
Proceedings Paper:
Yang, R, Ouyang, X, Chen, Y et al. (2 more authors) (2018) Intelligent Resource Scheduling at Scale: a Machine Learning Perspective. In: IEEE International Symposium on Service Oriented System Engineering. 2018 IEEE SOSE, 26-29 Mar 2018, Bamberg, Germany. IEEE , pp. 132-141. ISBN 978-1-5386-5207-7

https://doi.org/10.1109/SOSE.2018.00025

Intelligent Resource Scheduling at Scale: a Machine Learning Perspective

Renyu Yang[1,2], Xue Ouyang[1,3], Yaofeng Chen[3,1], Paul Townend[4], Jie Xu[1,2]
[1]*University of Leeds* [2]*BDBC, Beihang University* [3]*NUDT* [4]*Edgetic Limited*
{*r.yang1, scxo, y.chen2, j.xu*}*@leeds.ac.uk; paul.townend@edgetic.com*

***Abstract*—Resource scheduling refers to the problem of packing tasks with multi-dimensional resource requirements and non-functional constraints. The exhibited heterogeneity of workload and server characteristics in Cloud-scale or Internet-scale environments has raised unprecedented new challenges for cluster scheduling. Compared with ad-hoc heuristics for a multi-resource cluster scheduling problem, machine learning (ML) approaches can in turn facilitate improved efficiency in resource management. In this paper, we describe and discuss how ML can be used to autonomously exploit and understand both workloads and environments, and to learn how to efficiently deal with scheduling problems such as consolidating co-located workloads, handling resource requests, guaranteeing applications' QoS, and mitigating tailed stragglers etc. The scheduling procedure can be fundamentally learned from experience rather than interference by human subjectivity. Additionally, we present a generalized ML-based solution and demonstrate its effectiveness through a case study of the performance-centric node classification and straggler mitigation method. We believe that the rethinking from a ML perspective can steer the architecture optimization and efficiency improvement.**

1. Introduction

In the past decade, Cloud computing has become an indispensable part of information technology, satisfying increasing demands for Internet services such as web search, social networking, and machine learning applications. Typically, large-scale Cloud data centers consist of hundreds of thousands of heterogeneous machines to provision reliable computing and storage services to customers. Through virtualization and container technology, multiple tenants are enabled to share data center resources and services by co-locating latency-sensitive applications or services encapsulated in virtual machines or Docker containers. Meanwhile, big data jobs such as batch processing or streaming are also submitted into shared production cluster environments. The exhibited heterogeneity of workload characteristics such as task scale, execution time and resource usage pattern [1][2] have raised new cluster scheduling challenges in terms of performance interference, resource utilization, power consumption, system resilience etc.

Resource management plays a fundamentally important role in assigning many kinds of resources such as CPU, RAM, network, disk or device I/O to applications within cluster systems or ubiquitous computer systems. Modern cluster management systems ([3] [4] [5] [6] [7]) are designed to effectively allocate jobs onto machines and manage their respective resource requirements. To maximize utilization whilst guaranteeing application's QoS, it is greatly desirable for the scheduler to take additional control of resource allocations, task placements, and fault-tolerant executions, etc. The pursued objectives vary according to different scenarios – balancing system loads, maximizing the provider's revenues, minimizing response time, saving power consumption, etc. Each of these problems can be solved by designing heuristics. Plausible heuristics are initialized followed by fine-grained tuning to reach an acceptable performance level; wherever we use the heuristics, we can leverage machine learning techniques to improve the accuracy and effectiveness of sophisticated decision making. Compared with ad-hoc heuristics, machine learning approaches can benefit the system with intelligent resource allocation, choosing the most suitable action based on contextual states and environmental factors. In effect, Machine learning (ML) is on the cusp of its revolution, with core AI algorithms and frameworks[8][9][10] proposed at an unprecedented speed. For example, Neural Networks (NN) have been adopted into data center management and reduced the overall cooling bill of Google data centers by 40% [11]. Deep Learning (DL) is being explored at the edge of the network to reduce the amount of data propagated back to data centers.

In this paper, we discuss how ML can facilitate resource management in massive-scale distributed systems. We firstly present a deep-dive overview of current research problems in the resource scheduling domain and the major challenges that have yet been solved. Such problems are sub-categorized in terms of scheduling effectiveness and efficiency. We then present a generalized ML-based solution and architecture through comprehensive problem formalizing, data driven profiling and supervised learning modeling. Finally, we introduce our preliminary exploration towards leveraging ML in specific resource scheduling problems – performance-centric node classification and the subsequent straggler mitigation, and use it as a case study to demonstrate the effectiveness of intelligent scheduling.

The rest of this paper is structured as follows: general resource scheduling problems at scale are discussed in Section 2 and Section 3 describes how and where machine learning can benefit resource scheduling. An overview of solutions is given in Section 4 followed by a detailed

discussion of useful ML models in Section 5. We depict the case study in Section 6 and then conclude with future work.

2. Resource Scheduling at Scale

Massive-scale distributed computing for data analysis workloads has been widely discussed in recent years and is becoming increasingly common. High operational and maintenance costs within heterogeneous workload management and resource allocation lead to a philosophy of sharing data center cluster resources among diverse computation frameworks ranging from batch jobs to long-running services. Scheduling such diverse workloads is inherently complex and difficult, especially as computing cluster size grows rapidly. This section firstly overviews the most fundamental problems in resource management and scheduling.

2.1. The Placement Problem

Job Scheduling and Machine Selection. The primary role of computing clusters at Internet scale is to manage the life-cycles of tasks and machines. Task scheduling or placement refers to the assignment of tasks to machines. Inversely, the scheduler finds a set of machines that meet the task's constraints whilst having sufficient resources. In this procedure of machine selection, the scheduler has to determine the most suitable machines driven by scoring or ranking, taking into account factors such as machine or application behaviors [12], failure manifestations [13] [14], energy efficiency [15] [16] [17], etc.

