Workload Management for Cloud Databases via Machine Learning

Ryan Marcus
Brandeis University
Email: ryan@cs.brandeis.edu

Olga Papaemmanouil
Brandeis University
Email: olga@cs.brandeis.edu

***Abstract*—As elastic IaaS clouds continue to become more cost efficient than on-site datacenters, a wide range of data management applications are migrating to pay-as-you-go cloud computing resources. These diverse applications come with an equally diverse set of performance goals, resource demands, and budget constraints. While existing research has tackled individual tasks such as query placement, scheduling, and resource provisioning to meet these goals and constraints, these techniques fail to provide end-to-end customizable workload management solutions, leading application developers to hand-craft custom heuristics that fit their workload specifications and performance goals. In this vision paper, we argue that workload management challenges can be addressed by leveraging machine learning algorithms. These algorithms can be trained on application-specific properties and performance metrics to *automatically* learn how to provision resources as well as distribute and schedule the execution of incoming query workloads. Towards this goal, we sketch our vision of WiSeDB, a learning-based service that relies on supervised and reinforcement learning to generate workload management strategies for both static and dynamic workloads.**

I. Introduction

Cloud computing has transformed the way data-centric applications are deployed by reducing data processing services to commodities that are paid for on-demand. As the diversity of cloud-based database applications increases, the diversity of the performance goals and workload characteristics needing to be served increases equally. At the same time, all these applications need to address common workload management challenges when deployed on cloud infrastructures. These include handling, in a cost-effective way, tasks such as *resource provisioning* (renting/releasing new VMs), *query placement* (routing the queries to a VM), and *query scheduling* (processing order within a VM) while meeting the application's performance goals and constraints.

Current efforts have tackled these challenges individually and on a per-application basis (e.g. [1]–[4]). However, they all suffer from two main limitations. First, while a broad range of performance criteria are covered by these systems (e.g., response time [1], [2], average workload latency [3]), each offers a solution tuned for *specific performance metrics*. Adapting them to support custom performance goals is not a straightforward task, and using the same query scheduling or VM provisioning strategy for two applications with different performance properties could result in significant monetary losses or degraded system performance. Second, existing systems do not provide an *end-to-end* workload management solution but instead focus on individual aspects of the problem, such as query placement [3], scheduling [1], [2], or resource provisioning [4]. Since these solutions are developed independently of each other, their integration into a unified framework requires substantial effort to "get it right" for each specific case. Hence, application developers often rely on ad-hoc custom heuristics to handle all the above tasks, aiming to satisfy their performance goals and budget constraints.

In this paper, we share our vision for a learning based approach to addressing the above mentioned challenges. We argue that existing machine learning techniques can be leveraged to eliminate such ad-hoc heuristics, and replace them with systems capable of *learning* workload management strategies for a given application and performance goal. Towards this goal we discuss our vision of WiSeDB, a learning-based workload advisor service designed to generate workload management strategies for cloud-based data management applications. We have identified three main design goals for such a service. First, we envision a *metric-independent* service, that allows applications to define custom application-level performance goals, and learns the "best" heuristic for executing a given workload while striving to meet specified performance expectations. Second, given an incoming workload and a performance goal, the service should provide *end-to-end solutions* for processing the workload on a cloud infrastructure, i.e., indicate: (a) the cloud resources to be provisioned (e.g., number/type of VMs), (b) the distribution of resources among the queries (e.g., which VM will execute a given query) and (c) the execution order of these queries within each VM. Finally, our service should be *cost-aware*, i.e., offer solutions optimized for monetary cost.

To satisfy the above requirements, WiSeDB will leverage machine learning algorithms to identify low-cost heuristics for executing incoming workloads. Here, we discuss two different learning approaches we have studied towards realizing our vision of WiSeDB. Specifically, in Section II, we demonstrate how supervised learning (particularly decision tree classifiers) can be used to identify low-cost solutions for workloads with known (or predictable) performance properties. In Section III, we discuss a reinforcement learning approach for actively learning and updating workload management strategies for dynamic, changing workloads. We conclude in Section IV with a discussion of the research problems we plan to pursue further.

