actual best option. In the test set, the fastest option was always the `g4dn.4xlarge` instance type. However, since this instance type has also the highest costs per hour, it is not always the cheapest option. Oikonomos was able to identify this dilemma and recommend the correct instance type in most cases.

VI. DISCUSSION

The experimental results reveal that Oikonomos can produce accurate resource recommendations for several applications:

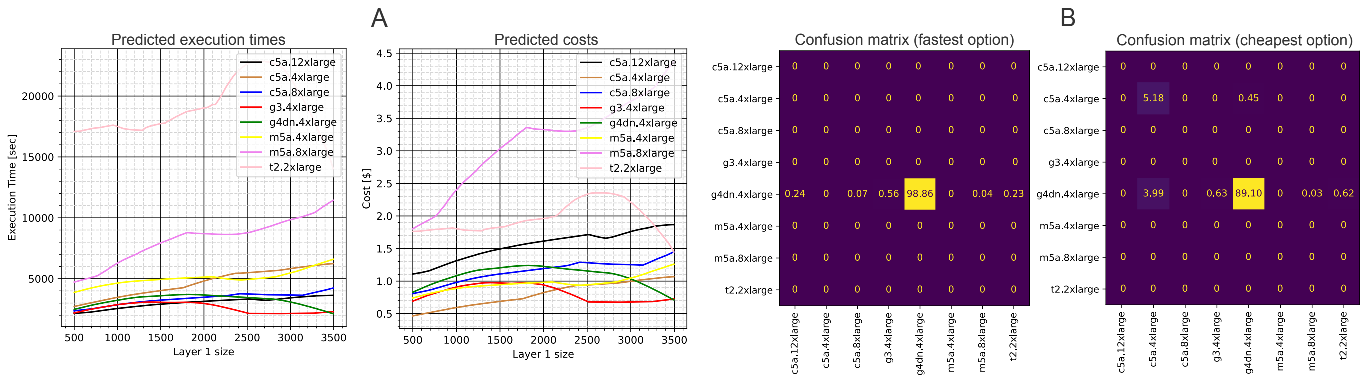

Fig. 3. **A**: Cost and execution time estimations for running the MNIST MLP training, as estimated by MLP 1. To allow for a two-dimensional plot, scaling of only a single parameter for two distinct values of a second parameter is shown here. Based on the plots, for each combination of parameter values, the cheapest instance type can be found. **B**: Confusion matrices for the fastest and cheapest option for MNIST MLP training. Recommendations shown on the y-axis, true labels shown on the x-axis. Numbers are percentages of the test set. Most estimations are on fields along the diagonal, which indicates that Oikonomos chooses the correct instance in most cases.

five benchmark applications and three HPC applications. It was possible to retrain the MLP for very different applications without the need to modify the MLP architecture. All in all, we showed that it is possible to create a recommender system for HPC applications in cloud environments that is portable to different applications and types of cloud instances.

Nevertheless, Oikonomos in its current form does have some potential limitations. It uses a lot of training data to train its MLP. In our experiments, we were able to generate that data, but in a real-life situation, where execution-time data from actual user runs is gathered, it might not be available at first. However, as an HPC application is used for a longer period of time by a significant number of users, the MLP can be retrained periodically so that its quality gradually improves. The reliance on a large data volume is a limitation. Further research is needed to determine what amount of data is sufficient for accurate instance-type recommendation.

The MLP was trained with data following a uniform distribution over all parameters. In situations where training data is derived from user runs, this will not be the case. There, it might be necessary to implement debiasing techniques, such as oversampling or weighting. However, since this is a common issue with datasets that are used for training DL networks, multiple techniques exist that could address this.

Only 8 instance types were used for each of the seven applications to generate the input data. At this point, Amazon EC2 offers 567 different types of instances, and even though only a subset of those will be suitable for a particular application, further research is needed to see how Oikonomos would work on the full set of available instance types.

We compared the performance of Oikonomos to a random and the optimal policy, but we were unable to compare Oikonomos' performance to earlier work. Unfortunately, in contrast to other fields, there are currently no standardized benchmarks for cloud-resource recommendation. Because of this, direct quantitative comparison between recommendation systems is currently infeasible. The eight applications used in this paper could be a starting point for the development of such a benchmark set.

Even though we carefully selected our benchmark applications and instance types to collect our data, Figure 3 illustrates that for most jobs, the best instance type (in terms of speed or cost) is still one of two options, or overwhelmingly one and the same instance type. The effect described by Smaragdos et al. is relatively minor for our applications, instance types, and parameter ranges. This is not a feature of Oikonomos (which also performs well in cases where there is one overall best option), but a feature of the benchmark set. This is another reason why a standardized set of cloud HPC benchmark applications is needed: a set of applications that truly behave differently on different hardware type would be better able to show the flexibility of Oikonomos, or any other cloud resource recommendation system.