Workload Co-location. Cloud data centers are multi-tenant environments where diverse workloads live together. Normally encapsulated into Virtual Machines (VMs) or Docker Containers, such workloads are co-located into the same servers sharing the underlying physical infrastructure to maximize the data center utilization. However, interference among co-located workloads may lead to performance degradation. In this context, several problems are raised in terms of co-location algorithm, preemption mechanism and queue management – which tasks or VMs should be co-located on the same machine? Which tasks should be preempted under prioritized tasks, limited resource capabilities, and process switch cost etc. Which tasks should be considered first if a unit of resource becomes available? Additionally, applications' tolerance to interference should be elaborately profiled and identified. For example, deadline constrained applications are not severely affected by transient resource shortages, while latency-sensitive or interactive applications will be tremendously influenced.

Service Composition. The service-oriented architecture packages functionality as a suite of inter-operable routines that can be used within multiple, separate systems from several business domains. These loosely-decoupled functions or APIs can be accessed via pre-defined interfaces over the network, which enables re-use and composition to form a chain of applications. At present, such functional services can be leveraged as fundamental appliances and composed in a mash-up style to control development cost and maintenance pressure. Service composition or orchestration is a key concept within distributed systems, enabling the alignment of deployed applications with users' business interests. In this context, according to the service topology, the composer has to choose and bind a suitable service instance for each component from the candidate set, and orchestrate all components to provide the required features, meet non-functional QoS constraints such as overall cost, latency, reliability, and response time etc. [18] [19].

2.2. Scheduling Efficiency and Optimization

Resource Sharing. Multi-level scheduling architectures are adopted by modern cluster management systems [3] [4] [5] [20] [7] [6]. They underpin diverse workloads through resource negotiation with a central or decentralized resource manager. Meanwhile, DRF[21], capacity scheduling[22] or fairness scheduling[23] are used to fulfill an efficient quota-based resource sharing among multiple jobs. Their main objective is the enforcement of scheduling invariants for heterogeneous applications to prevent excessive resource occupation.

Resource Utilization. A long-standing challenge in cluster scheduling is the ability to effectively improve the utilization of heterogeneous resources. Users are typically conservative and over-provision resources for their bursting requirements to avoid SLA violation. Over-provisioning resources for performance sensitive applications can reduce the risk of such violation[24]. However, this typically results in low machine utilization. Also, almost all production jobs are generated by higher-level abstractions(e.g., Hive[25] or Pig[26] etc.). Therefore, such frameworks typically depend on coarse-grained resource models to uniformly guarantee jobs' runtime execution, ignoring the varying input data size and resource requirements. In production systems, reservation based resource request and the over-estimation phenomenon is very common[3][4][5]. In fact, resource consumption depends on many factors such as dataset size, parallel degree, operators etc. Accordingly, unless a precise resource estimation model can be produced, no one would like to take risks in violating QoS and even failing tasks. It is extremely challenging to estimate resource requirements accurately, rather than simple resource reservation. To mitigate this, overbooking (also known as overselling or oversubscription) [27][28][29][30] is often used as an alternative way to compensate the lowered utilization produced by overestimation.

Resource Contention and QoS Guarantee. Resource over-allocation is widely used for latency-sensitive applications to guarantee QoS. However, there is a trade-off balance between improved cluster utilization and increased resource contention level among co-located applications. Due to the inherent sharing and resultant contentions, the noisy neighbor phenomenon becomes a norm. One of the challenges is precise QoS prediction and management for latency-sensitive applications, and this procedure needs to precisely capture, depict and predict QoS interference and degradation considering multi-dimensional resource types

such as CPU, memory, storage, IO, etc. Algorithms should be carefully designed to effectively improve server utilization while enforcing QoS requirements, without causing significant QoS degradation.

Datacenter Energy Efficiency. The primary goal is to make efficient use of machines and reduce the energy consumption due to resource contentions. Arguably, this action represents a significant investment; millions of dollars can be saved by increasing utilization by only a few percentage points. Cloud providers still need to address a number of key challenges, such as striking a balance between optimal energy efficiency and satisfying increasing demand and the high performance expectations of users. Understanding the impact of performance interference on a datacenter's energy-efficiency is also very critical if we are to design energy-efficient mechanisms that maintain performance under realistic environmental conditions[15].

Straggler Mitigation. Due to factors such as machine heterogeneity, resource contention, and data skew, parallelized tasks running on large-scale systems exhibit varied execution performance and duration [31] [32] [33]. The straggler problem occurs when a small proportion of these tasks experience abnormally longer execution compared with other sibling tasks from the same parallel job, leading to extended job completion time. A speculative execution scheme is proposed to deal with the straggler problem [34], which creates redundant task replicas for identified stragglers. However, its performance can be undermined due to the fact that speculative copies could occupy available resources for new tasks. Therefore, the most critical challenges for straggler mitigation are to accurately identify the most needed stragglers for mitigation, and to determine when and where to launch the replica tasks.

2.3. Challenges

The fundamental problem in resource management and job scheduling is the match-making between available resources and resource requests from waiting jobs or applications. With the large number of resources available and the network condition variability in Cloud datacenters, the computational challenges will become increasingly intricate:

- The system should fully exploit the diversities and dynamicity of computing clusters at massive scale to improve job throughput, reduce the occurrence of task eviction, and autonomously handle component failures. Considering workload and server heterogeneity[1][35] is extremely important when conducting scheduling. Such heterogeneity leads to different resource capacities and unique machine characteristics.
- For online decision making, real-time or sometimes faster-than real time is urgently required. We have to find out the machines that are determined to be most suitable for specific purposes such as workload consolidation, speculative task launching, precise resource oversubscription etc. This can be done through a filtering process that comprehensively considers estimated load, correlative workload performance, and queue states. Only through recognizing the accurate targets for placement can the scheduler mitigate the computation straggler or promote resource utility and compaction.
- Performance issues are very difficult to cope with in a white-box or top-down way due to the intricate factors and the complicated combinations that might influence the results. Therefore, learning-based approaches are especially best fit to resource management systems, where decisions are often repetitive and the generated training data can be abundantly re-used.

