GridFormation: Towards Self-Driven Online Data Partitioning using Reinforcement Learning

Gabriel Campero Durand
University of Magdeburg
campero@ovgu.de

Marcus Pinnecke
University of Magdeburg
pinnecke@ovgu.de

Rufat Piriyev
University of Magdeburg
piriyev@ovgu.de

Mahmoud Mohsen
University of Magdeburg
mohsen@ovgu.de

David Broneske
University of Magdeburg
broneske@ovgu.de

Gunter Saake
University of Magdeburg
saake@ovgu.de

Maya S. Sekeran
University of Magdeburg
santhira@ovgu.de

Fabián Rodriguez
University of Magdeburg
fabian.rodriguez@st.ovgu.de

Laxmi Balami
University of Magdeburg
balami@ovgu.de

ABSTRACT

In this paper we define a research agenda to develop a general framework supporting online autonomous tuning of data partitioning and layouts with a reinforcement learning formulation. We establish the core elements of our approach: agent, environment, action space and supporting components. Externally predicted workloads and the current physical design serve as input to our agent. The environment guides the search process by generating immediate rewards based on fresh cost estimates, for either the entirety or a sample of queries from the workload, and by deciding the possible actions given a state. This set of actions is configurable, enabling the representation of different partitioning problems. For use in an online setting the agent learns a fixed-length sequence of n actions that maximize the temporal reward for the predicted workload. Through an initial implementation we assert the feasibility of our approach. To conclude, we list open challenges for this work.

CCS CONCEPTS

• **Information systems** → **Physical data models**; *Autonomous database administration*;

KEYWORDS

Physical design, Adaptive layouts, Deep Q-Learning

ACM Reference Format:
Gabriel Campero Durand, Marcus Pinnecke, Rufat Piriyev, Mahmoud Mohsen, David Broneske, Gunter Saake, Maya S. Sekeran, Fabián Rodriguez, and Laxmi Balami. 2018. GridFormation: Towards Self-Driven Online Data Partitioning using Reinforcement Learning. In *aiDM'18: First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management , June 10, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3211954.3211956

aiDM'18, June 10, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-5851-4/18/06…$15.00
https://doi.org/10.1145/3211954.3211956

1 INTRODUCTION

Data partitioning and layout selection are cornerstone physical design functions for storage engines. Through data partitioning tables are divided into subsets of tuples and attributes, such that they can be managed at a finer granularity. In turn, in layout selection a specific format from the given DBMS (e.g., NSM, DSM, PAX) is assigned to each of the resulting subsets. By using partitions and data layouts with high affinity to a workload, I/O requirements and cache misses during query processing can be reduced, resulting in overall runtime improvements [8, 10, 25]. Further gains from partitioning are manageability [1] and improved recovery handling.

Much research has been devoted to these topics in the last decades, with efficient methods developed for finding optimal partitions in offline [7, 10, 12, 17, 19–21, 23] and online settings [15], complemented with techniques for reaching approximately optimal solutions [27]. The automation of layout selection has also been investigated for diverse use cases [25], often supported by code-generation for layout-tuned operators [4], or comprehensive DBMS designs extending to the query engine [3, 9].

We observe that this abundance of approaches can be specially challenging for DBMS development, and that solutions that enable to bring these alternatives within a single framework could be beneficial. We consider that with the continued development of novel layout choices and use cases, the goal of automated online partitioning runs the risk of being approached through specialized implementations targeting focused challenges (e.g. index alignment, co-processor utilization, raw data or skew in OLTP partitioning), missing opportunities from a more general approach.

Though offline partitioning methods are effectively optimization problems, online methods might not be able to explore the entire search space within limited time horizons and might be better addressed under a machine learning formulation. Thus, we propose adopting one of such formulations. We also observe that machine learning methods might be useful as a principled way of structuring the exploration process, inferring the value of unvisited partitions states and in keeping fixed memory budgets. Furthermore, in adopting deep learning, massively parallel co-processors, such as GPUs, could be leveraged.

In this paper we propose an early concept for framing the online self-managing partitioning and layout selection tasks as a reinforcement learning problem, with the goal of building a general solution capable of leveraging experience and mimicking specialized methods. For this we define precisely agents, environments, actions and supporting components. Since our work is in an early stage we do not offer comprehensive evaluations yet.

We structure our presentation as follows:

- In Sec. 2 we define the basics of reinforcement learning and the components of our solution.
- In Sec. 3 we present basic aspects from our preliminary implementation.
- In Sec. 4 we summarize related work and in Sec. 5 we list open challenges that we seek to address in the next steps of our research.

2 A BLUEPRINT FOR DATA PARTITIONING WITH REINFORCEMENT LEARNING

In this section we start by presenting elementary background on reinforcement learning (Sec. 2.1), then we present the core components of our proposal (Sec. 2.2).

