GridFormation:

Towards Self-Driven Online Data Partitioning Using Reinforcement Learning

Early Work

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

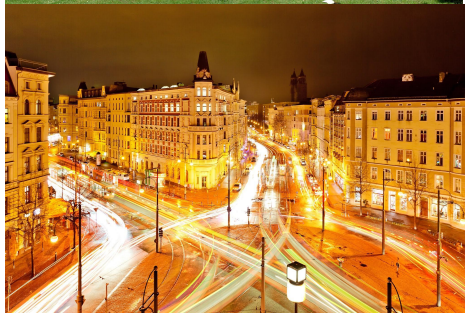
Our Team

STUDY PROGRAMME

Data and Knowledge Engineering

Our Team


Hamburg
Bremerhaven
Wilhelmshaven
Schwerin
Neubrandenburg
Waren
Leer
Oldenburg
Bremen
Lüneburg
Celle
Hanover
Wolfsburg
Berlin
Potsdam
Osnabrück
Brunswick
Magdeburg
Bielefeld
Münster
Paderborn
Dortmund
Göttingen
Halle (Saale)
Leipzig
Kassel
Germany
Dresden
Cologne
Jena
Chemnitz
Bonn
Zwickau
Fulda
Koblenz
Karlovy Vary
Frankfurt
Bayreuth
Mainz
Würzburg
Bamberg
Pilsen
Trier
Mannheim
Nuremberg
Heidelberg
Karlsruhe
Regensburg
Stuttgart
Straubing
Ingolstadt
Passau
Strasbourg
Ulm
Augsburg
Wildlife Park
Parc naturel
Munich



Agenda:

Our focus today

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

1. Intro + Research Question

Physical Design:

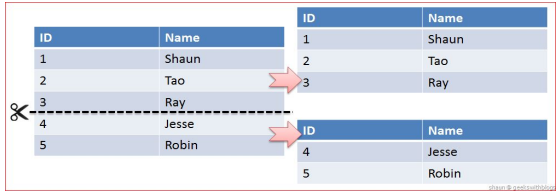
1

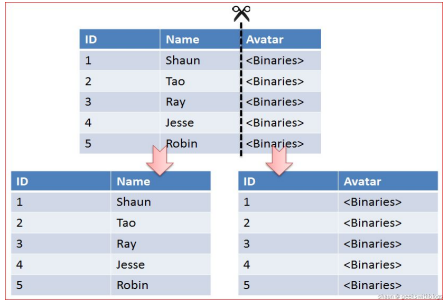
To determine the best storage configuration for a given workload:

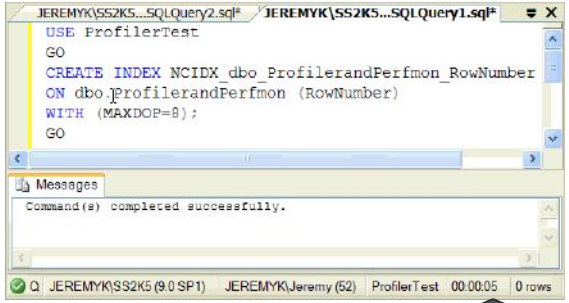
indexes, layouts, partitions, column-orders, etc.

Core task in database management.

Especially relevant for novel data management designs, where adaptive techniques are adopted.







Source: http://gwb.blob.core.windows.net/

Physical Design:

An embarrassment of riches:

1

Selected examples:

Static Choices:

Horizontal partitions & compression per column or table types.

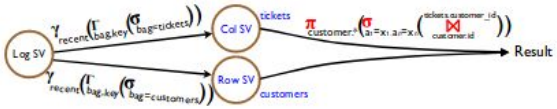
Parquet-like formats

Adaptive Partitioning and Layouts:

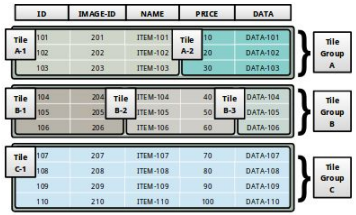
OctopusDB[2], Peloton[3]

Partitioning Algorithms

AutoPart[4] (offline), Hyrise[5] (main-memory)