3. When Scheduling Meets Machine Learning

3.1. Why to Use ML

Machine Learning (ML) is a field of computer science that enables computers to learn and solve problems without being explicitly programmed. Compared with direct human approaches, ML approaches not only significantly reduce human labor and time, especially when solving complex problems, but also are capable of dealing with multi-dimension and multi-variety data in dynamic or uncertain environments. There are several characteristics that can be greatly harnessed within resource scheduling:

Automatic Feature Learning and Selection. One of the most important advantages of machine learning is the capability of learning proper features after random initializing and training on given datasets. At present, machine learning can be used to discover relevant features in disordered datasets, substituting manual feature selection by domain experts or technical staffs. The features that are demonstrated relative to scheduling can be very powerful and critical in targeting the optimal scheduling. Even a large group of parameters can be automatically extracted through a deep neural architecture. It is infeasible for people to find such an optimal setting for large number of parameters manually.

Accurate Prediction and Inference. Due to the fact that system utility is highly dependent on the prediction accuracy of resource, during resource mapping and scheduling, accuracy is always the top concern. In ML fields, accuracy – one of the performance indicators – is used to evaluate the differences between the trained model and the real model. Currently, there are a huge number of classical models such as decision tree, support vector machine and neural network etc. that have been illustrated to be accurate enough in specific scenarios [36] [37] [38]. There are many considerations when choosing an algorithm, based on the requirements including accuracy, training time, number of parameters, number of features, and the relationship between variants. However, accuracy is not always the necessity. For example, even by using approximate scheduling, scheduling quality can be further calibrated by tuning and rescheduling at runtime, given that the re-adjustment overhead is acceptable.

3.2. Where to Use ML

Some of the aforementioned problems are solved by designing heuristics: a plausible heuristics for a simplified

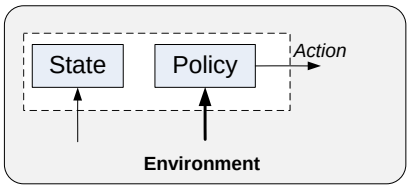

Figure 1. A formalized concept

model is first initialized, then followed by fine-grained tunning to reach an acceptable performance level. However, we believe that wherever we use heuristics, we can leverage machine learning techniques to improve the accuracy and effectiveness of decision making. Based on customized assumptions, performance metrics or system indicators can be outlined and formulated. Afterwards, heuristic rules can be proposed based on such indicators. Many examples demonstrate how rule-based scheduling is implemented: Interference-aware VM placement in the data center can fulfill workload consolidation with minimized performance degradation and resource contention; security-aware microservice orchestration can be used to minimize the overall discrepancy between user security requirements and actual system capacity.

The most problematic thing involved in the heuristic method is to continuously tune parameters and validate until a satisfactory model appears. However, when the workload or environment dynamically changes, the procedure of tuning might be endless before convergence. To this end, data-driven analytics and machine learning based approaches can effectively steer the optimization design of a scheduling system with more precise indicators and reasoning inferences. Also, learning the relationship between the finished placement/assignment decision and the resultant effectiveness of the system states(such as overall throughput, scheduler's response time, application's QoS, and job makespan etc.) is the key for success in adaptivity and flexibility.

4. Solutions Overview

4.1. Problem Definition

Scheduling Behavior. As shown in Figure 1, the cluster system can be seen as an environment where different applications are running. The runtime status of alive machine resources and running tasks are monitored and collected as the observable states. The scheduler acts as the agent which interacts with the environment through taking actions according to the build-in policy. In this context, the policy is referred to the specific scheduling approach and the output actions such as resource assignment or process operators, resulting in the dynamic changes of states in the next time frame. The state can be referred to the current cluster state represented as a tuple $(\mathbf{AR}, \mathbf{RQ})$, where $\mathbf{AR}$ stands for the state matrix including the available resource amount of all dimensional attributes within each machine while $\mathbf{RQ}$ depicts the pending requests from different workloads. The dimensions may include CPU, memory, disk load, bandwidth or other non-functional attributes such as kernel version, clock speed, Ethernet speed etc. that can be specified by the scheduler [1]. During match-making, the state of available resources $\mathbf{AR}_i$ can be instantiated to represent the available amount on the i^{th} machine and $\mathbf{AR}_{n\times d}$ is the aggregated resource state set of all machines. Additionally, the resource request can be formalized as a 4-tuple including the basic resource slot (the demanded resource for each dimension), the number of slot, the preference of resource location, and other soft or hard constraints. The $\mathbf{RQ}$ is depicted as $[ResReq_1, \ldots, ResReq_q]$ where total q requests are suspended and waiting for scheduling. The core process of scheduling is to find a model or algorithm to output the assignment results. A general output of the policy is the probability of scheduling $\mathbf{RQ}_i$ to machine M_j.

TABLE 1. WORKLOAD CATEGORIES

	Virtualized Workload	**Big Data Workload**
Representatives	Virtual Machine; Docker containers	Map-Reduce Spark
Type	online + interactive	offline + batch
Sensitivity	latency-sensitive	deadline-constrained
Running Time	long-running (hours, days)	short-alive (second, sub-second)
Decision Impact	instant and short-term	delayed and long-term
Key Concerns	temporal QoS	end-to-end time
State Space	small	large

Workload. With big data processing demands soaring and service decoupling, there is a general manifestation that heterogeneous workloads (in terms of execution durations, resource patterns, etc.) run and operate in the data center cluster [39]. Herein, we make coarse comparisons shown in Table 1. Virtualized jobs are encapsulated inside VM or Docker containers, resulting in a relative guaranteed resource isolation. By contrast, the big data workload often exists as a native process which is directly controlled by the middleware daemon and maintained by *cgroups*. Due to the inherent interaction characteristic, VMs or Dockers usually stay alive and execute for a long period of time (typically a few hours or a couple of days), while tasks in big data scenarios are short-lived and only last for seconds and subseconds. Therefore, temporal QoS is critically concerning in the virtualized environment, since performance is very susceptible to fluctuations in traffic or allocated resources. On the contrary, the key indicator of batch workloads is the holistic guarantee of end-to-end performance. All these disparities need to be considered when selecting and designing different machine learning techniques into different scheduling problems.