2.1 Background

Reinforcement Learning (RL) is one of several approaches in Machine Learning where an intelligent agent learns an optimal control policy through interactions with its environment and observing the results of these interactions. After each iteration, the intelligent agent receives a scalar reward and tries to improve the learned policy to maximize the cumulative reward. RL aims to estimate the optimal policy without knowing the exact model of actions that should be followed, by only relying on the interactions with the system environment.

The system implementing RL, covering the agent/environment interaction settings, needs to be formally modeled as a Markovian Decision Process, consisting of a tuple of 5 elements, as described in RL textbooks [28]:

- S : Set of states s
- A : Set of actions a
- $p(s_{t+1}|s_t, a_t)$: Set of Markovian transition probabilities
- $p(r_{t+1}|s_t, a_t)$: Reward function
- γ : Discount factor which is set between 0 and 1. The more into the future the reward is, the less we take it into consideration.

Here, the interaction between the agent and the system is modeled by a policy which associates to each state s, a probability distribution over actions a. The quality of such a policy is quantified by a value function which associates to each state the expected cumulative discounted reward. An optimal policy is one that maximizes the value function. Value functions are state-action pair functions that estimate how good a particular action will be in a given state, or what the return for that action is expected to be. The value function of a given policy satisfies the (linear) Bellman evaluation equation and the optimal value function satisfies the (non-linear) Bellman optimality equation.

2.2 The Grid Formation Problem

Given these definitions, the formulation of the generic partitioning problem (i.e., the grid formation problem) as a RL instance represented by a MDP becomes apparent. In the following we describe the core components:

2.2.1 Environment. The environment is represented by the current state S of the table. We considered that two representations are possible for the state. In one a matrix can be used to represent the table, with entries in each cell encoding the ids of the partitions which include that cell. For this representation a careful enumeration of partitions is needed. Another, arguably simpler, representation is to store each partition as a set of the item ids (i.e., the fixed positions of the item in the table) with the additional information of the chosen layout. Hence, a partitioning can be represented as a set of sets.

The environment is also initialized with the predicted workload for the current training process, this is formed by a group of queries, and the frequency of each query. The queries themselves can be described (as partitions) by the items that they access. This formulation of the workload can be extended to include sequences of queries or selectivity information.

The reward, $p(r_{t+1}|s_t, a_t)$, given to an action by the environment does not depend on the distance to a fixed target. It is a function of the fitness of a partitioning to support all queries from the workload. For large workloads, additional skew and noise parameters can be used to sample queries in each step. Hence, given a state (incl. current partitioning and workload), action rewards are assigned based cost estimations (i.e., immediate rewards) for either the complete workload or a sample.

Deterministic rewards that consider a complete workload are common in vertical partitioning algorithms. For example, open-source implementations are available for different algorithms using costs adapted to HDD (with seek and scan components) and main memory assumptions (based on Hyrise [10]) over all queries from the TPC-H workload that access a selected table [16].

Generic stochastic rewards can be calculated by determining the most similar partition (MSP), from the partitioning which goodness we are measuring. The MSP is the one with the most common items to the current query (CQ). An additional aspect that can be considered are the untouched partitions by in the selected query set (UPQ). With these components a reward can then be calculated as follows:

$$\sum_i (\left|MSP_i \bigcap CQ_j\right| - \left|MSP_i - CQ_j\right|) - UPQ \tag{1}$$

Penalizing unused partitions models in-memory DBMS, with unnecessary partitions having the side effect of filling the memory with irrelevant data. Through extended testing we have seen that such a stochastic rewarding strategy can indeed guide agents to a solution. More evaluations are in the scope of our future work.

2.2.2 Actions. For the set of possible *actions* A, given the current state, the following can be considered:

- add a row, a column or an item, this can only be done if these do not already exist as a partition
- merge two partitions, here since each partition is a set, common items are de-duplicated
- remove a partition, an action that is only possible if the complete set of partitions covers the entirety of the table

- divide a partition into two
- change a layout within a partition

Each action, in turn, can be represented as the resulting partitioning after applying the action. Hence, in our formulation, state transitions, $p(s_{t+1}|s_t, a_t)$, are deterministic. Our aim in choosing these actions is to model both horizontal and vertical partitioning with optional replications and layout changes. We build these actions as a generalization of diverse physical design characteristics developed for mixed workloads[24]. Furthermore, we selected these simple actions as they could reasonably be applied incrementally as part of query processing in an online system. Another consideration is that by choosing relatively simple actions, we reduce the action space given a state (for example, all the ways of splitting a partition into 2 results in less actions than all the possible ways of splitting a partition into 3). As a drawback from the design of our action space, complex sequences of actions might be needed for some trivial changes (e.g., to remove a specific item from a partition it might be necessary to first split a partition and then to remove the item).