II. Offline learning of static strategies

In this section we describe a proof-of-concept supervised learning framework, SLEARN, that we have implemented for WiSeDB. SLEARN utilizes decision tree classifiers to automatically "learn" effective strategies for executing incoming workloads. SLEARN's strategies are expressed as *decision tree*

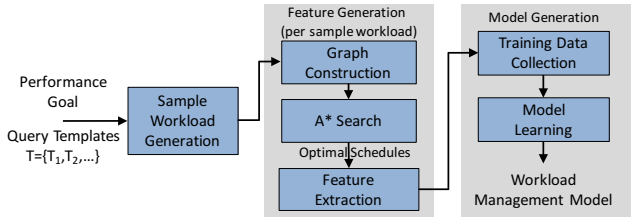

Fig. 1. The SLEARN training process

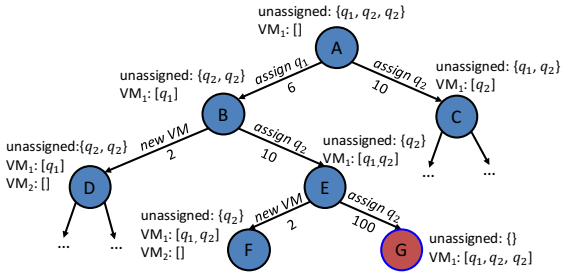

Fig. 2. A workload scheduling graph example

models tuned for application-defined performance goals and workload specifications. Decision models are trained offline, and they are used at runtime to generate low-cost execution strategies for any query set that matches the model's workload specifications. Specifically, for a given incoming query set, WiSeDB parses the model to identify (a) the number/type of VMs to provision, (b) the placement of queries to these VMs and (c) the query execution order within each VM. Each model is cost-aware: it is trained on a set of performance and cost related features that account for provisioning costs *and* any penalties paid due to violations of the performance goals.

Workload & Performance Goal Specification Our learning framework allows users to specify a set of representative *query templates* $T_1, T_2, \ldots$, each corresponding to groups of queries of the same type and hence similar execution times when processed on the same execution environment (e.g., resource configurations, concurrent queries, etc). Incoming workloads can include any combination of queries of these templates. Latency estimates per template must also be provided by either the application itself (e.g., by executing representative queries a-priori on various VM types), or by using existing prediction models [7].

In addition to query templates, applications need to specify a performance goal as a function of query latency. Our current implementation of SLEARN can learn effective strategies for the following performance metrics: (1) *Query Latency*: users specify an upper bound for the latency of each query template. (2) *Max*: the users express an upper bound on the worst query response time in their workload. (3) *Average*: an upper limit on the average query latency of a workload. (4) *Percent*: specifies that at least $x\%$ of the workload's query must be completed within t seconds. These metrics cover a range of performance goals typically used for cloud databases (e.g., [2]–[4]), but, to the best of our knowledge, our framework is the first one to support all of them. Performance goals are expressed as part of a Service-Level-Agreement (SLA) between the IaaS provider and the application. The agreement states the workload specification (i.e., query template), the expected performance goal, and a penalty function that defines the penalty to be paid to the application if that goal is not met.

Supervised Learning Our supervised-learning framework is actualized through a three-step process shown in Figure 1. First, we create a small number of *sample workloads* by drawing queries at random from the specified query template. Second, for each random workload we *identify the optimal workload execution strategy* and *extract performance and cost related features* from these optimal solutions. We then train a decision tree classifier on the training set that consists of all the features and optimal strategies we collected for all the random workloads. The generated *decision model* represents a workload management heuristic tailored to the application's query templates and performance goals.

The above process raises two main challenges: (a) generating the optimal execution strategy for a given random workload and (b) identifying the set of features that can effectively characterize these optimal decisions (e.g., assign a query on an existing VM, provision a new VM, etc). To address these, SLEARN represents the problem of scheduling a given workload as a graph navigation problem. It constructs a directed, weighted graph where vertices represent steps of the workload scheduling process. Specifically, each vertex stores the scheduling decision for already-scheduled queries as well as the still-unassigned queries. The scheduling decision defines a set of VM to be provisioned for the workload and the query queue in each VM.

Figure 2 shows part of an example graph for a query set $\{q_1, q_2, q_2\}$, where q_i represent a query of template T_i. For a given random workload, the first vertex on the graph has all the queries unassigned (i.e., node A). Edges represent a decision to (a) assign a query to one of the provisioned VMs (e.g., edge from A to B) or (b) start a new VM (e.g., from B to D). The weight of each edge is equal to the cost of performing the corresponding action. The graph allows us to generate and cost all possible decisions for a given workload. Finding the optimal strategy (i.e., the least-cost strategy) is reduced to finding the shortest path through the graph. SLEARN searches the graph using the A* algorithm [5], taking advantage of a few search heuristics we have developed to improve its efficiency.