4.2. Combining Profiling with Supervised Learning based Scheduling

For scenarios where training data is slowly generated and the states are numerable, we can leverage supervised learning such as classification (discrete variables) or regression (continual variables) with unsupervised techniques such as clustering, etc. to deal with the scheduling problem. In Supervised Learning (SL), the core goal is to reconstruct or

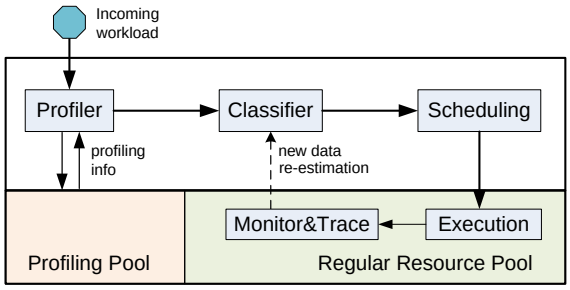

Figure 2. Profiling-based scheduling architecture

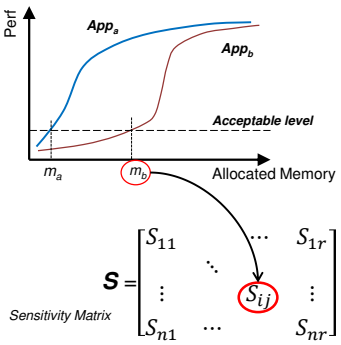

Figure 3. Relationship curves and profiling matrix

forecast the unknown function $\mathscr{F} : \mathbb{R} \rightarrow \mathbb{R}$ that assigns output values y to data points x, i.e., $y = \mathscr{F}(x)$. The value may include the tolerance degree of resource under-provision, sensitivity of interference in co-location circumstances, and the belonging category in terms of resource submission or workload scale etc.

Core Idea. Based on the profiling information from the previously scheduled workloads, the application-specific preferences such as resource amount or location preferences of an incoming workload can be accurately generated without a detailed priori analysis that will be repeated again. This methodology can rapidly indicate the performance levels according to the similarities with the existing workloads. Regarding the trained learning model, the incoming applications usually need to perform a series of operations (e.g., some matrix operations) based on input data, in order to infer possible output values. The procedure of value prediction literally functions as an indicator to imply a performance score or sensitivity level that further scheduling actions are directly dependent on. To achieve this, the architecture is shown in Figure 2. In the first phase, a profiling pool provisions the experimental testbed to detect and capture the tracelogs that are necessitated in the performance modeling. Subsequently, the resource scheduler can perform the match making based on the implications hinted from the classifier. During execution, the run-time adjustment can guarantee the QoS and task re-scheduling or migration may occur if necessary.

Profiler – Characteristics and the performance profiling. Due to the approximate equivalence between the resource limitation and the resource contention [40], we can use the resource throttling subsystems to limit the allocated amount of resources to the specific application for the purpose of resource pressure test. Basically, we would like to find the relationship between the allocated resource and the resultant performance such as the QPS and MPKI for the interactive application, or the slowdown/speedup to the makespan of batch jobs. For instance, we quantitatively characterize the performance impact by different resource dimensions such as CPU cores, memory, last level cache(LLC), disk IO, network bandwidth etc. The profiler executes the profiling for each application for a couple of minutes, collects the results of different experimental tests with variable values of each resource dimension, and then records the performance degradation caused by such resource bindings. Accordingly, the relationship curves can be generated and stored. The procedure is simply demonstrated in Figure 3. For the convenience of modeling and measurement, we can also extract the performance critical points to representatively indicate the *sensitivity level* of the given resource. Specifically, we extract the value if an acceptable level of the performance is targeted and put it into the sensitivity matrix $\mathbf{S}$ where $\mathbf{S_{ij}}$ represents the bounded sensitivity of i^{th} application to j^{th} resource dimension. It is noteworthy that different application may have differentiated sensitivity. For example, app_b is more sensitive to the memory allocation than app_a, thus with a larger m_b than m_a. That means that we have to spare more resource for app_b to guarantee its performance level. Equivalently, the lowered sensitivity implicates a strengthened tolerance level of such resources when co-allocating with other applications.

In light of the same principle, we can similarly generate the corresponding matrices to include the impact of application configuration and machine heterogeneity on the overall application performance or straggler behaviors. Therefore, in general, the feature vectors for all applications will constitute a full profiling matrix $\mathbf{P_a}$ considering app-specific configuration $\mathbf{C}$, machine heterogeneity's impact on application's performance $\mathbf{H}$, and the sensitivity to resource contention $\mathbf{S}$, etc.

$$\mathbf{P_a} = \mathbf{C} + \mathbf{H} + \mathbf{S} \tag{1}$$

Classifier – Supervised Learning for Value Prediction and Labeling. Based on the stored profiling curve set and full information $\mathbf{P_a}$, similarity-based classification methods [41] can be utilized to detect affinities among profiled applications and new ones. In particular, according to the sensitivity and server configuration, we can accurately determine how many resources should be at least requested for the given application and which machines with specific capacity or capability should be preferably prioritized. These inferences can achieve a safe but saving workload execution without performance degradation. Similarly, the status of machines and the pertaining tasks' status such as progress and behaviors (eviction, failures, stragglers, etc.) can also be captured, recorded, and profiled the same way as we did for applications. The profiling information can be represented

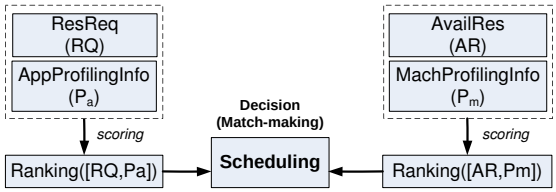

Figure 4. Basic scheduling workflow

by $\mathbf{P_m}$ where each row represents profiling value of a given machine.

