Novel Applications of AI Techniques in Database Management

Johannes Wünsche

July 15, 2018

Agenda

1. Overview
2. Short Summary of Natural Language Interfaces, ML in DB and Data Managment Applications
3. Self-Management of Databases
4. Software 2.0

Overview

Overview

1 Natural Language Interfaces

1 Data Management Applications

2 Self-Management of Databases X

1 Machine Learning in Databases

3 Optimization by "Software 2.0" X

Short Summary of Natural Language Interfaces, ML in DB and Data Managment Applications

Natural Language Interfaces & ML in DB

Natural Language Interfaces

- Idea of using AI to interpret NL questions or requests and react accordingly
- quite old concept improved by modern hardware and technology

Machine Learning in Databases

- machine learning methods on data
- e.g. *Apache MADlib*, which implements
  - decision trees
  - random forest
  - bayes classifier
  - clustering
  - association rules
  - ...

Data Management Applications

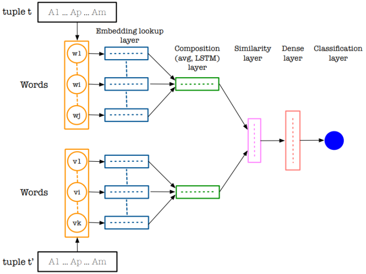


Figure 1: *by DeepER-Deep Entity Resolution, Muhammad Ebraheem et al*

Entity Resolution

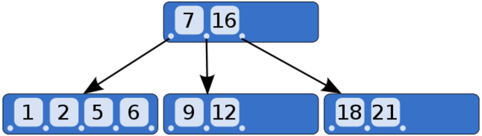
- Finding of Records that refer to the same entity
- Required if shape of data is not unitary

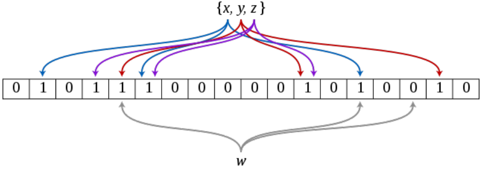
Self-Management of Databases

Self-Management of Databases

- *Tuning of parameters of Database Management Systems*
  - like cache amount and frequency of writing to storage
  - implementations like OtterTune *by Database Research Group at Carnegie Mellon University*
- *Elastic Scaling of Machine Allocation*
  - avoid latency spikes by action prediction through time-series prediction
  - implementations like P-Store *by Taft, MIT*
- *Learned Index Structures*
  - a “model can learn the sort order or structure of lookup keys and use this signal to effectively predict the position or existence of records”
  - alternative technology to exisiting Bloom-Filters or B-Trees

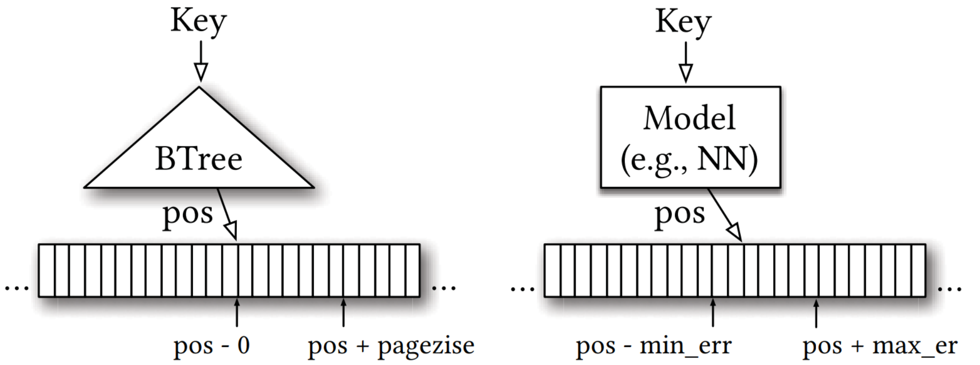
B-Tree Bloom-Filter





Learned Index Structures

What's the difference between Learned Index Structures and existing Models?



Learned Index Structures

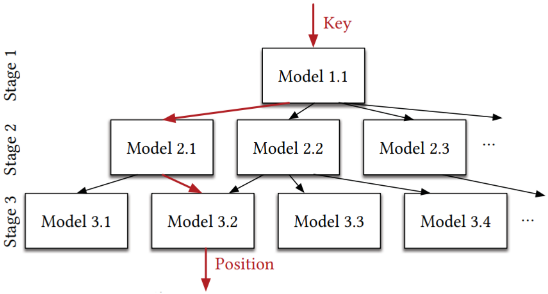
- Performance of Index Access can be unintuitively enhanced by predicting the index of a searched instance with a Neural Network or Linear Regression
- Shown to result in equally good or better performance than conventional Index Structures

Learned Index Structures by Example

Searching for a index

- Similar to every other model used
- Transform key to vector and use as input for trained NN

Enhancement of LIS: Recursive Model Index(RMI)



- refinement of models after every step to better display details
-

Learned Index Structucures Result

		Map Data			Web Data			Log-Normal Data		
Type	**Config**	Size (MB)	Lookup (ns)	Model (ns)	Size (MB)	Lookup (ns)	Model (ns)	Size (MB)	Lookup (ns)	Model (ns)
Btree	page size: 32	52.45 (4.00x)	274 (0.97x)	198 (72.3%)	51.93 (4.00x)	276 (0.94x)	201 (72.7%)	49.83 (4.00x)	274 (0.96x)	198 (72.1%)
	page size: 64	26.23 (2.00x)	277 (0.96x)	172 (62.0%)	25.97 (2.00x)	274 (0.95x)	171 (62.4%)	24.92 (2.00x)	274 (0.96x)	169 (61.7%)
	page size: 128	13.11 (1.00x)	265 (1.00x)	134 (50.8%)	12.98 (1.00x)	260 (1.00x)	132 (50.8%)	12.46 (1.00x)	263 (1.00x)	131 (50.0%)
	page size: 256	6.56 (0.50x)	267 (0.99x)	114 (42.7%)	6.49 (0.50x)	266 (0.98x)	114 (42.9%)	6.23 (0.50x)	271 (0.97x)	117 (43.2%)
	page size: 512	3.28 (0.25x)	286 (0.93x)	101 (35.3%)	3.25 (0.25x)	291 (0.89x)	100 (34.3%)	3.11 (0.25x)	293 (0.90x)	101 (34.5%)
Learned Index	2nd stage models: 10k	0.15 (0.01x)	98 (2.70x)	31 (31.6%)	0.15 (0.01x)	222 (1.17x)	29 (13.1%)	0.15 (0.01x)	178 (1.47x)	26 (14.6%)
	2nd stage models: 50k	0.76 (0.06x)	85 (3.11x)	39 (45.9%)	0.76 (0.06x)	162 (1.60x)	36 (22.2%)	0.76 (0.06x)	162 (1.62x)	35 (21.6%)
	2nd stage models: 100k	1.53 (0.12x)	82 (3.21x)	41 (50.2%)	1.53 (0.12x)	144 (1.81x)	39 (26.9%)	1.53 (0.12x)	152 (1.73x)	36 (23.7%)
	2nd stage models: 200k	3.05 (0.23x)	86 (3.08x)	50 (58.1%)	3.05 (0.24x)	126 (2.07x)	41 (32.5%)	3.05 (0.24x)	146 (1.79x)	40 (27.6%)

Figure 2: *The Case for Learned Index Structures by Kraska et al*

Learned Index Structures Conclusion

Advantages

- can take advantage of real world data patterns(ML) $\rightarrow$ allows for high optimization
- lower engineering costs

Disadvantages

- initial work on B-Trees and alike is lower since they do not require additional training

Software 2.0

Intro

- Classical Software Dev major downsides like difficult optimization of code and human error
- S2.0 doesn't base on declarative programming and tries to learn the desired functionality with a approximate base net
- enabled by development of Neural Networks in last 20 years allowing >100 layers deep networks
- program space is restricted for future training (backpropagation, gradien descent)
- many real world problems easier to detect desirable behaviour than to write a specific program

Advantages

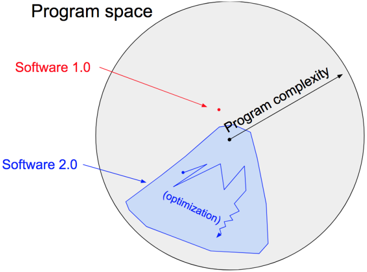
Higher Portability

- Smaller operation set
- Matrix Multiplication and thresholding at zero required
- small instruction set of chips with pretrained nets allows for

Better Performance

- allows for better performance and correctness predictions because closer implementation in hardware less core primitives are needed
- modules can be introduced to a single module reducing communication overhead by sacrificing clarity of separation, which is due to the human unlike nature of so2 sacrificed beforehand either way
- well trained neural nets outperform code implementation

Advantages



Better Runtime Predictability

- requires same amount of memory each iteration $\rightarrow$ low probability of infinite loops or locks
- speed well adjustable $\rightarrow$ speed can easily improved by reducing performance

Disadvantages

Unintuitivity

- can be treated as different new paradigm $\rightarrow$ requires rethinking of development style
- even though the network may work well, for humans difficult to understand
- developing in so2 is unintuitive and not well developed
- requires manually curating, maintaining, cleaning and labelling of datasets
- may not be applicable easily to all problems

Disadvantages

Nonrecognizable errors

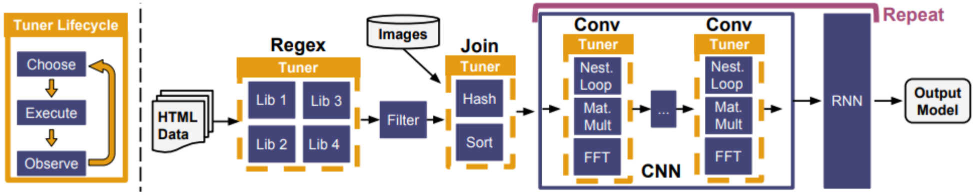
- errors may occur unpredictable
- can silently fail due to changed biases (hard to track since a large amount of them are being trained)

Lack of Tools

- no tools exist that support the development process like IDE, highlighting and alike for classical software

Example

Cuttlefish



Conclusion

Optimization

- Allows for greater optimization of complex problems
- e.g. Cuttlefish achieves 7.5x speedup to other query optimizers *Cuttlefish: A Lightweight Primitive for Adaptive Query Processing*

Development difficult

- No Tools and unexperienced developers

Thank you for your attention.

Do you have any questions? Ideas?
Be free to ask them.