For each decision within the optimal solution (i.e., for each edge in the optimal path), we extract a feature set to characterize the decision. We select features that appear in human-derived heuristics [2], [3], hoping that a machine-learning algorithm will be able to construct a customized heuristic for arbitrary task templates and performance goals from them. The most important features extracted were:

1) `wait-time`: the amount of time that a query would have to wait before execution. This helps our model decide which tasks should be placed based on their deadline.
2) `cost-of-X`: the cost incurred (including any penalty costs) by placing a query of template `X` on the most recently created VM. This allows our model to check the cost of placing a certain query and decide whether to assign it or to create a new VM.
3) `have-X`: if a query of template `X` is still unassigned. This helps our model understand how the template of unassigned queries affects decisions on the optimal path.

We concatenate all the collected features into a set and train a decision tree model on it. The model takes the above features as input and predicts an edge (i.e., a decision to provision a new VM or assign a query to a VM). This model can then be used to schedule new workloads of the given query template.

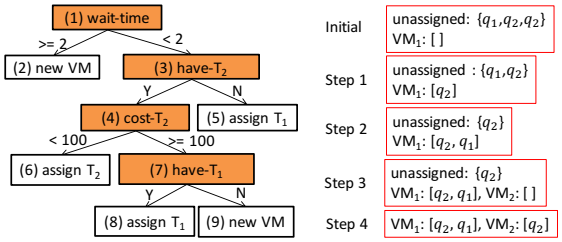

Fig. 3. An example decision tree generated by SLEARN

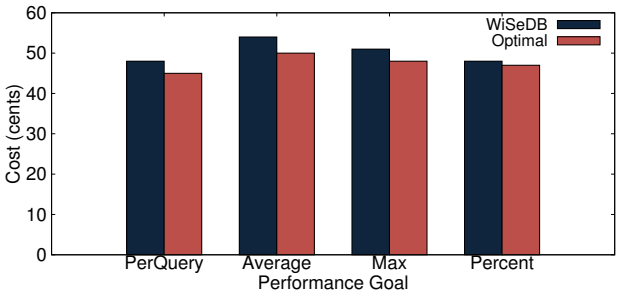

Fig. 4. Performance of SLEARN models compared to optimal schedules

Figure 3 shows an example decision tree. Each orange node represents a binary split on a feature. The decision nodes (white) represent suggested actions. The right side of Figure 3 shows an incoming workload $\{q_1, q_2, q_2\}$. To schedule this workload, we parse the decision tree as follows. In the first node (1), we check the wait time, which is zero since all queries are unassigned, so we proceed to (3). The workload has unassigned queries of template T_2, so we proceed to (4). Here, we calculate the cost of placing an instance of T_2 on a VM (we assume for simplicity a single VM type). Let us assume the cost is less than 100, which leads to (6), which assigns an instance of T_2 to the initial VM (Step 1). Since we have more queries in the workload, we re-parse the tree. In (1) let us assume the wait time on the most recent VM is 1 minute (this is the runtime of queries in T_2) so we move to (3). Since we have one more query of T_2 unassigned, we move to (4). Let us assume the cost of assigning a query of T_2 is more than 100 (because it would need to wait for q_2 to complete). We move to (7) and we check if there are unassigned instances of T_1. Since there is, we assign q_1 to the last provisioned VM (Step 2). We re-parse the tree in the same way and by following nodes (1)→(2), then again as (1)→(3)→(4)→(7)→(9), so we provision a new VM (Step 3) and assign q_2 onto it (Step 4).

In general, the decision model will place an instance of T_2, then an instance of T_1, then create a new VM. This repeats until tasks of T_1 or T_2 are depleted. Then, single instances of the remaining templates are placed on new VMs until none remain. For the case of these two query templates, the heuristic learned by SLEARN is equivalent to sorting queries in increasing order of latency and placing each query on the first VM where the query "fits" (i.e., incurs no penalty).