Additionally, the classifier will also forecast the performance-related indicators (e.g., the operational scores based on different machine types and capacities, the sensitivity score to resource allocations, etc.) for the new, unknown application. Those supplementary indicators are significantly important for the scheduler to rank and sort out a series of questions such as which pending applications should be first handled, which servers are most suitable with sufficient resources and demanding attributes, and where to place the applications taking the sensitivity into account.

Scheduling. Intuitively, the profiling-based classification or prediction is also towards a heuristic objective. The forwarding step is highly likely to use a greedy or heuristic algorithm to rapidly obtain a sub-optimal solution which combines and leverages the learning results to select a group of machines or allocate the specific set of resources that best fit the pre-supposed constraints or preferences. The essence of scheduling is to make the placement decision – mapping between the $[\mathbf{RQ}^*, \mathbf{P_a}]$ and servers $[\mathbf{AR}, \mathbf{P_m}]$. The fundamental workflow of scheduling is depicted in Figure 4. For example, a general and exemplified scheduling problem is to firstly deploy a service instance into a VM or Docker container, and then find a physical machine with adequate resources to host those containers. More specifically, after obtaining a candidate instance h for service j from the candidate list H, and the instance will be deployed into an asset i (VM or docker container) which is hosted by physical machine k. The objective is to maximize a utility function ($UtilFunc$) that describe the scheduling direction – such as minimizing the interference whilst maximizing the resource utilization.

$$Sched(k,i,j,h)^* = \arg\max_{k,i,j,h} UtilFunc \tag{2}$$

Runtime Re-calibration. After the initial placement, the monitor in the regular resource pool will monitor application behavior and the scheduler will continuously improve the decision quality by leveraging new information and re-evaluating constraints. The use of task- and data-migration can also enhance the quality, with QoS strictly guaranteed. Meanwhile, the new profile information will be back-flowed to the profiling models. As a result, more accurate models will be iteratively generated to reclassify the workload, adjust the scheduling decision, and best fit the dynamic application change at runtime.

5. Useful Models

By leveraging the learning techniques, additional profiling information can be obtained through a series of clustering, labeling and classification. The unknown information within the incoming tasks' request and the environments' states can be collaboratively inferred and predicted. Such implications and interpretations can be extremely informative, thus should be fully exploited in the scheduling for improved scheduling quality. In this section, we briefly discuss the useful models that may be adopted in the proposed scenarios.

Decision Tree[36] is a general ML model where the type of input and output data can be categorical, binary and numeric value. Based on the maximum entropy or minimized mean square deviation, the decision tree iteratively selects a feature as the decision node, partitions training dataset and finally forms a tree structure model. Moreover, Gradient boosting decision tree (GBDT) [42] is a stagewise ensemble model that combines the decision tree with gradient boosting, and it can be generalized by optimizing the loss function. Specifically, at each stage m where $1 \leq m \leq M$, it is assumed that a better model F_m can be achieved by adding an estimator $h(x)$ to the last stage F_{m-1}. In effect, $h(x)$ is constructed by fitting new CART decision tree[43] to the residual. $y - F_{m-1}(x)$. Due to the reduced negative effect from incremental update, GBDT and XGBoost[44] can be used instead of the original DT in the online ML area. Especially in the dynamic environment where uncertain and unexpected system failures or outages frequently manifest, the static model is difficult to reach the accuracy requirement. In this case, GBDT can keep itself updated over the whole running period. For example, assuming the time period can be divided into equal length discrete slots $T = \{t_0, t_1, ..., t_N\}$, the base model $F_0(x)$ is initially trained by the pre-set profiles. Afterwards, at the time slot t_n $(0 < n \leq N)$, the specific CART decision tree $h_n(x)$ will be trained by the profile sampling at t_{n-1}. A stronger learner $F_n(x)$=$F_0(x)$+$h_n(x)$ can be consequently generated to forecast the matching between task and machine in t_n.

Neural Network(NN)[38] is inspired by the biological neural networks within brains. The connections between neural cells determine the knowledges. As shown in Eq.3, x is the output of last layer cell and y_j^T represents the weight of j^{th} connection. The b is bias and σ depicts the activate function which enables NN to learn the non-linear relationship. The aggregation of such information is then passed to the next cell.

$$G(x) = \sum_{j=1}^{N} a_j \sigma(y_j^T x + b_j) \tag{3}$$

The universal approximation theorem[45] proved that a simple neural network is able to represent a wide variety of interesting functions given appropriate parameters. Through transforms such as categorical input into multiple binary variable or numeric input into binary encoded string, NN can be widely applied into any kinds of mapping problems. The problem of resource assignment and allocation is typically a

multi-input and multi-out mapping problem. The expected resource utilization such as CPU, memory, disk, network will be determined by inputs such as resource requirement, dataset size, parallel degree, operators etc. Deep Neural Network(DNN) that consists of a multi-layer network with millions of weights are increasingly important especially in image and speech recognition areas. In DNN context, an optional loss function can be defined as follows:

$$Loss(y, y') = \sum_{i=1}^{n} f(y, y') \tag{4}$$

$$f(x, y) = \begin{cases} aMSE(x, y) & x > y \\ bMSE(x, y) & x < y \end{cases}, (a > b > 0) \tag{5}$$

where $MSE(x, y)$ represents the squared error and the loss function means the model will learn to predict the value y that approaches to the ground truth y'. Due to the fact that DNNs are dependent on the hand-crafted features, they can be widely used as function approximators. The parameters a to b can be fine-grained tuned according to the boundary constraints and customized QoS requirements.

