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

Gabriel Campero Durand, Marcus Pinnecke, Rufat Piriyev, Mahmoud Mohsen, David Broneske, Gunter Saake, Maya S. Sekeran, Fabián Rodriguez, Laxmi Balami
Our Team
Our Team

Databases and Software Engineering Workgroup, University of Magdeburg
Agenda:

1. Intro + Research Question
2. Case Study: Vertical Partitioning
   a. Components of a general RL approach
3. Early Results
4. Research Agenda and Open Questions

Our focus today

Databases and Software Engineering Workgroup, University of Magdeburg
1. Intro + Research Question
Physical Design:

To determine the best storage configuration for a given workload: indexes, layouts, partitions, column-orders, etc.

Core task in database management.

Especially relevant for novel data management designs, where adaptive techniques are adopted.
Physical Design:
An embarrassment of riches:

Selected examples:
Static Choices:
- Horizontal partitions & compression per column or table types.
- Parquet-like formats

Adaptive Partitioning and Layouts:
- OctopusDB[2], Peloton[3]

Partitioning Algorithms

Our target use case
No Replication
Introduces cost model

Replication Query-based partitioning

Databases and Software Engineering Workgroup, University of Magdeburg
Automated Physical Design: Finding Common Ground

- Usually an optimization problem (offline or online)
  - Large spaces of combinations (possible solutions)
  - Evaluated with a cost model
- Pruning is needed:
  - When the reward does not improve after a step
  - By focusing on interesting areas only
- Usually memory-less
  
  “Traditional systems plan a query, execute the query plan, and forget they ever optimized this query.” [6]

Without memory, how to catch and improve errors from relying on the cost-model only?
Our Research Question:

- Would it be possible to build a **common and general approach** able to bring together existing solutions, and **successfully address new use cases**, by leveraging experience, in an **online** manner?

  => Considering that **Reinforcement Learning** fits the domain, plus recent technical advances in deep RL, we selected to study its utility to answer our RQ.
Our Research Proposal:

What RL brings to the table:

- A way to store **experience** and learn from it
- **Sequentaility** + **Instant recommendations** (online-friendly)
- **Scalable** w.r.t memory budgets (when using NNs)
- Offline exploration with a baseline of standard exploration policies
- Opportunities for designing without relying only on cost models (which could be flawed)
- Active research field with opportunities for extensions
Known Antecedents: (in RL for physical design)

- **COREIL** [7], National University of Singapore, 2015
  - Case study: Index selection
  - Policy iteration
  - Including penalty with cost
  - Single-query workload
  - Concepts for focusing the search

- **NoDB** [8], Saarland University, Germany, 2018
  - Index selection
  - DQN
  - Generically trained on 5 query workload
  - Evaluated against index all vs. no index in Postgres
RL Basics:

- Generate own data by interacting with environment, learn patterns
- Useful when exploration is cheap
- Components:
  - Agent
  - Environment
    - Actions
    - State
    - Transition probabilities given state and action
    - Rewards
RL Basics:

- RL models can learn:
  - The sequence of actions to win a game (policy)
  - The long-term rewards expected for actions, given a state (value)
- In the learning process, using some parameters, a target function is fitted.
- Learned values kept in either basic tables or approximations with models, like neural nets.
- Exploration is guided through a policy (e.g. $\varepsilon$-greedy, Boltzmann, etc.)
- Numerous methods exist: Q-Learning, Actor critic, DDPG, etc.
2. Case Study: Vertical Partitioning
Vertical Partitioning:
A quick overview:

- **Offline:**
  - Brute Force, HillClimb[9], AutoPart[5], Navathe[10], Hyrise[4], Trojan[11]

- **Online:**
  - O2P[12], Amoeba[13]
Vertical Partitioning:
A quick overview:

(b) Lineitem

Source: Jindal et al.[14]
Vertical Partitioning:
A quick overview:

Figure 1: Optimization time for different algorithms

Figure 3: Estimated workload runtime for different algorithms
What are we learning on VP here?