The environment is responsible for providing the agent with a list of actions to consider, and an expected reward for each action. The selection of actions enables to frame the problem either as vertical/horizontal partitioning only, or specific hybrids. The choice of actions considered could also be adapted to limit the amount of exploration and to prevent cycles. Regarding rewards, they can be discounted with a cost associated to performing each action (e.g. as considered by Basu et al. [5]).

2.2.3 Agents. In our formulation of the problem the agent is in charge of learning the long-term value of a given action for a combination of workload-state pairs. This leads itself to two alternatives for training. In order to produce a general agent, then it should be trained on different workloads. For less general agents, it is possible to train them on a specific workload. In order for the last formulation to be online, a fixed sequence of actions of length n for a given logical table and workload can be learned through a number of training iterations k, given an initial state and a workload passed as a parameter both to the agent and to the environment. Once a good policy is learned, the agent outputs this solution and waits for a next state and workload.

The policy learning process can be achieved in several ways. For our current research and implementation we selected have Q-Learning, to manage simple cases, and Deep Q-Learning choices for more complex ones.

Q-learning is a simple off-policy algorithm for temporal difference learning, corresponding to a model-free Reinforcement Learning method[13] used when the agent initially only knows the set of possible states and actions. Since the state transition and reward models are also not known, the agent improves its behavior through learning from the history of interactions with its environment. Under sufficient training, the Q-learning algorithm is proven to converge to a close approximation of the action-value function for an arbitrary policy. We can express the Q-value of state s and action a in terms of the Q-value of the next state sfi as stated by the Bellman equation:

$$Q(s, a) = r + \gamma max_{a'} Q(s', a') \quad (2)$$

The maximum future reward for this state and action is the immediate reward plus maximum future reward for the next state. The main idea in Q-learning is that we can iteratively approximate the Q-function, the long-term value of an action, by using the Bellman equation. In the simplest case the Q-function is implemented as a table, with states as rows and actions as columns.

$$Q(s, a) = (1 - \alpha)Q(s, a) + \alpha\{r + \gamma max_{a'} Q(s', a') - Q(s, a)\} \quad (3)$$

Here, α is a learning rate that controls how much of the difference between previous Q-value and newly proposed Q-value is taken into account. The $max_{a'}\ Q(s', a')$ used to update $Q(s, a)$ is only an approximation and in early stages of learning it may be completely wrong. However under proper training the approximation improves and if we perform this update enough times, then the Q-function will converge and represent the true Q-value.

We selected Q-Learning for our study since it is a value-based method that does not require a complete exploration of possible policies to converge to a reasonably optimal solution. Policy based methods can be expected to require more training for large action spaces like the one we consider.

Vanilla Q-Learning could be adequate when the search space is limited, as is the case of vertical partitions for few steps. However, when including horizontal partitions, the number of partitions that can be generated given a table state increases at a fast rate, thus maintaining a Q-table could be unfeasible. In fact, the number of possible partitions of a set with N elements into disjoint, non-empty subsets is described by the Bell numbers[11]. This number increases even more when we consider replication, having its upper bound limited by the square of the amount of items that conform the table. Accordingly, the general grid formation problem (considering replications and partitions in both dimensions) might be framed as a *double Bell number* problem. Since the size of the Q-table grows rapidly, a naive Q-Learning approach becomes intractable for real world scenarios due to the large memory requirements. Furthermore, the exhaustive process of traversing the entire space of state-action pairs that guarantee the convergence to the optimal value function becomes computationally expensive for a large table. In addition by requiring exact matches, unvisited actions cannot be estimated from visited ones.

Deep Q-Learning extends Q-Learning for large domains, approximating the Q-Function instead of finding the optimal, meaning that we don't need to store the complete Q-table as this is approximated by the weights within a neural network. On the plus side, this approach enables the system to give approximate values for unvisited states. The main goal of this approach is to minimize the distance between the approximated function and the Q-Function.

$$\widehat{Q_\theta}(s, a|\theta) \sim Q(s, a) \quad (4)$$

By tuning the amount of layers and units within layers, a Deep Neural Network can approximate functions that work with high-dimensional data and increasing complexity. Deep Q-Learning employs a Deep Neural Network called Deep Q-Network (DQN) to represent the Q-Function, that takes as its input a state and provides the estimate of the corresponding Q-value for each of the possible actions the agent can take as its output.

In an alternative approach, the network can be used to learn the Q-value given a state and an action. In consideration of the large number of states in the grid formation problem, in our implementations we've only studied the second approach.