Recurrent Neural Network(RNN) is a class of NN that can obtain strong outcomes on sequence modeling tasks whose current states are related to the previous ones. RNN can use their internal memory to process arbitrary sequences of inputs. Basically, each unit, representing a stage in the sequence, has its own input, output and memory cell.

$$h(t) = \sigma(Wx_t + Uh_{t-1} + b) \tag{6}$$

Eq. 6 shows how status is delivered between memory cell h from stage $t-1$ to t. The current status is formed by aggregating all weighted input Wx_t, weighted previous status Uh_{t-1} and bias b. To increase the non-linear characteristic, the activate function σ is also utilized. This can allow RNNs to break through the obstacle from the fixed input, thus is able to process arbitrary sequences of inputs and predict the output accordingly. GRU(Gated Recurrent Unit)[46], LSTM(Long Short-Term)[47] are proposed to further solve the vanishing gradient problem. The topological service composition and resource allocation can be naturally solved by such LSTM models where the structure connections between units are very close to the workflow-based orchestration.

6. Case Study: Intelligent Straggler Mitigation

In this section, we introduce our exploration towards leveraging ML in a specific resource scheduling problem – performance-based node classification and the subsequent straggler mitigation. The effectiveness of our preliminary work is validated through the OpenCloud cluster dataset [48].

To mitigate stragglers is one of the intricate challenges encountered by current parallel computing systems. Tasks, even with similar designed duration, can exhibit various execution time once running on different nodes due to heterogeneous capacity and resource contention. Also, since node performance is a dynamic attribute that changes over time, it is difficult to precisely estimate task durations and identify straggler without node performance knowledge. Under such circumstances, machine learning based methods can help with revealing the non-trivial correlation between task execution time and node-level statistics. To this end, we firstly attempt to analyze and predict the node's performance by exploiting the timing statistics and the resultant task execution. The profiled models can be utilized to design and implement efficient algorithm to mitigate stragglers.

6.1. Machine Learning based Node Analyzer

Our node performance analyzer consists of several steps: feature extraction, clustering and labeling, and classification.

Feature Extraction. We explored a series of features extracted from historical task execution data in order to describe node performance. These features are mainly derived from two aspects: the value of task number per node and statistics of normalized task duration. They reflect the contention level and the relative processing speed of a node respectively. In addition, those values are collected in a time-incremental manner to capture the cumulative effects. For example, if training data are collected over a month time (30 days), a tuple consisting of 90 attributes will be generated to model the node performance as following:

$$\begin{aligned} &< avg\{\widetilde{D^i_{j\,day1}}\}, \sigma\{\widetilde{D^i_{j\,day1}}\}, norm\{N\_task_{day1}\}, \\ &avg\{\widetilde{D^i_{j\,day2}}\}, \sigma\{\widetilde{D^i_{j\,day2}}\}, norm\{N\_task_{day2}\}, \cdots, \\ &avg\{\widetilde{D^i_{j\,day30}}\}, \sigma\{\widetilde{D^i_{j\,day30}}\}, norm\{N\_task_{day30}\} > \end{aligned}$$

where $\widetilde{D^i_j}$ represents the normalized task duration using z-score normalization [49] to represent the relative speed of a certain task T^i_j within job J_j. The average and the standard deviation $avg\{\widetilde{D^i_{j\,day}}\}$ and $\sigma\{\widetilde{D^i_{j\,day}}\}$ are collected daily in a cumulative manner. In other words, for the 30^{th} day, the results are derived from the whole month's data rather than only from a single day.

Clustering and Labeling. In order to find out the affinities, it is necessary to cluster nodes with similar performance into a group by utilizing the above features and existing clustering techniques such as *k-means*. However, to enable the information usability for the scheduler when deciding which nodes are best fit for launching tasks, there is a urgent need of a labeling method to quantitatively determine and represent the pertaining performance level. The early experience of the automatic labeling procedure is conducted [50] and we can sort out roughly 10% weak nodes from the total training data.

Classification and Prediction. Based on the output of supervised model training and learning, we further propose the classification algorithms to timely determine the performance level for a given machine. In particular, models such as SVM, Boosting, Decision Tree, Random Forest, and Naive Bayes, etc. emphasize specific attributes from the training data to get the optimal performance. For example, the Bayesian classifier requires all attributes to be

TABLE 2. ALGORITHM COMPARISONS

	Precision	Recall	Accuracy
Random Forest	89.47%	58.62%	92.86%
SVM	100%	6.9%	86.22%
Ada Boosting	78.95%	51.72%	90.82%
Decision Tree	62.96%	58.62%	88.78%
Naive Bayes	16.67%	27.59%	68.88%
XGBoost*	82.61%	65.52%	92.86%

independent of each other, which is not suitable in our case. The detailed experimental evaluation to predict node performance category is demonstrated in Table 2 in terms of the precision, recall and accuracy. It is apparently observable that the prediction accuracy of node performance category can be maximized to 92.86% if suitable classification techniques are carefully selected.

6.2. Straggler-aware Scheduling with Blacklisting

The effectiveness of speculative execution can be undermined if improper nodes are selected – the opportunities of task replications that overtake the stragglers will be dramatically diminished when such duplicated tasks are assigned to weakly-performed nodes. Failed speculations will also inevitably lead to the wasted resources and the long tail issues will be exacerbated under systems with high utilization, resulting in deteriorated straggler occurrence and extended job execution. By contrast, if the scheduler is aware of the predicable node performance through the performance forecasting and inferences, the speculations tends to be launched to the most suitable nodes with adequate available resources, minimized interference possibility. With the insightful information from learning, the straggler-aware scheduling can be achieved through performance rating based node ranking, node filtering, and dynamic blacklisting of weak nodes. In this manner, the speculative tasks can be executed in an optimal or sub-optimal place.