In this work, we assumed that the instances were already running. Since it takes time to launch an instance and install the necessary applications on it, shutting down an instance after launch is not optimal if simulations are requested frequently. However, keeping an instance running in anticipation of new jobs is also costly. Developing an algorithm that determines the cost- or time-effectiveness of refraining from terminating an instance post-execution holds promise but, since it requires additional data about the frequency patterns of simulation requests, it is outside the scope of this work.

While the MLP architectures were designed regardless of the applications, they performed exceptionally well for all the benchmarking tools and HPC applications. This provides evidence that a well-designed, general-purpose MLP could be applicable for any number of HPC applications. However, further research is needed to trace the lowest amount of data needed for the predictor to work correctly.

VII. Conclusions

Modern clouds are powerful platforms for executing HPC applications. However, the growing number of heterogeneous instance types and hardware configurations makes it crucial for users to gain insight in optimal choices. This work confirmed earlier research that, for HPC applications, the optimal choice of hardware for an application job can depend on the parameter values of that job. To tackle this challenge, we proposed Oikonomos, an optimal resource-recommender system. Oikonomos uses a general-purpose MLP for predicting the execution time of an HPC application with different input parameters and on different cloud-instance types. Based on that, it recommends the option that minimizes cost or execution time per run. We tested Oikonomos with two different MLP architectures, on three HPC applications – two TensorFlow kernels and a brain simulator – as well as a mix of five synthetic and realistic ARCHER benchmarks, on various different instance types on Amazon EC2. Evaluation results showed that the Oikonomos policy led to very low regret rates compared to a random policy, approaching an optimal policy. For seven out of the eight applications, the optimal instance type was chosen in more than 90% of cases, scoring a MAPE below 20% for all applications.

References

[1] S. Venkataraman, et al., "Ernest: Efficient performance prediction for large-scale advanced analytics," in 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16), 2016.

[2] N. J. Yadwadkar, et al., "Selecting the best VM across multiple public clouds: A data-driven performance modeling approach," in Proceedings of the 2017 Symposium on Cloud Computing, 2017.

[3] R. Chard, K. Chard, R. Wolski, R. Madduri, B. Ng, K. Bubendorfer, and I. Foster, "Cost-aware cloud profiling, prediction, and provisioning as a service," *IEEE Cloud Computing*, vol. 4, no. 4, pp. 48–59, 2017.

[4] O. Alipourfard, et al., "CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics," in 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), 2017.

[5] C.-J. Hsu, et al., "Scout: An Experienced Guide to Find the Best Cloud Configuration," arXiv preprint arXiv:1803.01296, 2018.

[6] C.-J. Hsu, et al., "Arrow: Low-level augmented bayesian optimization for finding the best cloud vm," in 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), IEEE, 2018.

[7] C.-J. Hsu, et al., "Micky: A cheaper alternative for selecting cloud instances," in 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), IEEE, 2018.

[8] F. Samreen, G. Blair, and Y. Elkhatib, "Transferable Knowledge for Low-cost Decision Making in Cloud Environments," IEEE Transactions on Cloud Computing, 2020.

[9] D. Samuel, et al., "A2Cloud-RF: A random forest based statistical framework to guide resource selection for high-performance scientific computing on the cloud," Concurrency and Computation: Practice and Experience, vol. 32, no. 24, 2020.

[10] G. Smaragdos, et al., "BrainFrame: a node-level heterogeneous accelerator platform for neuron simulations," Journal of Neural Engineering, vol. 14, no. 6, 2017.

[11] M. Ettaouil and Y. Ghanou, "Neural architectures optimization and Genetic algorithms," WSEAS Transactions On Computer Research, vol. 8, no. 3, pp. 526–537, 2009.

[12] A. Turner, "UK National HPC Benchmarks - ARCHER," https://www.archer.ac.uk/documentation/white-papers/benchmarks/UK, 2016, accessed 2022-08-17.

[13] D. Henty, et al., "Performance of Parallel IO on ARCHER," 2015.

[14] M. James, et al., "GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers," SoftwareX, vol. 1-2, pp. 19–25, 2015.

[15] T. D. Kühne, et al., "CP2K: An electronic structure and molecular dynamics software package - Quickstep: Efficient and accurate electronic structure calculations," The Journal of Chemical Physics, vol. 152, no. 19, 2020.

[16] W. Smith and I. Todorov, "A short description of DL_POLY," Molecular Simulation, vol. 32, no. 12-13, pp. 935–943, 2006.

[17] P. Luszczek, et al., "Introduction to the HPC challenge benchmark suite," Technical report, Lawrence Berkeley National Lab (LBNL), Berkeley, CA (United States), 2005.

[18] M. Engelen, "Scalable GPU Acceleration for Complex Brain Simulations," MSc thesis, Delft University of Technology, 2021. Available at: http://resolver.tudelft.nl/uuid:b79bbfa7-0c57-4949-b974-83a7d9ee6b39

[19] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 559–563, 2017.

[20] Image classification, MNIST digits, http://neupy.com/2016/11/12/mnist_classification.html, accessed 14-12-2021.

[21] Easy Amazon EC2 Instance Comparison, https://EC2Instances.info, accessed 14-12-2021.