Given that we are approximating the function, we need to update Bellman's equation accordingly.

$$Q(s, a) = r_t + \gamma max_{a'} \widehat{Q}(s', a'|\theta) \quad (5)$$

There is a mismatch, of sorts, between how neural networks need to be trained (since they assume less correlated data) and how reinforcement learning works (with possibly strong data correlations within steps of one training episode). To address this, DQNs are trained through a mechanism called experience replay. In this process, previous experiences are stored in a "replay memory". During training, we randomly sample experiences from this memory and use them, instead of the most recent transition, trying to prevent the network to drive into a local minimum.

The exploration-exploitation dilemma considers the problem of an agent getting stuck in a local minimum because of the greedy nature of its decision making process, which leads it to tend to choose the best available action. One strategy to avoid this situation is called ϵ -greedy approximation, which consists on choosing in every step between the best action and a random one, ϵ being the probability of taking the random action.

A strategy that has shown great results is to start training the model with a high degree of exploration and slowly decreasing the value of ϵ over time as the model gets confident about the policy it learned. These are some of the aspects that we considered when designing and training our model. Some other optimizations introduced in our implementation, to help the training times were to initialize the network weights with carefully small values, and to bias the sample selection of items from the experience replay towards items from successful runs.

Though for our research we have used Q-Learning and Deep Q-Learning, it is possible to adopt other algorithms like Actor-Critic models or PPO. In our work the key distinction in how we select algorithms is between simple and deep models, since both have different characteristics and are suitable for managing cases of different complexity.

2.2.4 Supporting Components. Our proposed solution relies on components other than those specific to reinforcement learning. Namely, a cost estimator and a workload forecaster. These components are separate because their learning targets are markedly different from the ones of our model. Workload forecasting can be improved in ways that might not affect our model, however changes in cost estimation might invalidate and impact previously learned features in the model. In future work we will consider the fragility of the learning process w.r.t the quality of estimations, and alternatives to updating learned features with new estimations.

With this we conclude our presentation on the fundamentals of our proposed approach for framing automated partitioning and layouts as a reinforcement learning problem. In Sec. 3 we discuss our early implementation and illustrate some limited practical observations.

3 PRELIMINARY IMPLEMENTATION

For our implementation we used OpenAI Gym[1] an open source toolkit for reinforcement learning. In OpenAI Gym, the agents experience is divided into a serial number of episodes. In each training episode, the agent chooses its action randomly (i.e., following a parameterized distribution that considers the exploitation vs. exploration trade off, and that is biased towards high Q-values). Each episode continues until an environment reaches a terminal state, which is reaching the maximum number of steps, n, defined for the learning instance. The main target of the agent is thus to achieve maximal rewards in as many episodes as possible. As a result of the training during k episodes, the agent learns a sequence of steps that correspond to small physical design tasks to achieve workload adaptivity. The solution of the process is the sequence of steps with associated maximal rewards.

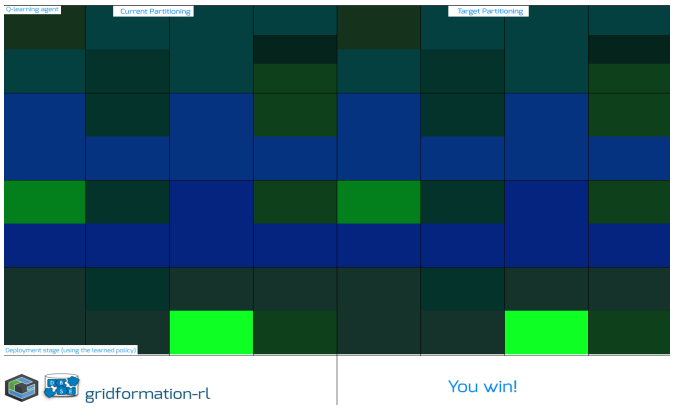

Figure 1: Screenshot of our visualization of the grid formation learning process.

Fig. 1 shows the visualization we developed to illustrate the process. We color-code the membership of items to grids. On the right side we display continuously the target partitioning or grid formation. On the left side we display first the progress of the training, with the agent exploring and exploiting the actions space. After training we display on the left side the sequence of steps of the learned policy.