In fact, such methodology of profiling, modeling and scheduling can be fully exploited and extended [12] [51] [52] [17]. Similar steps can be integrated into problems where performance-centric tasks are involved such as recommendation of location preferences, priority dynamic evaluation for pending application, etc. For instance, when opportunistically over-subscribing the idle resources, learning-based methods can facilitate the optimal placement based on historical task location and the execution effect.

6.3. Discussion

Overhead Evaluation. It is true that the ML can increase the accuracy during the decision making, we still have to quantitatively assess the incurred system overhead and performance degradation. For example, the offline profiler module can train the workload and work out a classifier or prediction model, but it inevitably brings in additional time and resource consumption. Furthermore, we have to know exactly the positive and negative gaining introduced by the ML module. In this context, the false positive and false negative error rates refer to concepts analogous to type I and type II errors in statistical hypothesis testing. For example, in the procedure of well-performed node prediction and selection, although the false-positive will regard the good nodes as faulty poorly-performed nodes, it will not degrade the straggler mitigation effectiveness. On the contrary, the false-negative case will result in the task executions on badly-behaviored nodes, even become catastrophic to the holistic system.

Online Learning. In another cases where data is continually generated, the model needs to be frequently updated in order to cater the accuracy demands. Especially in the changing scenarios, it is very likely for the prediction or regression models to become invalid or deviate from the precision. Additionally, the model quality is heavily dependent on the parameter numbers and their ranges. Therefore, the profiling procedure is both time- and resource- consuming. To mitigate the overheads, online learning can be further exploited to autonomously enable the model well-suited for environments that change dynamically and even unexpectedly. Such learning techniques such as Reinforcement Learning(RL) [53] [54] aim to predict and evaluate the effectiveness of a series of steps which need long-term decision support. RL can tolerate the temporal (e.g., one-step) reward or penalty as long as the long-term reward can be targeted. We are planning to integrate them into the current schedulers by designing the long-term gains to address a variety of planning and control issues.

7. Conclusion

Resource management plays a fundamentally important role in assigning different kinds of resources within cluster systems to co-located workloads. Compared with ad-hoc heuristics, machine learning approaches can benefit schedulers with intelligent resource allocation, action selection based on contextual states and environmental factors. This paper presents a generalized ML-based solution through comprehensive problem formalizing, data driven profiling, supervised learning modeling, and gives an overall architecture. Furthermore, we can draw the following conclusions:

- Performance-centric and QoS-aware learning can significantly steer the effectiveness and efficiency in consolidated cluster environments, and thereby greatly improve the placement quality of tasks, VMs, and etc.
- It is considerably important to span ML and AI algorithms and systems into both the Cloud and Edge devices. In Fog environments[55], more heterogeneity of resources and appliances will become the norm, and there are larger discrepancies between the Cloud nodes and the Edge devices. Developing intelligent scheduling mechanisms becomes subsequently much more intricate and challenging.
- It is feasible to apply the state-of-the-art Deep Learning or Deep Reinforcement Learning techniques into large-scale systems. For long-term scheduling and optimization goals, such approaches can deal with a large number of planning and controls with fast convergence, and can strengthen the system's adaptability.

Acknowledgments

This work is supported by the National Key Research and Development Program (2016YFB1000103, 2016YFB1000101) and the National Natural Science Foundation of China (61421003). This work is also supported by Beijing Big Data and Brain Computing (BDBC) innovation center.

References

[1] C. Reiss, A. Tumanov, G. R. Ganger *et al.*, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *ACM SoCC*, 2012.

[2] I. S. Moreno, P. Garraghan, P. Townend, and J. Xu, "An approach for characterizing workloads in google cloud to derive realistic resource utilization models," in *IEEE SOSE*, 2013.

[3] V. K. Vavilapalli, A. C. Murthy, C. Douglas *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *ACM SoCC*, 2013.

[4] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center." in *USENIX NSDI*, vol. 11, no. 2011, 2011, pp. 22–22.

[5] Z. Zhang, C. Li, Y. Tao, R. Yang, H. Tang, and J. Xu, "Fuxi: a fault-tolerant resource management and job scheduling system at internet scale," *VLDB Endowment*, vol. 7, no. 13, pp. 1393–1404, 2014.

[6] E. Boutin, J. Ekanayake, W. Lin *et al.*, "Apollo: scalable and coordinated scheduling for cloud-scale computing," in *USENIX OSDI*, 2014.

[7] K. Karanasos, S. Rao *et al.*, "Mercury: hybrid centralized and distributed scheduling in large shared clusters," in *USENIX ATC*, 2015.

[8] M. Abadi, P. Barham, J. Chen, Z. Chen *et al.*, "Tensorflow: A system for large-scale machine learning." in *USENIX OSDI*, 2016.

[9] Y. Jia, E. Shelhamer, J. Donahue *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *ACM Multimedia*, 2014.

[10] T. Chen, M. Li, Y. Li *et al.*, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv preprint arXiv:1512.01274*, 2015.

[11] R. Evans and J. Gao, "Deepmind ai reduces google data centre cooling bill by 40%," *DeepMind blog*, vol. 20, 2016.

[12] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," in *ACM SIGPLAN Notices*, 2013.

[13] O. Yildiz, S. Ibrahim, T. A. Phuong, and G. Antoniu, "Chronos: Failure-aware scheduling in shared hadoop clusters," in *IEEE Big Data*, 2015.

[14] R. Yang, Y. Zhang, P. Garraghan, Y. Feng, J. Ouyang, J. Xu, Z. Zhang, and C. Li, "Reliable computing service in massive-scale systems through rapid low-cost failover," *IEEE Trans. on Services Computing*, 2017.

[15] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J.-M. Pierson, and A. V. Vasilakos, "Cloud computing: Survey on energy efficiency," *ACM Computing Surveys (csur)*, 2015.

[16] R. Yang, I. S. Moreno, J. Xu, and T. Wo, "An analysis of performance interference effects on energy-efficiency of virtualized cloud environments," in *IEEE CloudCom*, 2013.