Results Figure 4 shows the cost of scheduling workloads of 30 queries uniformly distributed across 10 query templates for four performance metrics. `Max` requires that the maximum query latency be less than 10 minutes, 2.5 times the latency of the longest query in our workload. `PerQuery` requires that the final latency of each query is not more that 2.5 times their individual latencies. `Average` requires that the average query latency is less than 4 minutes, and `Percent` requires that 90% of the queries finish with 10 minutes. The experiment was conducted on the Amazon EC2 instances using the `TPC-H` benchmark, where query templates 1-10 were used as query templates. Here, we compare schedules generated by WiSeDB with the optimal, brute-forced schedules for this specific setting. WiSeDB's schedules are within 8% of the minimum cost schedule for all the performance metrics. These results were consistent independently of the size of the runtime workloads and the strictness of the performance metrics. SLEARN also learns models very quickly, requiring at most one minute to train for workloads of 15 query templates.

III. Adaptive Learning of Dynamic Strategies

Our supervised learning technique relied on two assumptions. First, incoming workloads will consist of already known query templates, and second, that the execution times of these query templates can be predicted. However, a plethora of data centric applications need to support dynamic workloads involving previously unseen queries in which users explore data, search for irregularities, etc., and for which performance prediction is a major challenge. In a dynamic setting, the performance of a fixed strategy generated by SLEARN will decay, possibly very quickly. To address this limitation, we sketch the foundation for *RLEARN*, a reinforcement learning approach that can be used to adaptively learn workload management strategies for dynamically changing workloads.

Reinforcement Learning Reinforcement learning differs from standard supervised learning in that optimal/correct decisions are not required a priori. The system decides on an action in order to maximize a cumulative reward. It learns from its past (possibly sub-optimal) decisions, and its decisions improve over time. Here, there is a focus on online learning, which involves finding a balance between exploration (reward maximization based on the knowledge already acquired) and exploitation (attempting new actions to further increase knowledge). In machine learning, this trade-off has been studied in the contextual multi-armed bandit (CMAB) problem [6].

The CMAB is a problem where a gambler plays on a row of slot machines (one-armed bandits) and needs to decide which machines to play (i.e., which arms to pull) in order to maximize the sum of rewards earned. In each round, the player decides which machine to play (action a) and receives a reward r. The decision is made by observing a feature vector X (aka *context*) which summarizes information on the state of the machine at this iteration. The gambler then improves their strategy through the new observation $\{a, X, r\}$. The gambler's goal is to achieve the highest cumulative reward. Hence, the gambler aims to collect information about how the feature vectors and rewards relate to each other so that they can predict the next best machine to play by looking at the feature vector.

Our workload management problem can be modeled as a *tiered network of CMABs*, illustrated in Figure 5. Each VM corresponds to a slot machine in one of several tiers, where each tier represents a different type of VM. Tiers can be ordered based on price or performance/resource criteria. Each machine has three arms/actions: `Accept`, `Pass`, and `Down`. When a query enters the system, it is first given to the root machine (top left). The algorithm makes a decision based on a set of features that capture information about the new query

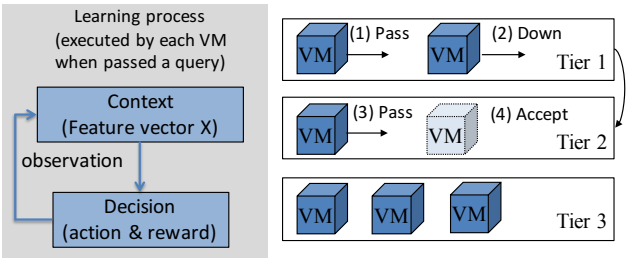

Fig. 5. The RLEARN learning process and an example decision process

and the current state of the underlying VM. If the `Accept` action is selected, the query is added to the VM's execution queue. If the `Pass` action is selected, the query is passed to another VM in the same tier. If there is no other VM, a new VM of that tier is provisioned. If the `Down` action is selected, the query is passed downwards to the first VM in the next tier. The last tier contains no `Down` arms. The network contains no cycles, so a query will eventually be accepted by some VM.

The rewards for each decision are determined after the query executes. Assuming that a query earned the service provider x dollars and that processing the query required y dollars, the accumulated reward is the profit $x - y$. Since each CMAB strives to maximize its own reward, it is not necessary to "divide up" the profit between the CMABs – each CMAB can assume it independently realized the entire profit. This technique can be thought of as backpropagating the rewards from the accepting VM to the CMABs involved in passing the query to that VM. Each CMAB adds a new entry to their reward observations, and as more queries move through the system, each CMAB's set of observations grows larger, and the system as a whole will learn a workload management strategy. Finally, the contextual information, X, would vary greatly between applications. For a database application, the feature vector could contain performance indicators (e.g., number of logical I/Os required by the query [7]), the queries in the queue of the VM, the tables accessed by them, etc.

To illustrate this process, let us assume a CMAB network is paid \$4 to process a query. Since the network has no prior experiences, it erroneously provisions ten new VMs (each new CMAB selects the `Pass` action) and assigns the query to the last VM (the 10th CMAB selects the `Accept` arm). Assuming that a VM costs \$1 to provision and that processing the query costs \$1, deciding to `Pass` the query when the machine's context indicates that it is empty will be associated a reward of $\$4 - \$11 = -\$7$. Now, each CMAB in the network will be biased against pulling the right arm when they have not accepted any queries yet. When a new query arrives, it is likely that the first CMAB will accept it (earning it a profit of \$3), or pass it downwards. In the first case, the CMAB will associate a positive reward with selecting the `Accept` action when empty. As more queries flow through the network, each CMAB will gain more experience and thus make more informed decisions.

One algorithm for effectively solving the CMAB problem is Thompson sampling [6], which chooses the action that maximizes the expected reward with respect to a randomly drawn belief. The sampling can be implemented by Bayesian approach as follows. The set of past observations $\mathcal{D}$ is made of triplets (a_i, X_i, r_i) and are modeled using a parametric likelihood function $P(r|a, x, \theta)$ depending on some parameters θ. For a generic application, θ could be a set of coefficients for a linear model of each arm relating each feature in X_i to a reward. Given some prior distribution $P(\theta)$ on these parameters, the posterior distribution of these parameters is given by the Bayes rule, $P(\theta|\mathcal{D}) = P(\mathcal{D}|\theta)P(\theta)$. In practice, the approach is implemented by sampling, in each round, a parameter θ from the posterior $P(\theta|\mathcal{D})$, and choosing the action a that maximizes the expected reward given the parameter θ, the action, and the current context. Conceptually, this means that the player instantiates their beliefs randomly in each round, and then acts optimally according to them.

IV. Open Challenges and Ongoing Work

Our two learning-based approaches indicate that there is room for advancement in the space of applying machine learning algorithms to the workload management problem for cloud databases. Overall, we believe that WiSeDB opens up exciting research paths, some of which we highlight next.

Adaptive Modeling It is often desirable to allow applications to explore potential performance and cost trade-offs for their workloads. This implies generating different models for the same workload with stricter/looser performance constraints and thus higher/lower costs. However, SLEARN tunes its decision model for a specific performance goal and changes in this goal will trigger WiSeDB to re-train the decision model. We are working on a technique that adapts an existing model trained for a given workload and performance goal to generate with minimal training a new model for the same workload but stricter goals. The main idea is that if two decision models are trained for the same query templates and similar performance goals, their training sets will share significant information. Our technique aims to identify and reuse these commonalities.

Online Scheduling The training overhead for RLEARN is paid every time a task enters the system. Furthermore, with RLEARN, scheduling within a VM is handled by using a basic FIFO queue in our example. In order to allow RLEARN to learn more advanced query scheduling strategies, the single choice of `Accept` could be replaced with a continuum-armed bandit [8], which selects a number between zero and one as a priority instead of simply pulling an arm (boolean).

Acknowledgment

This research was funded by NSF IIS 1253196.

References

[1] Chi et al. SLA-tree: a framework for efficiently supporting SLA-based decisions in cloud computing. *EDBT '11*.

[2] Chi et al. iCBS: incremental cost-based scheduling under piecewise linear SLAs. *VLDB '11*.

[3] Liu et al. PMAX: tenant placement in multitenant databases for profit maximization. *EDBT '13*.

[4] Jalaparti et al. Bridging the Tenant-Provider Gap in Cloud Services. *SOCC '12*.

[5] Hart et al. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE SSC '68*.

[6] Chapelle et al. An empr. evaluation of Thompson sampling. *NIPS '11*.

[7] Duggan et al. Performance Prediction for Concurrent Database Workloads. *SIGMOD 2011*.

[8] Auer et al. Improved rates for the stochastic continuum-armed bandit problem. *Learning Theory '07*.