In terms of the implementation, the DQN agent is not very different from the basic one. It takes as input the encoded state of the table (with one numeric value stating, per item, the partitions to which it belongs), the workload (represented as the set of values from a usage matrix with queries as rows, and each position serving as a flag for the query-item uses), and finally an action (as the encoded state resulting from the selected action). The network predicts the future value of the input features (i.e the action-workload-state), for this, the maximum one-hop predicted reward of the next step is propagated during training. The specifics of network configuration and training optimizations are currently hand-tuned. For the example of the vertical partitioning the lineitem table in TPC-H we implemented as input 17 x 16 nodes corresponding to the queries using columns of the table and the number of columns of the table,

[1] https://github.com/openai/gym

in addition to 16 x 2 input nodes for the state and the action. For this case we selected 3 hidden layers of 4 nodes each. For implementing the network we use Keras, ReLu activation functions and select Adam as an optimizer. Advanced optimizations to speed-up the training process, not possible in the straight-forward approach, such as parallelization of the training, are possible for this model, and we seek to evaluate them in next steps.

Since we do not have an integration with a database, we cannot yet provide practical evaluations using our implementations. However, to illustrate some of the observed performance-impacting aspects of the training process, we give an example related to the number of k iterations needed for convergence in simple transformations requiring few steps (n=2 or 3) using a straightforward Q-Learning implementation. We also select to train on a fixed workload, such that the learner overfits it. Since the described cases are quite basic, optimality is asserted by simple cost calculations.

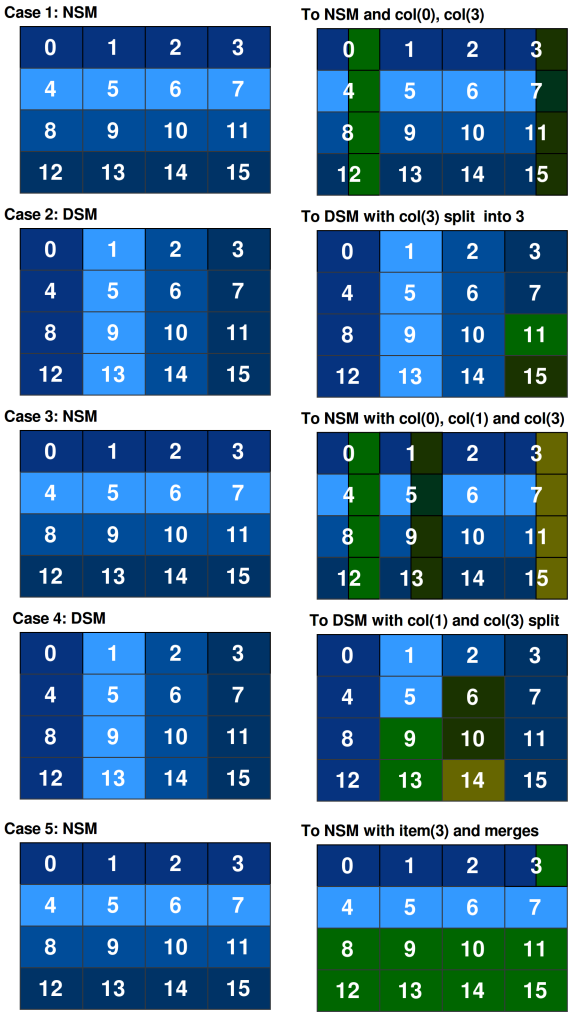

Figure 2: Simple start and target partitions for example cases.

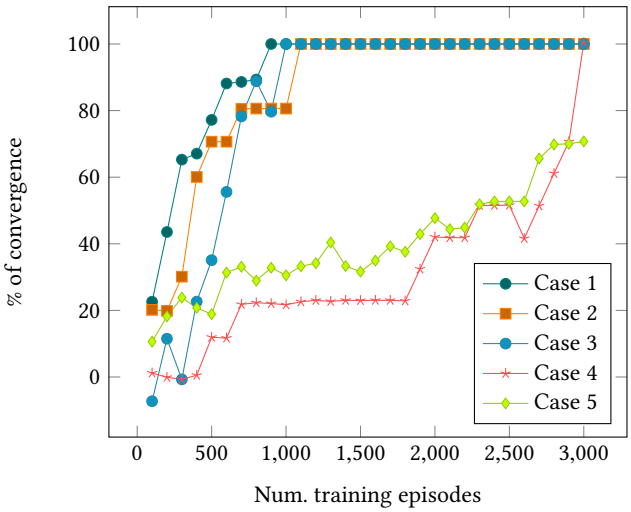

Figure 3: Amount of convergence per number of episodes for simple cases

Case 1: This case represents the simplest one in our illustration test. The small table should start as a row-store and in 2 steps should add two new partitions, namely column 0 and column 3. This learning case requires only two steps, which can also be performed independently (i.e., they are commutative).

Case 2: The table starts from all columns and needs to partition column 3 in a specific way. This requires 2 steps. Some dependencies between steps are introduced (i.e., the partitioning steps need to be done one after another).

Case 3: This case represents an instance similar to case 1, with commutative steps, but with an addition of an extra column (which is also an extra step) to the target partitioning.