OctopusDB[3]



Peloton[4]

Our target use case

Replication
Query-based partitioning

No Replication

Introduces cost model

Replication

Automated Physical Design: Finding Common Ground

1

- 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:

1

- 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:

1

- 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

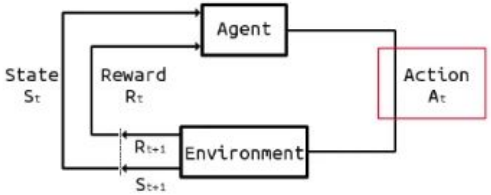
Known Antecedents:
(in RL for physical design)

1

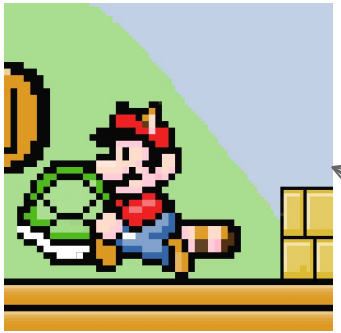
- **COREIL** [7], National University of Singapore, 2015
  - Case study: Index selection
  - Policy iteration
  - Including penalty with cost
  - Single-query workload
  - Concepts for focusing the search
- **NoDB** [8], Saarland University, Germany, 2018
  - Index selection
  - DQN
  - Generically trained on 5 query workload
  - Evaluated against index all vs. no index in Postgres

Quick Recap

RL Basics:

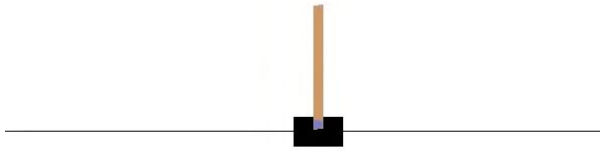


- Generate own data by interacting, with environment, learn patterns
- Useful when exploration is cheap
- Components:
  - Agent
  - Environment
    - Actions
    - State
    - Transition probabilities given state and action
    - Rewards



Quick Recap

RL Basics:

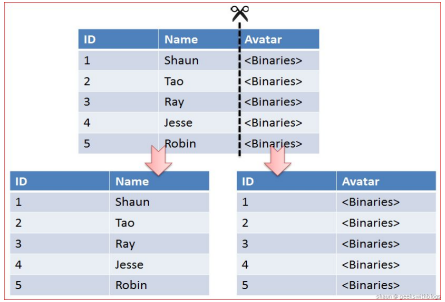


- RL models can learn:
  - The sequence of actions to win a game (**policy**)
  - The long-term rewards expected for actions, given a state (**value**)
- In the learning process, using some parameters, a target function is fitted.
- Learned values kept in either basic tables or approximations with models,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

2. Case Study: Vertical Partitioning

Vertical Partitioning: A quick overview:

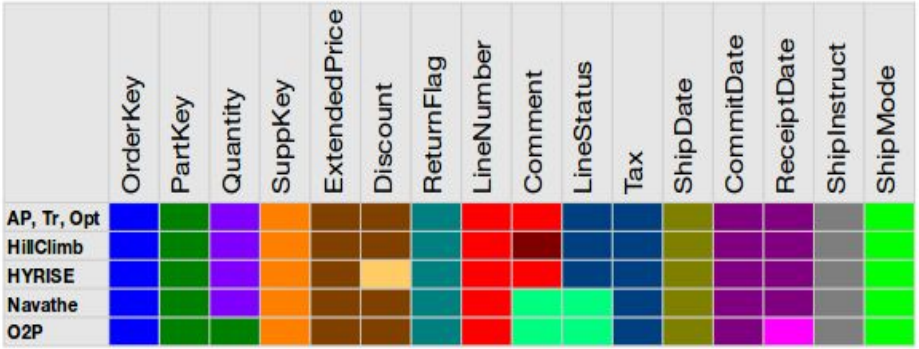
2



- **Offline:**
  - Brute Force, HillClimb[9], AutoPart[5], Navathe[10], Hyrise[4], Trojan[11]
- **Online:**
  - O2P[12], Amoeba[13]

Vertical Partitioning:
A quick overview:

2



(b) Lineitem

A Comparison of Knives for Bread Slicing

Alekh Jindal*,† Endre Palatinus* Vladimir Pavlov* Jens Dittrich*

*CSAIL, MIT
http://csail.mit.edu

*Information Systems Group, Saarland University
http://infosys.cs.uni-saarland.de

Source: Jindal et al.[14]

Vertical Partitioning:
A quick overview:

2

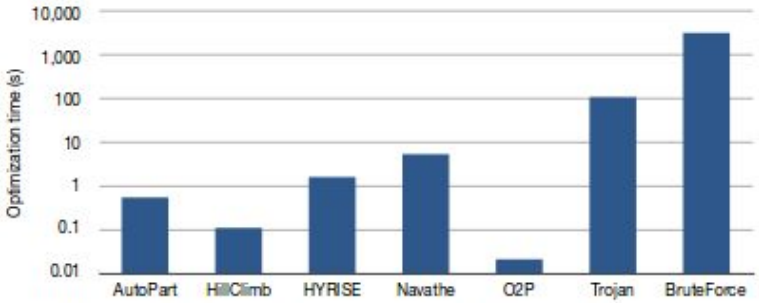


Figure 1: Optimization time for different algorithms

A Comparison of Knives for Bread Slicing

Alekh Jindal*,† Endre Palatinus* Vladimir Pavlov* Jens Dittrich*

*CSAIL, MIT
http://csail.mit.edu

*Information Systems Group, Saarland University
http://infosys.cs.uni-saarland.de

Source: Jindal et al.[14]

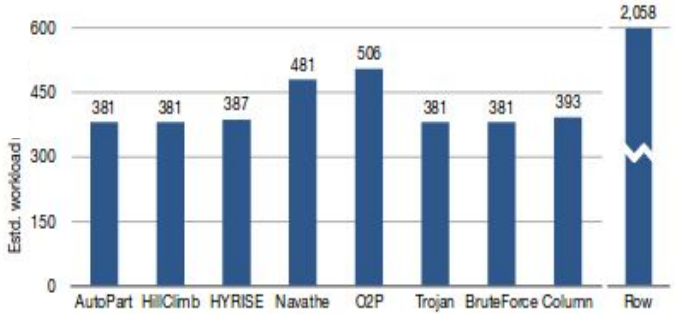


Figure 3: Estimated workload runtime for different algorithms

What are we learning on VP here?

2

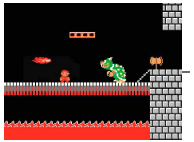
- **Value:** The long-term value of actions given a state.
- **A recommendation of sequences:** The sequence of k steps that reaches the best partitioning, while running parallel to the workload.
- **User-defined generality:**
  - From state only - > *we learn for a workload*
  - From workload -> *we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

VP RL Components:

2

Agent

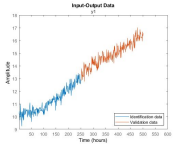
Get actions (given a state (also wkld & table) + Act



Environment

Get cost of action, for rewards

Cost Estimator



Workload forecaster

Get forecasted workload

Table info (for env.init())

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

Get sequence of actions to apply (only inference)

Database

Train (Offline)

Deploy (Online)

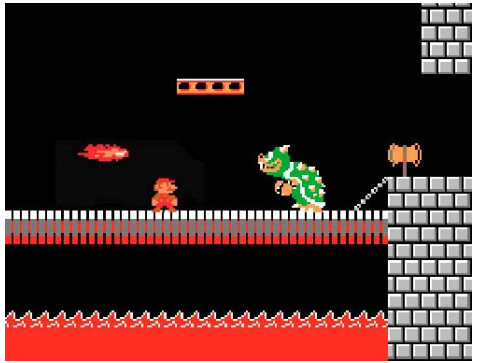
Actions:

2

- **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:

2



- **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:

2

- **User-defined generality**
- In charge of the **exploration strategies**
- Tuneable **replay priority**

Our Team

STUDY PROGRAMME

Data and Knowledge Engineering

Agenda:

Our focus today

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

Physical Design:

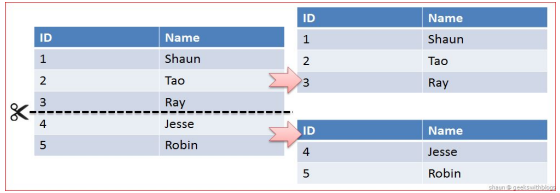
1

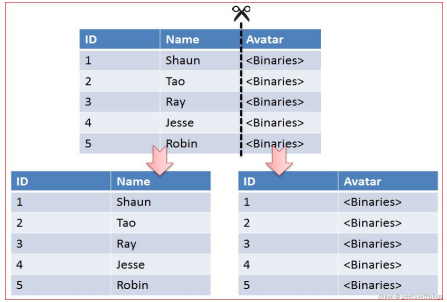
To determine the best storage configuration for a given workload:

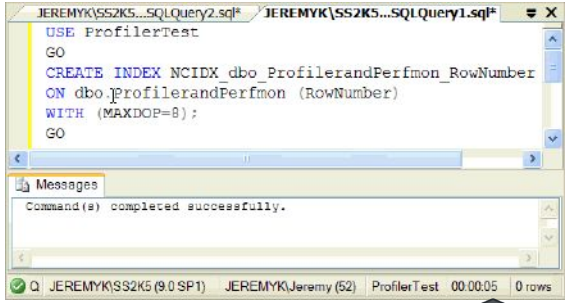
indexes, layouts, partitions, column-orders, etc.

Core task in database management.

Especially relevant for novel data management designs, where adaptive techniques are adopted.







Source: http://gwb.blob.core.windows.net/

Physical Design:

An embarrassment of riches:

1

Selected examples:

Static Choices:

Horizontal partitions & compression per column or table types.

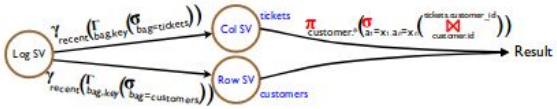
Parquet-like formats

Adaptive Partitioning and Layouts:

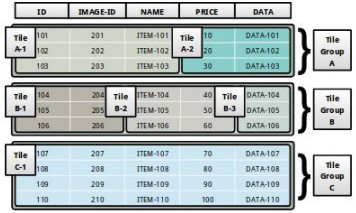
OctopusDB[2], Peloton[3]

Partitioning Algorithms

AutoPart[4] (offline), Hyrise[5] (main-memory)



OctopusDB[3]



Peloton[4]

Replication
Query-based partitioning

Our target use case

No Replication

Replication

Introduces cost model

Automated Physical Design: Finding Common Ground

1

- 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:

1

- 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:

1

- 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

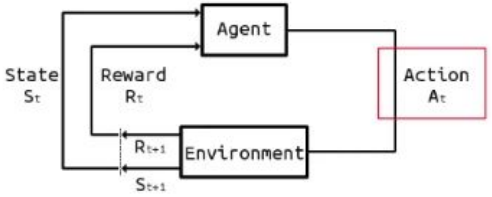
Known Antecedents:
(in RL for physical design)

1

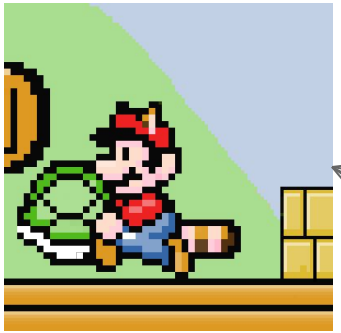
- **COREIL** [7], National University of Singapore, 2015
  - Case study: Index selection
  - Policy iteration
  - Including penalty with cost
  - Single-query workload
  - Concepts for focusing the search
- **NoDB** [8], Saarland University, Germany, 2018
  - Index selection
  - DQN
  - Generically trained on 5 query workload
  - Evaluated against index all vs. no index in Postgres

Quick Recap

RL Basics:

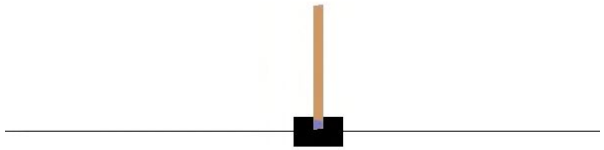


- Generate own data by interacting, with environment, learn patterns
- Useful when exploration is cheap
- Components:
  - Agent
  - Environment
    - Actions
    - State
    - Transition probabilities given state and action
    - Rewards



Quick Recap

RL Basics:

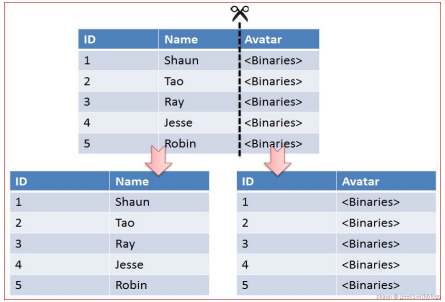


- RL models can learn:
  - The sequence of actions to win a game (**policy**)
  - The long-term rewards expected for actions, given a state (**value**)
- In the learning process, using some parameters, a target function is fitted.
- Learned values kept in either basic tables or approximations with models,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

2. Case Study: Vertical Partitioning

Vertical Partitioning: A quick overview:

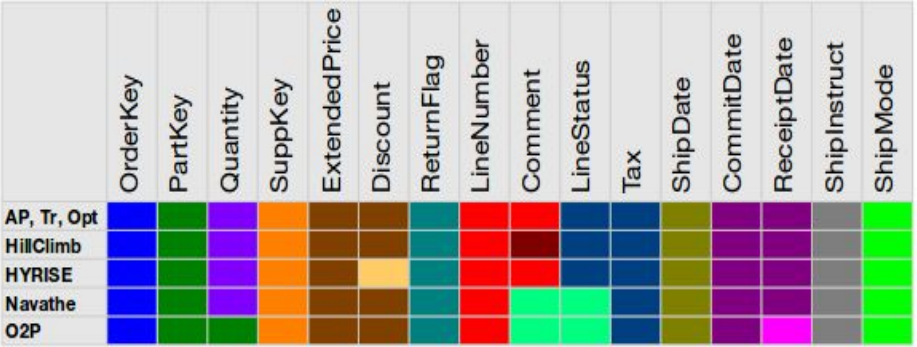
2



- **Offline:**
  - Brute Force, HillClimb[9], AutoPart[5], Navathe[10], Hyrise[4], Trojan[11]
- **Online:**
  - O2P[12], Amoeba[13]

Vertical Partitioning:
A quick overview:

2



(b) Lineitem

A Comparison of Knives for Bread Slicing

Alekh Jindal*,† Endre Palatinus* Vladimir Pavlov* Jens Dittrich*

*CSAIL, MIT
http://csail.mit.edu

*Information Systems Group, Saarland University
http://infosys.cs.uni-saarland.de

Source: Jindal et al.[14]

Vertical Partitioning:
A quick overview:

2

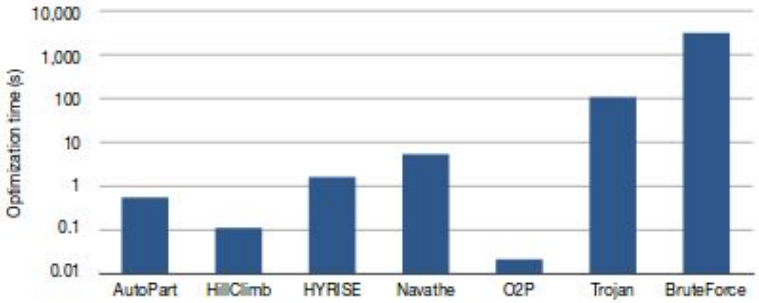


Figure 1: Optimization time for different algorithms

A Comparison of Knives for Bread Slicing

Alekh Jindal*,† Endre Palatinus* Vladimir Pavlov* Jens Dittrich*

*CSAIL, MIT
http://csail.mit.edu

*Information Systems Group, Saarland University
http://infosys.cs.uni-saarland.de

Source: Jindal et al.[14]