- **Value:** The long-term value of actions given a state.
- **A recommendation of sequences:** The sequence of \( k \) steps that reaches the best partitioning, while running parallel to the workload.
- **User-defined generality:**
  - From state only \( \Rightarrow \) *we learn for a workload*
  - From workload \( \Rightarrow \) *we could learn for several workloads*
  - From some table data \( \Rightarrow \) *we could learn for several tables*
VP RL Components:

Agent

Get actions (given a state (also wkld & table) + Act

Environment

Get cost of action, for rewards

Cost Estimator

Get sequence of actions to apply (only inference)

Get forecasted workload

Workload forecaster

Table info (for env.init())

Database

Learn values of actions, Select action, based on values learned (Expl. policy)

Train (Offline)

Deploy (Online)

Databases and Software Engineering Workgroup, University of Magdeburg
Actions:

- **Very basic possible actions:**
  - Merge 2 vertical partitions
  - Split a partition in 2
- **Transitions:** deterministic
  - An action can be represented as a new state => **Configurable actions**
- **State representations:**
  - Sets of sets
  - **Encodings** for NNs
Environment:

- **Provides actions available**
  - Not the same as Keras-RL

- **Costs provided by external component**
  - Rewards are given as 1/cost.

- **Both stochastic & deterministic rewards work**
  - Usual approach:
    - Size of partition
    - Add costs per query per referenced partition
  - Stochastic and with replication:
    - Sample queries
    - Determine cost for most-similar partition
    - Penalize unused partitions with replication
Agents:

- User-defined generality
- In charge of the exploration strategies
- Tuneable replay priority
Our Team
1. Intro + Research Question
2. Case Study: Vertical Partitioning
   a. Components of a general RL approach
3. Early Results
4. Research Agenda and Open Questions
To determine the best storage configuration for a given workload:
indexes, layouts, partitions, column-orders, etc.

Core task in database management.

Especially relevant for novel data management designs, where adaptive techniques are adopted.
Physical Design: An embarrassment of riches:

Selected examples:

**Static Choices:**
- Horizontal partitions & compression per column or table types.
- Parquet-like formats

**Adaptive Partitioning and Layouts:**
- OctopusDB[2], Peloton[3]

**Partitioning Algorithms**

Replication
- Query-based partitioning

Our target use case
- No Replication

Introduces cost model

OctopusDB[3]

Peloton[4]
Automated Physical Design: Finding Common Ground

- Usually an **optimization problem** (offline or online)
  - Large spaces of combinations (**possible solutions**)
  - Evaluated with a **cost model**

- **Pruning is needed**:
  - When the reward does not improve after a step
  - By focusing on interesting areas only

- **Usually memory-less**
  
  "Traditional systems plan a query, execute the query plan, and forget they ever optimized this query." [6]

Without memory, how to catch and improve errors from relying on the cost-model only?
Our Research Question:

- Would it be possible to build a **common and general approach** able to bring together existing solutions, and **successfully address new use cases**, by leveraging experience, in an **online** manner?

⇒ Considering that **Reinforcement Learning** fits the domain, plus recent technical advances in deep RL, we selected to study its utility to answer our RQ.
Our Research Proposal:
What RL brings to the table:

- A way to store **experience** and learn from it
- **Sequentiality** + **Instant recommendations** (online-friendly)
- **Scalable** w.r.t memory budgets (when using NNs)
- Offline exploration with a baseline of standard exploration policies
- Opportunities for designing without relying only on cost models (which could be flawed)
- Active research field with opportunities for extensions
Known Antecedents: (in RL for physical design)

- **COREIL** [7], National University of Singapore, 2015
  - Case study: Index selection
  - Policy iteration
  - Including penalty with cost
  - Single-query workload
  - Concepts for focusing the search

- **NoDB** [8], Saarland University, Germany, 2018
  - Index selection
  - DQN
  - Generically trained on 5 query workload
  - Evaluated against index all vs. no index in Postgres
RL Basics:

- Generate own data by interacting with environment, learn patterns
- Useful when exploration is cheap
- Components:
  - Agent
  - Environment
    - Actions
    - State
    - Transition probabilities given state and action
    - Rewards
RL Basics:

- RL models can learn:
  - The sequence of actions to win a game (policy)
  - The long-term rewards expected for actions, given a state (value)
- In the learning process, using some parameters, a target function is fitted.
- Learned values kept in either basic tables or approximations with models, like neural nets.
- Exploration is guided through a policy (e.g. $\epsilon$-greedy, Boltzmann, etc.)
- Numerous methods exist: Q-Learning, Actor critic, DDPG, etc.
2. Case Study: Vertical Partitioning
Vertical Partitioning: A quick overview:

- **Offline:**
  - Brute Force, HillClimb[9], AutoPart[5], Navathe[10], Hyrise[4], Trojan[11]

- **Online:**
  - O2P[12], Amoeba[13]
Vertical Partitioning:
A quick overview:

(b) Lineitem

Source: Jindal et al.[14]
Vertical Partitioning:
A quick overview:

Source: Jindal et al. [14]

Figure 1: Optimization time for different algorithms

Figure 3: Estimated workload runtime for different algorithms
What are we learning on VP here?

- **Value:** The long-term value of actions given a state.
- **A recommendation of sequences:** The sequence of $k$ steps that reaches the best partitioning, while running parallel to the workload.
- **User-defined generality:**
  - From state only $\rightarrow$ *we learn for a workload*
  - From workload $\rightarrow$ *we could learn for several workloads*
  - From some table data $\rightarrow$ *we could learn for several tables*
VP RL Components:

- **Agent**: Get actions (given a state (also workload & table) + Act
- **Environment**: Get cost of action, for rewards
- **Cost Estimator**: Get sequence of actions to apply (only inference)
- **Workload forecaster**: Get sequence of actions to apply (only inference)
- **Database**: Get costs of action, for rewards

Learn values of actions, Select action, based on values learned (Expl. policy)

Train (Offline) Deploy (Online)

Databases and Software Engineering Workgroup, University of Magdeburg
Actions:

- **Very basic possible actions:**
  - Merge 2 vertical partitions
  - Split a partition in 2

- **Transitions:** deterministic
  - An action can be represented as a new state \(\Rightarrow\) **Configurable actions**

- **State representations:**
  - Sets of sets
  - **Encodings** for NNs
Environment:

- **Provides actions available**
  - Not the same as Keras-RL

- **Costs provided by external component**
  - Rewards are given as $1/cost$.

- **Both stochastic & deterministic rewards work**
  - Usual approach:
    - Size of partition
    - Add costs per query per referenced partition
  - Stochastic and with replication:
    - Sample queries
    - Determine cost for most-similar partition
    - Penalize unused partitions with replication
Agents:

- User-defined generality
- In charge of the exploration strategies
- Tuneable replay priority
3. Early Results
3. Early Results
Setup for Early Results

Intel® Core™ i7-6700 CPU @ 3.40GHz × 8
31,3 GiB Memory
Ubuntu 16.04

Implementation on OpenAI Gym, TF and Keras

Cost Model: HDD costs based on Jindal et al.[1]
Agent 1: Fixed Workload

Results for 32 input, 2 hidden layers (4 N each), 1 output.

4 step partitioning on Lineitem for TPC-H Queries as workload.

Our inference time needs to be improved for online applicability.

By design we should be much more competitive.

1.82X faster than AutoPart, 5x slower than O2P. Only AutoPart and RL find the optima

Runtime of the algorithms vs. inference
Agent 1: Fixed Workload

Different convergence rates for simple tasks with naive Boltzmann exploration, using Q-tables. *There’s room for improvement in exploration, to focus on interesting areas.*
Current Takeaways

1. Agents can be developed for different generalities + with configurable actions => A lot of DBMS designs from today could share a common framework

2. Online applicability:
   a. Physical design as a sequential problem
   b. Fast inference is expected

3. Exploration is quite challenging

4. Can we integrate measurements, such that we could gain more from the RL approach?
4. Research Agenda and Open Questions
Research Agenda:

1. **Improvements in inference**
2. Make public as an academic library and plugin
3. **Understanding and accelerating exploration**
   - Progress bar, Model for sequence training
   - Affinity or interestingness-based prioritization
4. The role of encodings (embeddings) for partitions
5. **Coupling with a DBMS:**
   - Comprehensive evaluations and outline of limitations
   - Opportunities for robustness via $\varepsilon$-greedy with measured rewards $\leq$ Beginning with index selection.
   - Align partitions with indexes
6. Extensions (e.g. changes in cost estimation, addressing similar problems)

Our early experience with stochastic rewards suggests this is feasible.
Thanks a ton!

- To reviewers + organizers for this opportunity to share our early ideas.
- Thx for the attention too :)

campero@ovgu.de

Toy examples: [https://github.com/gabrielcc2/gridformation_examples](https://github.com/gabrielcc2/gridformation_examples) (available within this week)
We will open source (within this month) our development process.
References


References


References


References