[17] I. S. Moreno, R. Yang, J. Xu, and T. Wo, "Improved energy-efficiency in cloud datacenters with interference-aware virtual machine placement," in *IEEE ISADS*, 2013.

[18] S. Rosario, A. Benveniste, S. Haar, and C. Jard, "Probabilistic qos and soft contracts for transaction-based web services orchestrations," *IEEE Trans. on Services Computing*, 2008.

[19] M. Alrifai and T. Risse, "Combining global optimization with local selection for efficient qos-aware service composition," in *ACM WWW*, 2009.

[20] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *ACM EuroSys*, 2015, p. 18.

[21] A. Ghodsi, M. Zaharia, B. Hindman *et al.*, "Dominant resource fairness: Fair allocation of multiple resource types." in *USENIX NSDI*, 2011.

[22] Capacity scheduler. [Online]. Available: http://hadoop.apache.org/docs/ current/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html

[23] Fair scheduler. [Online]. Available: http://hadoop.apache.org/docs/ current/hadoop-yarn/hadoop-yarn-site/FairScheduler.html

[24] G. Lodi, F. Panzieri, D. Rossi, and E. Turrini, "Sla-driven clustering of qos-aware application servers," *IEEE Transactions on Software Engineering*, 2007.

[25] Apache hive. [Online]. Available: http://hive.apache.org

[26] Apache pig. [Online]. Available: http://pig.apache.org/

[27] B. Urgaonkar, P. Shenoy, and T. Roscoe, "Resource overbooking and application profiling in shared hosting platforms," *ACM OSDI*, 2002.

[28] L. Tomás, E. B. Lakew, and E. Elmroth, "Service level and performance aware dynamic resource allocation in overbooked data centers," in *ACM/IEEE CCGrid*, 2016.

[29] X. Sun, C. Hu, R. Yang, P. Garraghan, and C. Li, "Rose: Cluster scheduling through efficient resource overselling," in *ACM SOSP poster*, 2017.

[30] Y. Wang, R. Yang, T. Wo, W. Jiang, and C. Hu, "Improving utilization through dynamic vm resource allocation in hybrid cloud environment," in *IEEE ICPADS*, 2014.

[31] Y. Xu, Z. Musgrave, B. Noble, and M. Bailey, "Bobtail: Avoiding long tails in the cloud." in *NSDI*, 2013.

[32] X. Ouyang, P. Garraghan, R. Yang, P. Townend, and J. Xu, "Reducing late-timing failure at scale: straggler root-cause analysis in cloud datacenters," in *IEEE DSN*, 2016.

[33] P. Garraghan, X. Ouyang, R. Yang, D. McKee, and J. Xu, "Straggler root-cause and impact analysis for massive-scale virtualized cloud datacenters," *IEEE Trans. on Services Computing*, 2016.

[34] X. Ouyang, P. Garraghan, B. Primas, D. McKee, P. Townend, and J. Xu, "Adaptive speculation for efficient internetware application execution in clouds," *ACM Trans. on Internet Technology*, 2017.

[35] I. S. Moreno, P. Garraghan *et al.*, "Analysis, modeling and simulation of workload patterns in a large-scale utility cloud," *IEEE Transactions on Cloud Computing*, 2014.

[36] J. R. Quinlan, "Induction on decision tree," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[37] J. A. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural processing letters*, 1999.

[38] S. Haykin, "Neural networks: A comprehensive foundation," *Neural Networks A Comprehensive Foundation*, pp. 71–80, 2008.

[39] R. Yang and J. Xu, "Computing at massive scale: Scalability and dependability challenges," in *IEEE SOSE*, 2016.

[40] J. Taheri, A. Y. Zomaya, and A. Kassler, "vmbbprofiler: A black-box profiling approach to quantify sensitivity of virtual machines to shared cloud resources," *Computing*, 2017.

[41] Y. Chen, E. K. Garcia, M. R. Gupta, A. Rahimi, and L. Cazzanti, "Similarity-based classification: Concepts and algorithms," *Journal of Machine Learning Research*, 2009.

[42] J. H. Friedman, "Greedy function approximation: A gradient boosting machine." *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[43] B. Leo, F. J. H, O. R. A, and S. C. J, *Classification And Regression Trees. Chapman et*. Wadsworth International Group,, 1984.

[44] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *ACM KDD*, 2016.

[45] B. C. Csáji, "Approximation with artificial neural networks," *Faculty of Sciences, Etvs Lornd University, Hungary*, vol. 24, p. 48, 2001.

[46] K. Cho, B. V. Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *Computer Science*, 2014.

[47] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: continual prediction with lstm," *Neural Computation*, 2000.

[48] open cloud cluster at Carnegie Mellon University. (2016). [Online]. Available: http://ftp.pdl.cmu.edu/pub/datasets/hla/dataset.html

[49] L. Al Shalabi and Z. Shaaban, "Normalization as a preprocessing engine for data mining and the approach of preference matrix," in *Dependability of Computer Systems, 2006. DepCos-RELCOMEX'06. International Conference on*. IEEE, 2006, pp. 207–214.

[50] X. Ouyang, C. Wang, R. Yang, G. Yang, P. Townend, and J. Xu, "Mlna: A machine learning based node performance analyzer utilizing straggler statistics," in *IEEE ICPADS*, 2017.

[51] H. Yang, A. Breslow, J. Mars, and L. Tang, "Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers," in *ACM ISCA*, 2013.

[52] C. Delimitrou and C. Kozyrakis, "Quasar: resource-efficient and qos-aware cluster management," in *ACM ASPLOS*, 2014.

[53] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[54] V. Mnih, K. Kavukcuoglu, D. Silver *et al.*, "Human-level control through deep reinforcement learning," *Nature*, 2015.

[55] Z. Wen, R. Yang, P. Garraghan, T. Lin, J. Xu, and M. Rovatsos, "Fog orchestration for internet of things services," *IEEE Internet Computing*, 2017.