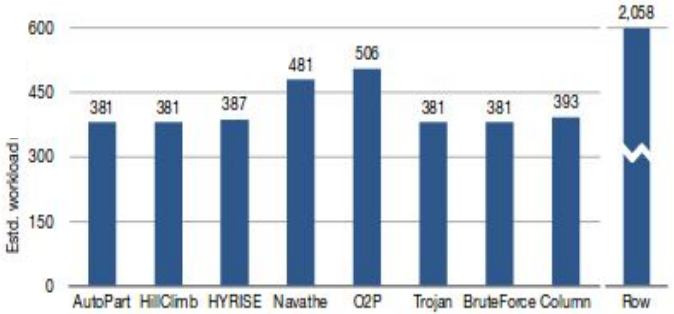


Figure 3: Estimated workload runtime for different algorithms

What are we learning on VP here?

2

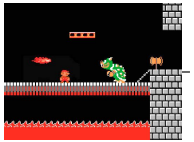
- **Value:** The long-term value of actions given a state.
- **A recommendation of sequences:** The sequence of k steps that reaches the best partitioning, while running parallel to the workload.
- **User-defined generality:**
  - From state only - > *we learn for a workload*
  - From workload -> *we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

VP RL Components:

2

Agent

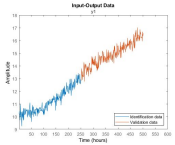
Get actions (given a state (also wkld & table) + Act



Environment

Get cost of action, for rewards

Cost Estimator



Workload forecaster

Get forecasted workload

Table info (for env.init())

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

Get sequence of actions to apply (only inference)

Database

Train (Offline)

Deploy (Online)

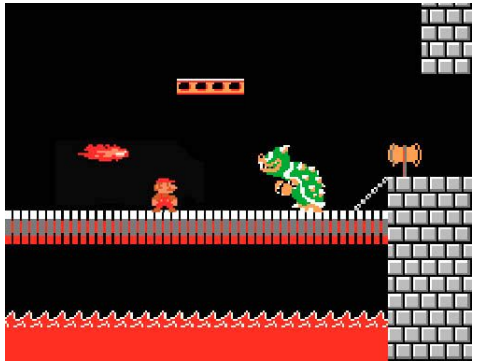
Actions:

2

- **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:

2



- **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:

2

- **User-defined generality**
- In charge of the **exploration strategies**
- Tuneable **replay priority**

3. Early Results

3. Early Results

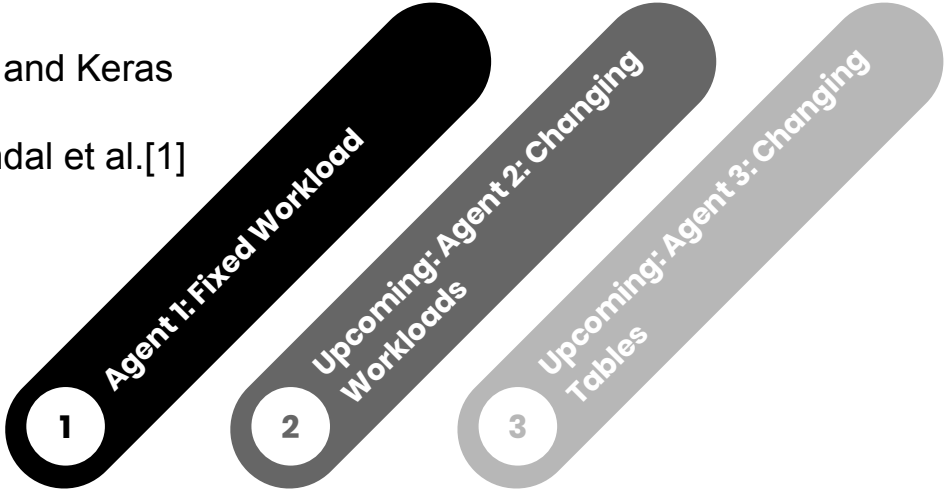
Setup for Early Results

3

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]




```
# Based on HDD costs from:  Jindal, Alekh, Endre Palatinus, Vladimir Pavlov, and Jens D
def _getCostForPartition(self, prs, rrs):
    bufferSize = 8 * 1024 * 1024
    blockSize = 8 * 1024  # type: int
    seekTime = 0.008
    readDiskBW = 92 * 1024 * 1024
    partitionBufferSize = max(math.floor(bufferSize * prs / rrs), blockSize) # we have
    # This is the number of blocks that fit into the buffer for the current partition.
    blocksReadPerBuffer = math.floor(partitionBufferSize / blockSize)
    # This is the total number of blocks of the current partition
    numberOfBlocks = math.ceil(prs * self.num_rows / blockSize)
    #Time spent on seeking and scanning.
    seekCost = seekTime * math.ceil(numberOfBlocks / blocksReadPerBuffer)
    scanCost = numberOfBlocks * blockSize / readDiskBW
    return seekCost + scanCost
```

Agent 1: Fixed Workload

3

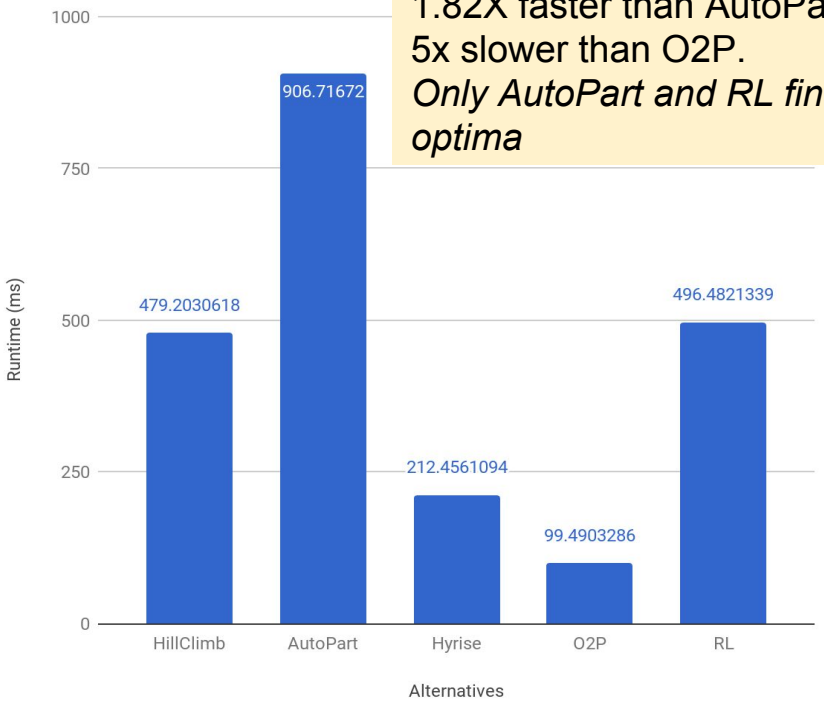
Results for 32 input, 2 hidden layers (4 N each), 1 output.

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.

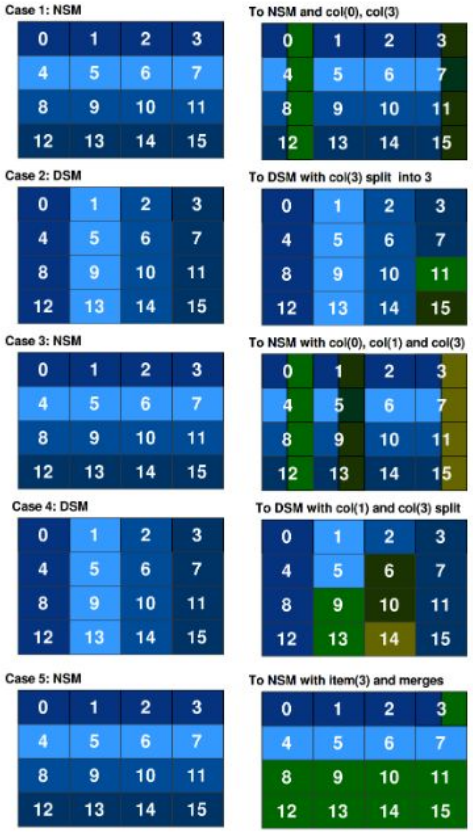
1.82X faster than AutoPart, 5x slower than O2P.
Only AutoPart and RL find the optima