Case 4: In this case the model needs to learn 3 steps, starting from all columns it has to divide column 1 and column 2 in specific ways. Just like in case 2, we observe that steps for partitioning within column 2 have certain dependencies between them.

Case 5: We designed this 3 step test such that the table would start from all rows, then add an item (i.e., item 3), subsequently merging this item with rows 2 and 3. This case introduces the dependency of adding the item before merging it to a row.

Fig. 3 displays the results for our evaluation of convergence times. We find that the first 3 cases require a small number of training iterations (1k) to find the optimal policy. On the other hand, test cases 4 and 5 converge after 3 times as many iterations (3k). This evaluation illustrates, specifically for the case of test 3 vs. tests 4 and 5, that convergence times depend more on the complexity (i.e. the dependency between steps) than on the number of steps themselves.

From our ongoing work we observe that dependencies among steps in a learning instance is an important factor that affects training convergence. The more dependencies the slower the convergence, since more exploration is required, first for finding the optimal, second for propagating the future rewards to the learned action-state values.

In fact there can be cases when increasing the number of steps, but keeping the complexity, can have less influence on convergence times. This can be also understood as a case of the model needing to explore more when there is only one way to reach a goal, and less when there are multiple ways.

4 RELATED WORK

Automated Partitioning: There is a long history of automated administration and tuning advisors in DBMSs, mostly championed by the AutoAdmin project from Microsoft. For the specific case of partitioning, Agrawal et. al. present an integrated solution combining index selection with partitioning and materialized views, employing estimations from the query optimizer to recommend partitions [1]. In their approach authors select column-groups based on interestingness measures and adopt greedy algorithms to select configurations under different considerations.

Automated Reorganization for Adaptive Layouts in DBMSs: Systems like H2O [2], Hyrise [10], Peloton [22] and OctopusDB [9] manage layout flexibility with either in-place or copy-based (e.g. H2O, OctopusDB) reorganization, offline or online (e.g. Peloton, H2O) reorganization, and for the latter with the choices of carrying it out as a batch background process (e.g. Peloton) or of scheduling reorganization as part of query execution (e.g. H2O). The authors of Peloton have proposed utilizing templates and arrival-rate-based clustering for queries to form representative groups, and to forecast the arrival rate for each group with an ensemble of methods[18] such that special patterns could be predicted. Authors study that a handful of query groups can be sufficient to represent a workload. Per-design, based on the predicted workload Peloton seeks to decide on the suitable layout. In order to achieve a flexible storage model they use a *tile-based architecture*. To our knowledge, reorganization in Peloton is approached using a fixed decision tree of actions or a learned decision tree. H2O shares with Peloton a fine-grained view on data partitioning with a heuristics-based approach to reorganization.

Vertical Partitioning: Strategies for Vertical Partitioning have been proposed as early as the 1980s. A useful evaluation of existing proposals is given in the work of Jindal et. al.[16].

Brute force algorithms enumerate all possible partitions and calculate for each the expected cost for running a pre-defined set of queries (which represent the workloads). Since the number of possible partitions can be large, this approach is not very efficient. Top-down or bottom-up methods improve brute force by starting either from the whole table as a single partition or each column as a partition, respectively. These methods then progress iteratively by finding in each step the best ways of increasing or decreasing the number of partitions by one.

Hill-climb is the most easy to understand bottom-up method. Other bottom-up methods include AutoPart [21], Hyrise [10] and Trojan [17]. AutoPart is similar to Hill-climb but it partitions the table categorically, allowing for more fine-grained partitions, like Peloton or H2O. Unlike them, but similar to our model, the authors allow replication of items across partitions. Hyrise allows only columnar partitions and it uses graph partitioning methods for guiding the optimization process. Trojan layouts frame vertical partitioning as a knapsack problem, authors uses interestingness measures for deciding if it is reasonable to merge partitions. In their work they can also delete replicated partitions when their utility falls below a threshold. Our work enables replication of items but, in order to reduce the action space, we do not consider replication of partitions like Trojan.

Navathe [19] and O2P [15] are examples of top-down methods. Navathe constructs an affinity matrix for columns, according to how their co-usage in the queries for the workload. Navathe partitions the table based on matrix clustering. O2P uses the matrix representation of Navathe but with an online matrix clustering. This makes O2P the only method able to work online. O2P also keeps track of unused partitions for removing them. In our model for rewards based on stochastically generated queries we attempted to mimic this through a penalty.

In our model by making actions very fine grained we support the exploration behaviors of bottom-up and top-down partitioning methods. Similar to O2P we offer a method which can be executed online. Unlike most of these methods we consider much more fine-grained partitions and we add a learning framework to the problem. As a result we select the second alternative in a trade-off between exploring the possible actions exhaustively (deterministic vertical partitioning algorithms) or learning the most efficient way of exploring the action space.

