Machine learning for dynamic resource allocation at network edge

Bong Jun Ko*[a], Kin K. Leung[b], Theodoros Salonidis[a]
[a]IBM Thomas J. Watson Research Center, 1101 Kitchawan Road, Yorktown Heights, NY 10598;
[b]Imperial College London, Exhibition Road, London SW7 2AZ, United Kingdom

ABSTRACT

With the proliferation of smart devices, it is increasingly important to exploit their computing, networking, and storage resources for executing various computing tasks at scale at mobile network edges, bringing many benefits such as better response time, network bandwidth savings, and improved data privacy and security. A key component in enabling such distributed edge computing is a mechanism that can flexibly and dynamically manage edge resources for running various military and commercial applications in a manner adaptive to the fluctuating demands and resource availability. We present methods and an architecture for the edge resource management based on machine learning techniques. A collaborative filtering approach combined with deep learning is proposed as a means to build the predictive model for applications' performance on resources from previous observations, and an online resource allocation architecture utilizing the predictive model is presented. We also identify relevant research topics for further investigation.

Keywords: Edge computing, resource allocation, machine learning, collaborative filtering

1. INTRODUCTION

The advances in mobile and embedded computing technologies are making the edge computing paradigm a viable alternative to centralized, cloud-based one. Running applications at network edges is particularly attractive as it effectively addresses so-called "data bottleneck" issues in cloud-based systems, such as communication bandwidth consumption, latency, and data security and privacy. These issues are becoming increasingly important as many applications are now employing various analytics approaches to processing a vast amount of data generated at the edge, in particular in IoT (Internet of Things) domains.

At network edge, however, the computing, networking, and storage resources required for performing application tasks are generally scarce, compared to the computing clouds where these resources can be easily provisioned and scaled according to the demand. Furthermore, the availability of these resources are dynamic and time-varying, due to workloads variation and the need of sharing the limited resources across different applications. For realizing the full benefit of the promise of the edge computing, it is therefore imperative to effectively, efficiently, and flexibly manage the dynamic edge resources.

Traditional approaches to the resource management rely on heuristics under the principle that we call "demand-capacity" (DC) model,[1,2] in which it is assumed that application tasks arrive with a certain level of resource demands (or requirements), and the resources have a certain level of capacities. Then, the applications are assigned to resources in a manner that the resource's capacity can accommodate the application demand with some kind of combinatorial optimization performed under certain criteria, such as minimizing the total resource consumption, maximizing the application performance, etc.

In reality, such a demand-capacity model represents only a crude approximation of how the applications actually behave on the resources. In fact, many applications are rather *elastic*, meaning that there is no clearly defined resource requirements, and their performance vary according to the differences in the resources assigned for them (e.g., increase or decrease in latency, response time, processing throughput, etc.). The applications may yield low performance under limited resource condition, yet they can still remain functional. In edge computing environments, where the resource availability is limited and dynamic, this notion of elasticity can be effectively exploited to improve the utilization of the resources if they can be adequately and accordingly managed.

There are several challenges in managing the resources for elastic applications. First, it is difficult to establish in advance the performance model of the applications on a given level of resources at the network edge, where both the resources and

the applications are heterogeneous. This heterogeneity issue becomes more severe when different configurations of the applications and the resources are to be taken into account. Furthermore, there could be multiple application performance metrics of interests to the applications, each of which can work differently with different composition of the resources in a non-trivial way. For example, an application can behave more or less the same in terms of one performance metric (e.g., throughput) under two different resource compositions (e.g., one with large computing resources and small memory, and the other with small computing power and large memory), but the same application can exhibit a very different performance profile for some other metrics (e.g., response time, reliability) under these two conditions. Such difficulty in establishing the analytical models of the applications' behavior renders the analytical/combinatorial approaches practically infeasible.

In this paper, we present the design of methods and architecture for dynamic resource allocations at edge, based on machine learning (ML) approaches as a novel alternative to the analytical solutions. At a high level, machine learning techniques provide an effective means to *learn* the model of the interactions between the applications and resources from the observed behaviors. Furthermore, recent advances in deep learning (DL) architectures have proved that non-intuitive and non-linear relationships hidden in the raw data can be effectively discovered without explicit feature engineering in many domains of data analytics. As such, ML/DL can provide an ideal platform to build a solution for addressing the aforementioned challenges in edge resource management.