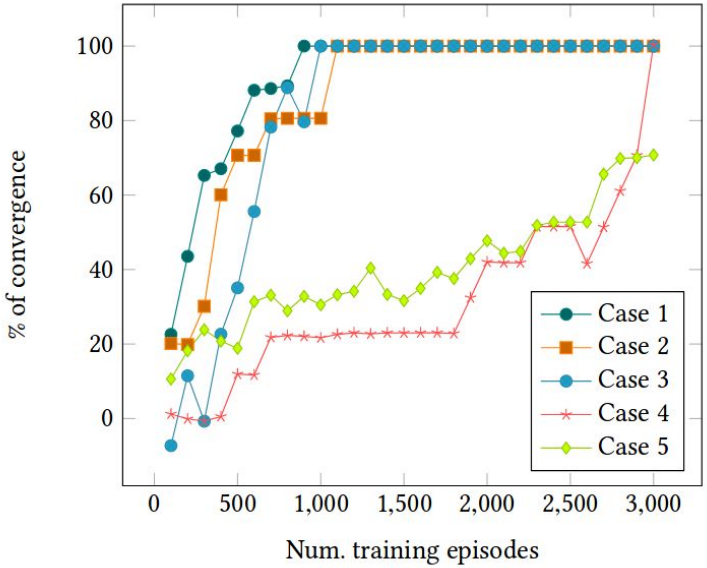


Runtime of the algorithms vs. inference

Agent 1: Fixed Workload

3





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

3

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:

4

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 ε-greedy with measured rewards <= 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
(available within this week)
We will open source (within this month)
our development process.

References

[1] Färber, Franz, Norman May, Wolfgang Lehner, Philipp Große, Ingo Müller, Hannes Rauhe, and Jonathan Dees. "The SAP HANA Database--An Architecture Overview." IEEE Data Eng. Bull. 35, no. 1 (2012): 28-33.

[2] Jindal, Alekh. "The mimicking octopus: Towards a one-size-fits-all database architecture." In VLDB PhD Workshop, pp. 78-83. 2010.

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

[4] Grund, Martin, Jens Krüger, Hasso Plattner, Alexander Zeier, Philippe Cudre-Mauroux, and Samuel Madden. "HYRISE: a main memory hybrid storage engine." Proceedings of the VLDB Endowment 4, no. 2 (2010): 105-116.

References

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

[6] Marcus, Ryan, and Olga Papaemmanouil. "Deep reinforcement learning for join order enumeration." arXiv preprint arXiv:1803.00055 (2018).

[7] Basu, Debabrota, Qian Lin, Weidong Chen, Hoang Tam Vo, Zihong Yuan, Pierre Senellart, and Stéphane Bressan. "Cost-model oblivious database tuning with reinforcement learning." In International Conference on Database and Expert Systems Applications, pp. 253-268. Springer, Cham, 2015.

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

References

[9] R. A. Hankins and J. M. Patel. Data Morphing: An Adaptive, Cache-Conscious Storage Technique. In VLDB, pages 417–428, 2003

[10] S. Navathe, S. Ceri, G. Wiederhold, and J. Dou. Vertical Partitioning Algorithms for Database Design. ACM TODS, 9(4):680–710, 1984.

[11] A. Jindal, J.-A. Quiaṅe-Ruiz, and J. Dittrich. Trojan Data Layouts: Right Shoes for a Running Elephant. In ACM SOCC, pages 21:1–21:14, 2011.

[12] A. Jindal and J. Dittrich. Relax and let the database do the partitioning online. In BIRTE, pages 65–80, 2011.

References

[13] Shanbhag, Anil, Alekh Jindal, Samuel Madden, Jorge Quiane, and Aaron J. Elmore. "A robust partitioning scheme for ad-hoc query workloads." In Proceedings of the 2017 Symposium on Cloud Computing, pp. 229-241. ACM, 2017.

[14]Jindal, Alekh, Endre Palatinus, Vladimir Pavlov, and Jens Dittrich. "A comparison of knives for bread slicing." Proceedings of the VLDB Endowment 6, no. 6 (2013): 361-372.