Reinforcement Learning for Database Tuning: To our knowledge, the adoption of Markov Decision Processes for database tuning has been limited. Basu et. al. present a formulation for online tuning w.r.t queries [5], the core MDP presented is similar to our modeling. In their design authors include cost-model learning as part of their framework and test their model for the index selection problem. Unlike their work in our approach we assume a workload as an input, rather than a query at a time. Sharma et. al. argue for the adoption of deep reinforcement learning for automating database administration [26]. Authors map the index selection problem to a reinforcement learning problem, providing an evaluation using DQNs with PostgreSQL.

5 OPEN CHALLENGES AND ONGOING WORK

In this short paper we introduced an early proposal for a reinforcement learning framework to approach online partitioning and layout selection. Our goal is to contribute towards a general and unified solution to frame partitioning as a learning problem that can improve with experience, for future DBMS developments.

Advances in artificial intelligence suggest that it is reasonable to expect learned approaches to outperform exhaustive searches or heuristics. These techniques have characteristics that could enable them to overcome traditional query optimization issues stemming from erroneous hard-coded assumptions. Moreover, parallelism in training, GPU utilization and memory footprint reductions can be expected from leveraging deep-learning methods in tackling such problems.

Though we have some early implementations, our development is very much a work in progress. There is a large space to explore and exploit in our ongoing work, as we seek to integrate GridFormation agents with a state-of-the-art DBMS, evaluate the results and improve the framework.

We identify the following basic challenges, which we seek to address:

- *Scalability*: The combinatorial nature of possible partitions, creating a large action space that impacts training and memory requirements, is a key challenge when including horizontal partitions. Assuming tile-based architectures [3] could help reduce the search space.
  Apart from the use of neural networks to reduce the memory costs, employing approximate data structures for supporting the management becomes important.
- *Accelerating the exploration*: Focusing on search space areas can accelerate the exploration during training, however this might introduce loss of generality for the resulting agent.
- *Encodings and Augmenting the state representation*: In our neural network implementation we create numeric encodings of the partitions. Different encodings could benefit the learning process.
  So far our implementation is data-agnostic. Statistical descriptions of data, such as feature vectors or basic min-max aggregates, can be impactful in horizontal partitioning (e.g. as studied by Sun et. al [27]). Leveraging such data for training, and providing similar descriptions for the resulting partitions from our model is an open research direction. Different query and workload representations (e.g. Query2Vec[14]) could be useful as well for our model.
- *Alignment*: Modeling alignment with indexes and materialized views is a necessary feature for physical design.
- *Cost models for sequence training*: In our formulation the training needs to learn n-sized sequences of optimizations. For this design to be adopted in a DBMS it is necessary to establish a cost/gain model according to the complexity of the sequence to be learned. Thus the configurable actions can be selected in a principled manner.
- *Update of cost estimations*: Since cost estimation (i.e., immediate rewards) is performed in a supporting component, improvements in the estimation need to be fed back into the learned models. We seek to consider alternatives for this.

In exploring solutions for these challenges, selecting goodness measures other than simple immediate performance for one task (i.e, measures like predictability, as evaluated by Borovica et. al [6]), and developing fair tests comparing with methods for more specific partitioning problems, are the essential prerequisites if we are to move the needle forward.

ACKNOWLEDGMENTS

This work was partially funded by the DFG (grant no.: SA 465/50-1).

REFERENCES

[1] Sanjay Agrawal, Vivek Narasayya, and Beverly Yang. 2004. Integrating vertical and horizontal partitioning into automated physical database design. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*. ACM, 359–370.

[2] Ioannis Alagiannis, Stratos Idreos, and Anastasia Ailamaki. 2014. H2O: a hands-free adaptive store. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. ACM, 1103–1114.

[3] Joy Arulraj, Andrew Pavlo, and Prashanth Menon. 2016. Bridging the archipelago between row-stores and column-stores for hybrid workloads. In *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data*. ACM, 583–598.

[4] Tahir Azim, Manos Karpathiotakis, and Anastasia Ailamaki. 2017. ReCache: Reactive Caching for Fast Analytics over Heterogeneous Data. *Proceedings of the VLDB Endowment* 11, 3 (2017).

[5] Debabrota Basu, Qian Lin, Weidong Chen, Hoang Tam Vo, Zihong Yuan, Pierre Senellart, and Stéphane Bressan. 2016. Regularized cost-model oblivious database tuning with reinforcement learning. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems XXVIII*. Springer, 96–132.

[6] Renata Borovica, Ioannis Alagiannis, and Anastasia Ailamaki. 2012. Automated physical designers: what you see is (not) what you get. In *Proceedings of the Fifth International Workshop on Testing Database Systems*. ACM, 9.