We explore the problem in the context of assigning heterogeneous application tasks to heterogeneous edge nodes, where the performance of the tasks varies elastically depending on the changes in the resource level made available to them. After presenting the system model for the task-to-node assignment problem (Section 2), we describe a method based on *collaborative filtering*, which can predict the tasks' performance on the nodes of various resource profiles by learning from previous experiences not only of the same application, but also of the other applications (Section 3). Then we present the overall architecture and strategies for the task-node assignment that take advantage of the predictive model (Section 4) before concluding the paper with some topics of further investigation in the future (Section 5).

2. MODELS, ASSUMPTIONS, AND OBJECTIVES

In this section, we present a system model for a resource management problem in the context of assigning the applications to a set of edge resources. As a canonical example, we consider the case of assigning application tasks (or simply 'tasks') to nodes at the network edge, where nodes could be either physical devices or virtual ones (e.g., virtual machines or containers).

We assume the tasks are elastic, that is, they do *not* have specific resource requirements for the nodes they are assigned to. Rather, assigning a task to a node would result in the task to exhibit a certain behavior in terms of its performance metrics, which is dependent on the task's own properties and the node's available resource profile. Here, a node's resource profile is a multi-dimensional vector that indicates the levels of various resources (CPU, memory, I/O, network bandwidth, etc.) made available for the task being assigned to it. We also assume that the performance profile, i.e., the performance metrics resulting from assigning a task to a node, is not known in advance—it needs to be *learned* from the observation on tasks getting executed on the nodes.

We denote a set of application *tasks* by $T = \{t_1, \cdots, t_m\}$, and a set of *nodes* by $V = \{v_1, \cdots, v_n\}$. Each task $t_i \in T$ is associated with its task profile vector $< t_{i1}, \cdots, t_{ik} >$, which indicates the configurable properties of the task (e.g., the configuration parameters of application). Similarly, each node $v_j \in V$ is associated with its resource profile vector $< v_{j1}, \cdots, v_{jl} >$ representing the levels of available resources on it.

Assigning a task to a node produces a performance profile, which is modeled by a mapping $F: T \times V \rightarrow \mathbb{R}^d$ that maps each (t, v)-pair into a vector of d performance metrics. This mapping function is what we would like to learn from the experience (or "observation") on assigning and executing the tasks on nodes. In addition, each task t can be associated with a utility function $U^t(p_1, \cdots, p_d)$ given a performance profile $< p_1, \cdots, p_d >$, which maps the multiple performance metrics into a single number. For example, a utility function could be defined as a weighted linear sum of the individual performance metrics, where weights are determined according to the relative importance of each metric (deemed by the application task). Note that the utility function can be different for different tasks.

In the remainder of the paper, we will assume the task profile is fixed for each task. When we need to consider the cases of a task with varying profiles (e.g., an application deployed with different configuration parameters at different times or on different nodes), we will simply treat those variations as separate tasks. The resource profile of the nodes, however,

would vary over time depending on the current assignments of the tasks, because deployed tasks consume the nodes' resources and hence generally decrease the available resources on them (In Section 4, we describe an architecture in which such time-varying resource profiles are monitored and updated when they change).

At a high level, our objective is to find the "best" assignment of the tasks to the nodes, where the "goodness" can be measured from two different perspectives:

- Optimize for the resource utilization, e.g., find fair resource allocation in min-max sense
- Optimize for tasks' performance, e.g., maximize the overall utility function of the tasks in aggregate or max-min sense

A conventional approach to formulating such an assignment problem is through a combinatorial optimization—a brute-force solution would (i) try every combination of assignment in $T \times V$, and observe the resulting performance profile and the utility function, and (ii) select the best assignment from the observed performance profiles. An obvious issue in such combinatorial approaches is their complexity in the search space as n^m possible assignments exist with m tasks and n nodes. Various heuristics could be used to reduce the search space (e.g., greedy approaches) and find approximate solutions for step (ii) above. Remember, however, in our case the assignment-to-performance mapping function F is not known in advance and needs to be learned from experience. Hence, even with moderate numbers m and n, obtaining necessary information (i.e., step (i) above) will be prohibitively expensive.

We address the scalability issue of the task assignment by leveraging machine learning approaches. Our proposed solution is broken down into two steps:

- Performance prediction, in which a model that predicts the performance of unobserved task-to-node assignments by learning from the measured performances of the observed assignments.
- Assignment method, in which the assignment is guided by the predicted and observed performance.

3. PREDICTIVE PERFORMANCE MODELING

Our goal here is to develop a method for learning a predictive model $\hat{F}$ of the performance mapping function $F: T \times V \rightarrow \mathbb{R}^d$ from observed task-to-node assignments. The challenge is the sparsity of the available observations: Since making an observation of a task's performance under an assignment requires deploying and running the task on the actual system, the available number of observations that can be used to build a predictive model can be limited, compared to the theoretical n^m combinations in total.

Our strategy is to view the assignment problem in the framework of a recommendation system, that is, a node is *recommended* to a task, based on the performance observations of other assignments, not only of the task itself, but also of other tasks. Brining the analogy to a typical recommendation problem, say, movie recommendation by users' rating, we can consider the tasks as the users (movie viewers), the resources as the movies, the performance observations as the movie ratings, and the assignment as the movie recommendation.

More specifically, we employ **collaborative filtering** (CF) approaches to building the performance prediction model. Collaborative filtering[3] is a method of recommending ('finding' in our context) items ('nodes') to users ('tasks') based on the similarity of the users' experience on common items. In the following, we start by describing the basic concept of CF approaches in the context of task-to-node assignment. We then propose a more scalable approach based on a deep neural network architecture.

3.1 Concept of CF-based Performance Prediction Model

Figure 1 illustrates the concept of applying CF to our assignment problem through an example. There are two basic categories of CF methods: user-based ones and item-based ones, and we use a user-based CF to describe the concept (the item-based approach works in a similar way). For simplicity, we assume in this example that the application tasks' profiles and nodes' resource profiles are all fixed, meaning that the problem is effectively reduced to finding the prediction of tasks' performance on nodes without considering their variations. We also assume there is only one performance metric—extending the scenario to multi-metric case can be done in a straightforward manner, either by considering each metric separately or by using the performance utility function as the single metric.

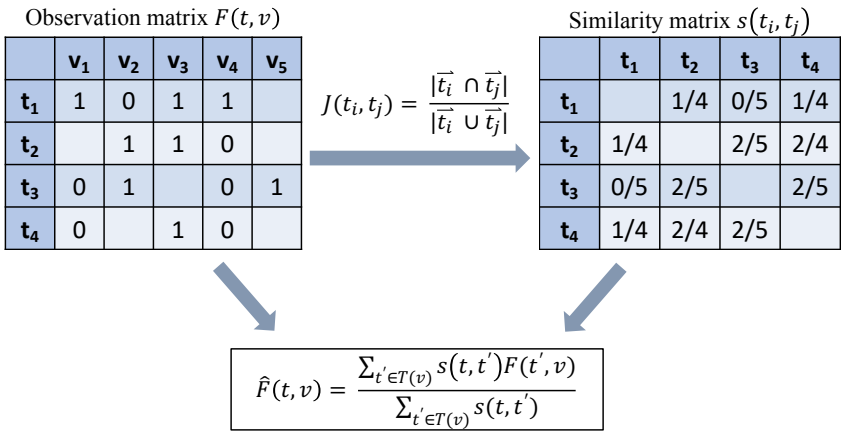

Figure 1. Concept of calculating performance estimate via user-based collaborative filtering

In this example, there are 4 tasks and 5 nodes in the system. The left-hand side table in the figure shows a matrix of each task t_i's observed performance on node v_j it is assigned to at the cell at i-th row and j-th column. Here the performance numbers are binary signals: 1 indicates 'good' performance and 0 indicates 'bad' one, while blank cells indicate unobserved task-node pairs.

The user-based ('task-based' in our case) CF works by first establishing the similarity between the tasks in terms of the commonality in their performances on the nodes. There are various similarity measures that could be used, but in our example, we use a Jaccard coefficient which is simpler to explain as we have only a binary indicator (1 or 0). The Jaccard coefficient measures the similarity between two finite sets, defined by

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

between two sets A and B. Using this measure, the similarity between two tasks t and t', denoted by $s(t,t')$, is calculated by dividing the number of nodes that the two tasks have the same performance by the total number of nodes on which the performance have been observed for either task. For instance, $s(t_1,t_2) = 0.25$ as the performance on 4 nodes (v_1, v_2, v_3, and v_4) have been observed for either task and the performance is the same for both tasks on one of them (v_3).

Once we have obtained the similarity between all pairs of tasks (the right-hand table in Figure 1), we now can calculate the predicted performance of unobserved task-node pairs. Specifically, the predicted performance $\hat{F}(t,v)$ of task t's performance on node v is given by the normalized weighted sum of the measured performances of the other tasks on v, where the weights are the similarity between t and other tasks; that is,

$$\hat{F}(t,v) = \frac{\sum_{t' \in T(v)} s(t,t')F(t',v)}{\sum_{t' \in T(v)} s(t,t')},$$

where $T(v) \subset T$ is the set of tasks whose performance on node v have been observed. For instance,

$$\hat{F}(t_4,v_2) = \frac{\frac{1}{4}\times 0 + \frac{2}{4}\times 1 + \frac{2}{5}\times 1}{\frac{1}{4}+\frac{2}{4}+\frac{2}{5}} \approx 1.48.$$

While we used the Jaccard coefficient as our similarity measure in the above example, there are other similarity measures that can handle continuous performance numbers more adequately. In particular, *Pearson correlation coefficient* and *vector cosine similarity* are commonly used in CF as they can provide effective measures between the items or users even when there are user biases, that is, some users tend to give higher ratings to the items than the others. In our context, it is reasonable to assume that such a heterogeneity *does* exist as the criteria for measuring the performance can very well be different for different tasks or the applications can have different utility functions from each other; e.g., some applications

are concerned more on reducing the response time metric, while some others prefer to maximize the throughput or minimize the task completion time.

3.2 Scalable CF Model for Performance Prediction

The basic CF approach described above has several limitations:

- When the number of tasks and nodes is large, it requires the evaluation of the similarities and predicted performance scores from a large number of parameters—$O(mn)$ parameters for m tasks and n nodes. The number will increase even larger when we consider the variations of the tasks and nodes with their different profiles as separate instances.
- It determines the predicted scores of unobserved samples via simple linear combinations of the observed ones. Hence it cannot effectively capture complex, non-linear relationship between the tasks' performance on a variety of nodes.
- The performance observations can be very sparse at the beginning, an issue called "cold start". When the number of tasks and nodes is large, there can be no or very few observation of the performance of some tasks, rendering the performance prediction using cross-similarity measures unreliable or even infeasible.

In the literature of CF, *matrix factorization* approaches have been widely employed to address the first issue. In general, a matrix factorization (MF) method works by decomposing a given matrix $X \in \mathbb{R}^{m \times n}$ into two matrices $U \in \mathbb{R}^{m \times k}$ and $W \in \mathbb{R}^{k \times n}$, whose matrix product UV approximates well the original matrix X, i.e., $X \approx UW$. The parameter k is usually selected to be much smaller than both m and n. Factorizing the matrix in this way has the effect of reducing the number of parameters representing the original matrix ($m \times n$) to a smaller one ($mk + kn$) if $k \ll m$ and $k \ll n$, while the values of the elements in X can be reconstructed (in estimate) by taking a dot product of two corresponding vectors in U and W; That is, $x_{ij} \approx < u_i, w_j >$, where x_{ij} is the element in i-th row and j-th column of X, u_i the i-th column vector of U, and w_j the j-th row vector of W. Note that this concept is similar to sparse signal processing techniques where a signal of very high dimension is compressed into a much smaller vector, yet the compressed vector after efficient processing can be used to closely approximate (and exactly in some cases) the original signal. This connection may be worth exploring in the future.

In the context of predicting performances of task-to-node assignments, MF provides a way of finding the estimate of $\hat{F}(t, v)$ via a simple dot product of two low-dimensional vectors in the two decomposed matrices, one representing the "task" matrix and the other the "node" matrix. This provides much better scalability than calculating the weighted sum of all observed performances with similarity measure.

An interesting aspect of the matrix factorization is that the resulting column- and row-vectors in U and V can be viewed as vector embedding of the tasks and nodes, respectively, to a low-dimensional latent space with respect to their 'affinity' with one another in collaborative filtering sense. Such an embedding facilitates other types of analytics on the tasks and nodes now that they are mapped into the same vector space. For example, one can perform nearest neighbor search in the latent space based on a certain distance measure (e.g., cosine distance) to find the best (e.g., top-N) candidate nodes for assigning a given task to.

While MF addresses the scalability issue, it falls short of providing effective means to resolve the second and the third issues above, for which we propose to use the combination of the followings:

(i) Learn the task- and node-embedding into a latent vector space using neural network, and

(ii) Use of side information regarding the tasks and the nodes to associate them with others at the lack of sufficient number of performance observations.

The rationale of using neural networks, especially deep learning architectures, is that they are effective in discovering non-linear relationship between the features in raw data without the need of extensive feature engineering efforts, as proven in a variety of data analytics and machine learning problems (images, speech, videos, etc.). For modeling the performance prediction for task-node assignment, we use the neural networks to find the embedding of tasks and nodes into a fixed

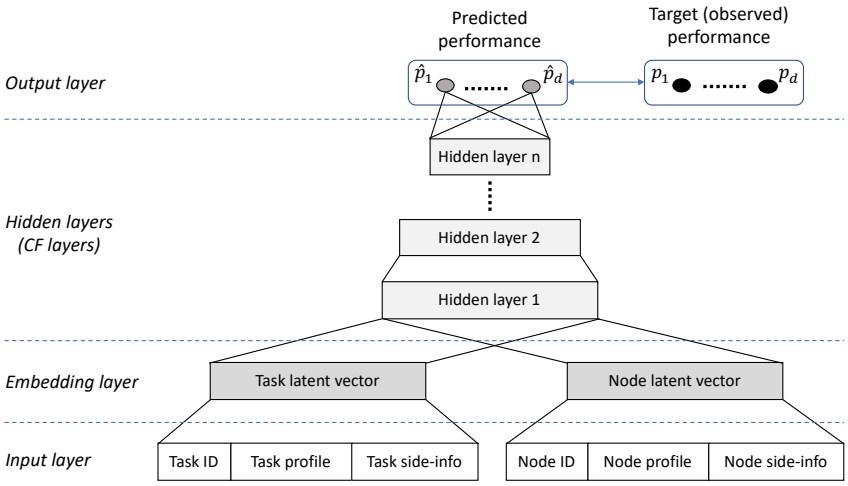

Figure 2. Learning tasks' and nodes' latent vector embedding through a variant of Neural Collaborative Filtering network.[4]

length latent vector space, like what matrix factorization does, but in a manner that can additionally take into account the non-linear interaction between tasks and nodes.

The side information refers to any data or meta-information that 'describes' the individual tasks and nodes and can help establish the similarity between the tasks or between the nodes. Examples may include the textual description of the application, version/revision information, performance metrics of the tasks in a known testing environment, meta-information above the devices (e.g., manufacturer, model number, OS version, etc.), static device profiles (e.g., number of cores, network interfaces, total memory size), etc.. Methods of using such side information in recommendation systems are generally referred to as content-based filtering, and can help overcome the sparsity of known data about user-item interaction in collaborative filtering.

More specifically, we propose to use a deep neural network (DNN) architecture in Figure 2, which is a variant of Neural Collaborative Filtering (NCF) network.[4] The inputs to the network are the raw vectors describing the tasks and the nodes; For the tasks, this would consist of the concatenated vector of task ID, the task profiles, and the vector of task's side information. Similarly the input vector of a node would consist of node ID, the resource profile, and its side information. We assume any variable size input, such as textual description in the side information, is mapped into a vector of fixed size (through, e.g., recurrent networks) before being fed into NCF network.

The first layer is the *embedding layer*, in which the input vectors of tasks and nodes are separately mapped into k-dimensional vectors, providing dense representations of the inputs. The upper, *hidden layers* (called "collaborative filtering layers" in NCF) combine the embedding layers' output into a single neural network, and are used to discover the non-linear relationship in the task-node assignments. The output layer is the *performance layer*, which consists of the performance vector $< p_1, \cdots, p_d >$ resulting from the given task-node assignment. In addition, another layer, called the *utility layer*, can be stacked on top of the performance layer into a single-value output, representing the utility function $U^t(p_1, \cdots, p_d)$ if it is defined in the system. In all layers, we use fully connected networks followed by a non-linearity such as ReLU (Rectified Linear Unit) to discover the cross-correlation across all features of the input data. In the future, however, more efficient network architectures like convolutional network may prove to be advantageous if a proper hypothesis on localized interaction between the input features can be established.

At the training phase, the observed task-to-node assignments and their side information are fed into the input layer as the training samples, and the output vector (or the utility function) is produced in the forward pass through the network. The standard back-propagation is performed in the backward pass, based on the MSE (Mean Squared Logarithmic Error) loss function:

$$\mathcal{L} = \frac{1}{d}\sum_{i=1}^{d}(\hat{p}_i - p_i)^2,$$

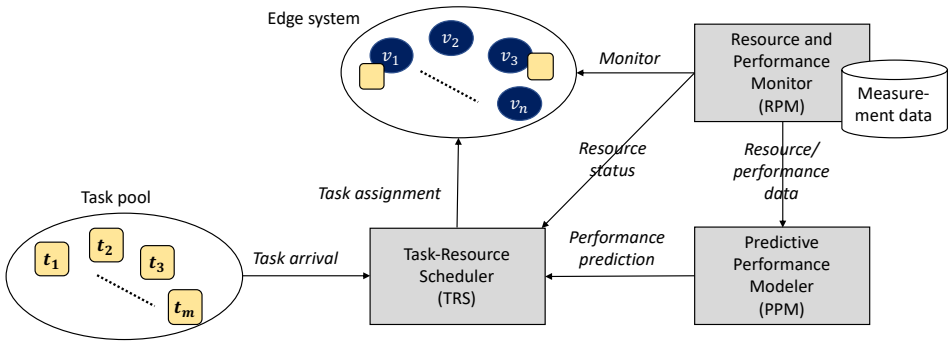

Figure 3. Task-to-node assignment architecture

where $\hat{p}_i$ is the i-th element of the performance layer output vector and p_i is the corresponding element of the target (i.e., observed) performance metric vector (If the utility layer is to be used, the loss function would simply be the square of the difference between the output utility and the observed utility).

There are essentially two ways of using the network trained in the above manner. The first is to directly use the output layer values as the performance prediction of a given task-to-node assignment. This, however, may result in a large error when measured in the absolute performance number due to, say, convergence to a local minimum. Alternatively, the performance estimate of any unobserved task-node assignment can be obtained by taking the dot product of the two k-dimensional vectors in the embedding layer (the task vector and the node vector) as with MF approaches. In the bigger contexts of assigning the tasks to nodes, however, we propose the prediction results be used indirectly as a means to generate the candidate node set for task assignment by evaluating their relative performance predictions through, e.g., nearest neighbor search.

4. RESOURCE ALLOCATION

In this section, we present how the CF-based predictive model described in the previous section can be used for performing the task-to-node assignment. We begin with a general architecture for the assignment, and then present specific assignment strategies.

4.1 General Architecture

In the previous section, we assumed the nodes' resource profiles are static for the purpose of describing the CF-based predictive model. In practice, however, they are fairly dynamic because a node's resource utilization changes whenever the tasks running on it changes, e.g., a new is assigned or an existing one leave. From the perspective of finding good task-to-node assignments, an obvious pitfall of assuming the static resource profiles of nodes is that a single node could be recommended to multiple tasks at the same time based on the given resource profile, with each task realizing only later that the node's resources are to be shared with other tasks.

In order to avoid such a situation, we use an online, sequential assignment, as opposed to the offline, batch scheme that was implied in the CF method description in the previous section. We assume the followings:

- The tasks arrive in sequence: $t^{(1)}, t^{(2)}, \cdots$, where each $t^{(i)}$ is a sample from a population of the tasks T.
- The resource profile of a node changes only when a new task is assigned to it or an existing task leaves it; any intrinsic fluctuation in between these events is assumed to be averaged out and represented as fixed profile for that particular period.
- When assigned to a node, each task is allocated a slice of the resources of the node (e.g., a portion of CPU cores, memory, and/or IO bandwidth) for its exclusive use. In practice, this could be measured as the average resource utilization by the task in the node. The available (remaining) resource profile of a node can be obtained by subtracting the aggregate resource utilization by all tasks on the node from the total amount of resources.

Figure 3 shows a high-level architecture for the online resource allocation, in which there are three components:

- **Resource and Performance Monitor (RPM)** measures the performance profile of each task when it is assigned to a node, and monitors the current available resource profile of each node. The updated information about the available resources and the task's performance on the allocated slice of the resources is stored in a database, and provided to the Predictive Performance Modeler and Task-Resource Scheduler.
- **Predictive Performance Modeler (PPM)** builds and re-builds the CF-based performance prediction model using the history of measurement data provided by RPM on the tasks' performance profiles and resource profiles. The model update is performed periodically with a fixed period of time or upon the arrival of every *n*-th new information. It also provides to the Task-Resource Scheduler the recommendation of the candidate node set for a given task, determined by the predicted affinity of the task to the nodes that CF-based model produces (e.g., top-N nearest neighbors in the latent vector space).
- **Task-Resource Scheduler (TRS)** performs the assignment of the tasks to the nodes whenever a new task arrives. It takes into account the candidate node sets recommended by PPM as well as the current available resource profiles of those nodes.

This architecture enables the task-to-node assignment based on (i) the up-to-date information about the resource profiles and (ii) the predictive model updated based on the history of previous assignments. In the following we present viable approaches to the actual strategy that TRS can employ.

4.2 Task-Resource Scheduling Strategies

Recall that the performance prediction by the CF-based model is only an estimate and subject to a large error (in their absolute values) especially when sufficient amount of historical data is not available, rendering it a fairly risky exercise to perform the task assignment solely based on the predicted performance (One can argue the outcome of CF should be used only as a measure of *relative* score between the items). Therefore, we use the performance prediction model to derive the *candidate set* of nodes for each task, and make the final decision of the assignment considering the current (and possibly historic) resource status of the candidates. We propose two specific assignment strategies, one designed toward optimizing resource utilization, and the other toward optimizing task performances, per the goals set in Section 2.

The first strategy is the statistical load balancing, with which each task is assigned to a node selected probabilistically from the candidate node set. A simple way is to select one uniformly at random, which basically corresponds to the balls-in-bins model[5]—this is the most applicable when checking the current load of the servers is an expensive operation; an improvement in terms of the expected maximum load is shown to be achievable by a less expensive power-of-two-choices strategy.[6] In practice, the assignment of the selection probabilities among the candidate set can also take into account their predicted performance and current resource availability together, e.g., with the following probability for selecting node v_j for task t_i:

$$P(t_i \to v_j) = \frac{\hat{u}_{ij} r_j}{\sum_{v_k \in V_i} \hat{u}_{ik} r_k},$$

where V_i is the candidate node set for task t_i, $\hat{u}_{ij}$ is the predicted utility function of t_i on v_j, and r_j the current available resources of v_j. These methods would help achieving the goal of resource allocation from the perspective of optimizing the resource utilization, especially helping achieve fair resource utilization. The beauty of these statistical selection methods lies in their simplicity, yet a reasonable degree of performance can be expected (in some cases guaranteed in a probabilistic sense.[5,6])

The second strategy is to use the reinforcement learning (RL). Various RL algorithms have been popularized as solutions to many problems involving the control of systems where the signals indicating the outcome of the control are only sparse. At a high level, an RL method finds the optimal action, denoted by π^* (also called the optimal policy) at each state of the system, that maximizes the expected long-term return with discount (called "value"), in environments where an action incurs the change in the system state by a Markov Decision Process (MDP) and an instantaneous reward from the system is observed after each action. There are three broad categories of RL algorithms: (i) Value-based, in which the optimal policy is determined by learning the expected return when following each action at each state, (ii) Policy-based, in which the optimal policy (typically a parameterized policy) is searched directly from the repeated experiments of action-state-reward, and (iii) Model-based, in which a simulated target system is used to perform the experiments. Recently a variety of deep reinforcement learning (DRL) methods have been successfully applied for the target systems that are too complex to be modeled after MDP.[7]

RL and DRL have been proposed for many resource management problems as well, e.g., autonomous tuning of application configurations,[8] allocating resources for the jobs in data center environments,[9] and device placement for neural network training and inference jobs.[10] Considering the representational power of the deep learning models, DRL could certainly be a promising approach to our problem of dynamic edge resource allocation. The problem is, our environment of edge computing provides an additional challenge that hinders the straightforward application of DRL: it can be very costly to run the experiments that can be used for exploring the vast action space because it would require gathering measurements of tasks actually deployed on the real resources.

Using the performance prediction model, more specifically the candidate node set for assignment, as a complementary element in RL-based resource allocation can alleviate this issue substantially. Specifically, the candidate set generated by our predictive model would provide a "guideline" in the exploration of the action space for the RL algorithms, effectively directing the search for the optimal policy towards a good optimum via importance sampling—Such an idea is termed "guided policy search (GPS)" in RL literature.[11] In general, RL-based methods would be effective in achieving the goal of maximizing the tasks' performances.

5. CONCLUSION AND FUTURE WORK

We have presented the design of a machine-learning-based method for a dynamic resource management problem in edge computing environments. The problem is explored in the context of assigning heterogeneous application tasks to the edge nodes whose resource availability fluctuate over time. The proposed method is based on building a predictive performance model of task-resource allocation in the collaborative filtering framework, and a specific model using a deep learning architecture is proposed to learn and predict various, non-linear relationship present in the task-to-resource assignments. The predictive model is then utilized as a recommendation engine that generates a candidate set of the target resources for the task assignments. We also present the architecture for the resource allocation, and further propose two specific assignment strategies based on the statistical load balancing and the reinforcement learning, both assisted by the predictive model.

Besides the comprehensive exploration and evaluation of the proposed methods, there are other issues that merit further investigation. One of them is on how to collect a sufficient amount of training data for building (deep) predictive models in an environment where measuring the performance and resource consumption data is a costly operation. An idea is to obtain multiple data points from a single experiment of task-to-node assignment by identifying different phases of the resource utilization and performance in the task's execution run (hence boosting the number of data samples). Another approach would be to use short experimental runs of task-to-node assignments and extrapolate the performance profile based on the correlation between the results from short experiments and the long-term performance metrics (hence reducing the experimental data generation overhead). Data generation and synthesis using GAN (Generative Adversarial Network)[12] is also worth exploring.

Another issue that has not been addressed is how to allocate resources for inter-dependent tasks in a distributed application, for which the resource allocation needs to made for a collection of tasks into a collection of resources, rather than per-task, per-resource basis. As such distributed applications are typically modeled by graphs, the challenge is then how to represent both the application graphs and the physical resource graphs in a manner that machine learning approaches can be effectively applied. This entails to a problem of finding the embedding of these two graphs into a latent vector space with high level of representational power. A potential approach would be to exploit recently proposed graph embedding techniques.[13,14] We plan to investigate these issues in the future.

ACKNOWLEDGEMENTS

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copy-right notation hereon.

REFERENCES

[1] Hou, Yun, et al. "On the mapping between logical and physical topologies." Communication Systems and Networks and Workshops, 2009. COMSNETS 2009. First International. IEEE, 2009.

[2] Alicherry, Mansoor, and T. V. Lakshman. "Network aware resource allocation in distributed clouds." Infocom, 2012 proceedings IEEE. IEEE, 2012.

[3] Goldberg, David, et al. "Using collaborative filtering to weave an information tapestry." Communications of the ACM 35.12 (1992): 61-70.

[4] He, Xiangnan, et al. "Neural collaborative filtering." Proceedings of the 26th International Conference on World Wide Web. International World Wide Web Conferences Steering Committee, 2017.

[5] Raab, Martin, and Angelika Steger. ""Balls into bins"—A simple and tight analysis." International Workshop on Randomization and Approximation Techniques in Computer Science. Springer, Berlin, Heidelberg, 1998.

[6] Mitzenmacher, Michael. "The power of two choices in randomized load balancing." IEEE Transactions on Parallel and Distributed Systems 12.10 (2001): 1094-1104.

[7] Arulkumaran, Kai, et al. "A brief survey of deep reinforcement learning." arXiv preprint arXiv:1708.05866 (2017).

[8] Tesauro, Gerald. "Reinforcement learning in autonomic computing: A manifesto and case studies." IEEE Internet Computing 11.1 (2007).

[9] Mao, Hongzi, et al. "Resource management with deep reinforcement learning." Proceedings of the 15th ACM Workshop on Hot Topics in Networks. ACM, 2016.

[10] Mirhoseini, Azalia, et al. "Device placement optimization with reinforcement learning." arXiv preprint arXiv:1706.04972(2017).

[11] Levine, Sergey, and Vladlen Koltun. "Guided policy search." International Conference on Machine Learning. 2013.

[12] Goodfellow, Ian, et al. "Generative adversarial nets." Advances in neural information processing systems. 2014.

[13] Grover, Aditya, and Jure Leskovec. "node2vec: Scalable feature learning for networks." Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2016.

[14] Narayanan, Annamalai, et al. "subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs." arXiv preprint arXiv:1606.08928 (2016).