[7] Carlo Curino, Evan Jones, Yang Zhang, and Sam Madden. 2010. Schism: a workload-driven approach to database replication and partitioning. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 48–57.

[8] Dinesh Das, Jiaqi Yan, Mohamed Zait, Satyanarayana R Valluri, Nirav Vyas, Ramarajan Krishnamachari, Prashant Gaharwar, Jesse Kamp, and Niloy Mukherjee. 2015. Query optimization in Oracle 12c database in-memory. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1770–1781.

[9] Jens Dittrich and Alekh Jindal. 2011. Towards a One Size Fits All Database Architecture.. In *CIDR, Conference on Innovative Data Systems Research*. 195–198.

[10] Martin Grund, Jens Krüger, Hasso Plattner, Alexander Zeier, Philippe Cudre-Mauroux, and Samuel Madden. 2010. HYRISE: a main memory hybrid storage engine. *Proceedings of the VLDB Endowment* 4, 2 (2010), 105–116.

[11] Paul R Halmos. 2017. *Naive set theory*. Courier Dover Publications.

[12] Richard A Hankins and Jignesh M Patel. 2003. -Data Morphing: An Adaptive, Cache-Conscious Storage Technique. In *Proceedings 2003 VLDB Conference*. Elsevier, 417–428.

[13] Marina Irodova and Robert H Sloan. 2005. Reinforcement Learning and Function Approximation.. In *FLAIRS Conference*. 455–460.

[14] Shrainik Jain and Bill Howe. 2018. Query2Vec: NLP Meets Databases for Generalized Workload Analytics. *arXiv preprint arXiv:1801.05613* (2018).

[15] Alekh Jindal and Jens Dittrich. 2011. Relax and let the database do the partitioning online. In *International Workshop on Business Intelligence for the Real-Time Enterprise*. Springer, 65–80.

[16] Alekh Jindal, Endre Palatinus, Vladimir Pavlov, and Jens Dittrich. 2013. A comparison of knives for bread slicing. *Proceedings of the VLDB Endowment* 6, 6 (2013), 361–372.

[17] Alekh Jindal, Jorge-Arnulfo Quiané-Ruiz, and Jens Dittrich. 2011. Trojan data layouts: right shoes for a running elephant. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 21.

[18] Lin Ma, Dana Van Aken, Ahmed Hefny, Gustavo Mezerhane, Andrew Pavlo, and Geoffrey J. Gordon. 2018. Query-based Workload Forecasting for Self-Driving Database Management Systems. In *Proceedings of the 2018 ACM SIGMOD International Conference on Management of Data (SIGMOD '18)*. 15. https://db.cs.cmu.edu/papers/2018/mod435-maA.pdf

[19] Shamkant Navathe, Stefano Ceri, Gio Wiederhold, and Jinglie Dou. 1984. Vertical partitioning algorithms for database design. *ACM Transactions on Database Systems (TODS)* 9, 4 (1984), 680–710.

[20] Rimma Nehme and Nicolas Bruno. 2011. Automated partitioning design in parallel database systems. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM, 1137–1148.

[21] Stratos Papadomanolakis and Anastassia Ailamaki. 2004. Autopart: Automating schema design for large scientific databases using data partitioning. In *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*. IEEE, 383–392.

[22] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd C Mowry, Matthew Perron, Ian Quah, et al. 2017. Self-Driving Database Management Systems.. In *CIDR, Conference on Innovative Data Systems Research*.

[23] Andrew Pavlo, Carlo Curino, and Stanley Zdonik. 2012. Skew-aware automatic database partitioning in shared-nothing, parallel OLTP systems. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 61–72.

[24] Marcus Pinnecke, David Broneske, Gabriel Campero Durand, and Gunter Saake. 2017. Are databases fit for hybrid workloads on GPUs? A storage engine's perspective. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*. IEEE, 1599–1606.

[25] Philipp Rösch, Lars Dannecker, Franz Färber, and Gregor Hackenbroich. 2012. A storage advisor for hybrid-store databases. *Proceedings of the VLDB Endowment* 5, 12 (2012), 1748–1758.

[26] Ankur Sharma, Felix Martin Schuhknecht, and Jens Dittrich. 2018. The Case for Automatic Database Administration using Deep Reinforcement Learning. *arXiv preprint arXiv:1801.05643* (2018).

[27] Liwen Sun, Michael J Franklin, Sanjay Krishnan, and Reynold S Xin. 2014. Fine-grained partitioning for aggressive data skipping. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. ACM, 1115–1126.

[28] Richard S Sutton and Andrew G Barto. 1998. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge.