DeepBM: A Deep Learning-based Dynamic Page Replacement Policy

Author: Xinyun Chen

Intro: DeepBM

- DeepBM learns from the past execution and dynamically adapts to the workload.

Existing Heuristic-Based Algo

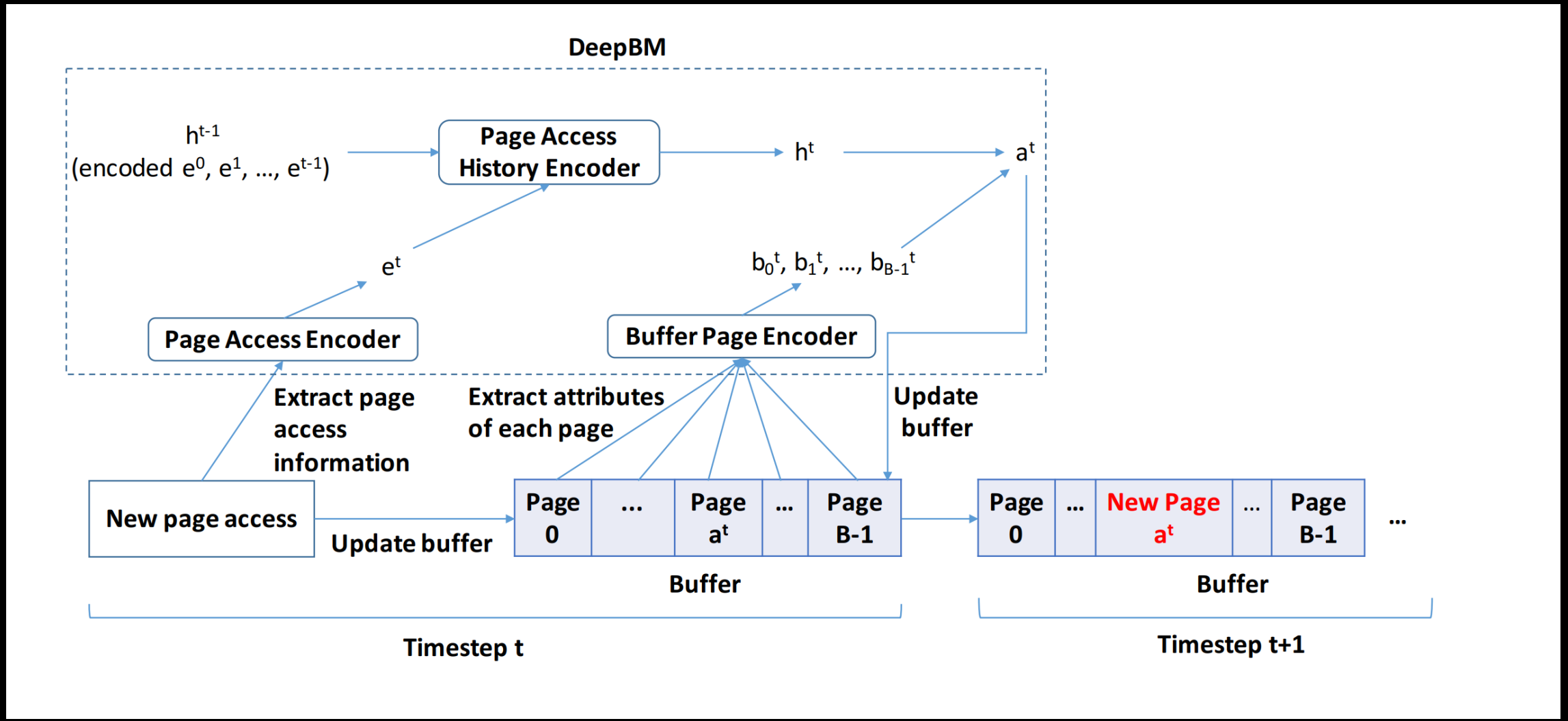
- Existing heuristic-based algorithms, which use a generic heuristic throughout the execution.

- These heuristic-based algorithms do not take the workload structure into consideration, thus forego the potential benefit of workload-specific optimization.

- A buffer with a maximal capacity of B pages.
- At each timestep t, a new page access is required, and the database system tries to find the page.
- If the page is missed and the buffer is full, the page eviction policy needs to predict an eviction action a $t \in \{0,1,\ldots,B-1\}$

DeepBM Framework

- **Page access encoder.** This component embeds the attributes of a new page access into a vector.
- **Buffer page encoder.** This component embeds the attributes of each page in the buffer into a vector.
- **Page access history encoder.** This component is a LongShort Term Memory neural network (LSTM), which encodes the entire trace of page access records into a vector.


DeepBM
h^{t-1}
(encoded e^0, e^1, ..., e^{t-1})
Page Access History Encoder
h^t
a^t
e^t
b_0^t, b_1^t, ..., b_{B-1}^t
Page Access Encoder
Buffer Page Encoder
Extract page access information
Extract attributes of each page
Update buffer
New page access
Update buffer
Page 0
...
Page a^t
...
Page B-1
Buffer
Page 0
...
New Page a^t
...
Page B-1
...
Buffer
Timestep t
Timestep t+1

Page access encoder.

- We can extract important attributes of each page access as the input features to our model. Table 1 illustrates the attributes we use in our evaluation, and more attributes could be included if needed.

No	Type	Name	Description
0	uint64	ts	time elapsed since the first buffer access (in ms)
1	uint32	rel_id	Relation ID
2	bool	is_local_temp	If this is a temporary relation in the current session
3	uint32	fork_num	(uint32)-1 - INVALID; 0 - MAIN; 1 - FSM; 2 - VISIBILITYMAP; 3- INIT
4	uint32	blk_num	disk block (page) number in a file
5	uint32	mode	0 - NORMAL (normal read) 1 - ZERO_AND_LOCK (dont read from disk; only lock the page) 2 - ZERO_AND_CLEANUP_LOCK (similar to 1, but lock the page in cleanup mode) 3 - ZERO_ON_ERROR (read page from disk, but return zeroed page on error) 4 - NORMAL_NO_LOG (the same as NORMAL here)
6	int32	strategy	-1 - DEFAULT 0 - NORMAL (random access) 1 - BULKREAD (large read-only scan) 2 - BULKWRITE (large multi-block write) 3 - VACUUM (vacuum)
7	uint32	rel_am	index access method; 0 - heap; 403 - btree; 405 - hash; 783 - gist; 2742 - gin; 4000 - spgist; 3580 - brin
8	uint32	rel_file_node	identifier of physical storage file
9	bool	has_toast	if it has toast table (for large values $\geq$ 8KB)
10	bool	has_index	if this table has or has had any index
11	char	rel_persistence	“p” - regular table; “u” - unlogged permanent table; “t” - temporary table
12	char	rel_kind	“r” - ordinary table “i” - secondary index “S” - sequence object “t” - for out-of-line values (toast table) “v” - view “m” - materialized view “c” - composite type “f” - foreign table “p” - partitioned table “I” - partitioned index
13	int16	rel_natts	number of user attributes
14	uint32	rel_frozen_xid	all xids that is smaller than this number are frozen
15	uint32	rel_min_mxid	all multixacts are greater than or equal to this number
16	bool	hit	whether the current page access is hit in the buffer or not. This attribute is obtained by running the page replacement policy used by PostgreSQL.

Table 1: Extracted attributes of each page access in our evaluation. These attributes are obtained by executing the workloads in PostgreSQL [39, 1].

Page access history encoder.

- To capture the page access pattern from the workload execution history, we employ a Long-Short Term Memory neural network (LSTM) as the page access history encoder.
- The main advantage of LSTMbased model is that it is more capable of learning long-term dependency and encoding the intricate characteristics in the long run.

Buffer page encoder.

- The approach of encoding pages in the buffer is similar to the encoding of new page accesses, except that the embedding matrices included in the buffer page encoder **use different weights.**
- Using the buffer page encoder, we first embed pages in the buffer into vectors.
- Afterwards, for each embedding vector, we concatenate it to another **weights** vector. **Note that pages with larger weights are accessed more recently.**

Evaluation

- We evaluate on two popular database workloads:
- The TPC-C benchmark
- Yahoo! Cloud System Benchmark (YCSB) benchmark.

Metrics

- **Hit rate.** The proportion of accessed pages that are already in the buffer.
- **Match rate.** This metric computes the proportion of eviction choices that are exactly the same as the optimal policy. This metric is mainly used to illustrate to what extent a page replacement algorithm aligns with the optimal policy.

Baseline Policies

- Least Recently Used (LRU).
- Most Recently Used (MRU).
- Page eviction policy used in PostgreSQL (PG).
- Optim. This is an oracle algorithm that computes the optimal decision assuming the knowledge of the subsequent page access information. This policy can be used as a reference to illustrate the optimality of different page replacement algorithms.

Hit rate	DeepBM (Ours)	PG	LRU	MRU	Optim
TPC-C	**67.11%**	66.02%	65.33%	4.83%	76.11%
YCSB	**48.70%**	47.17%	47.33%	3.12%	59.08%

Table 2: Hit rate of different page eviction policies on TPC-C and YCSB workloads.

Match rate	DeepBM (Ours)	LRU	MRU	Optim
TPC-C	**41.36%**	1.04%	0	100%
YCSB	**49.58%**	0.88%	0	100%

Table 3: Match rate of different page eviction policies on TPC-C and YCSB workloads.

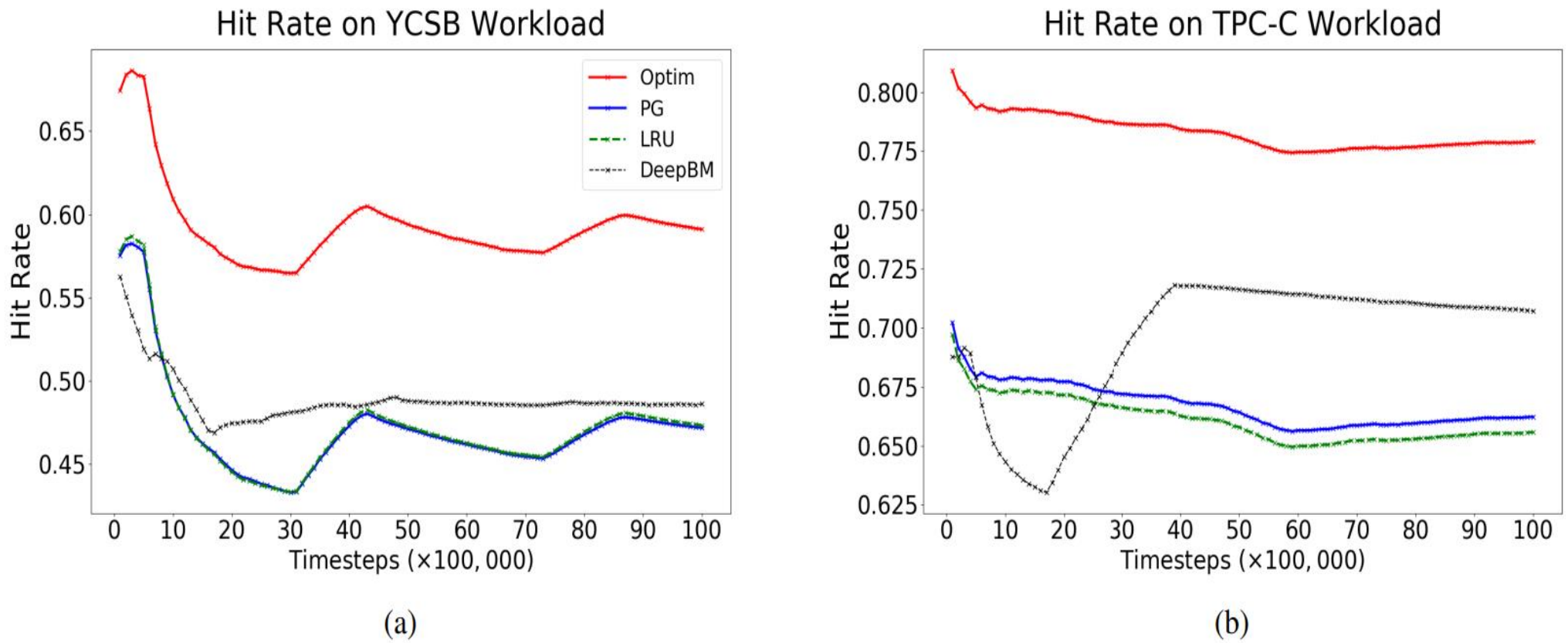
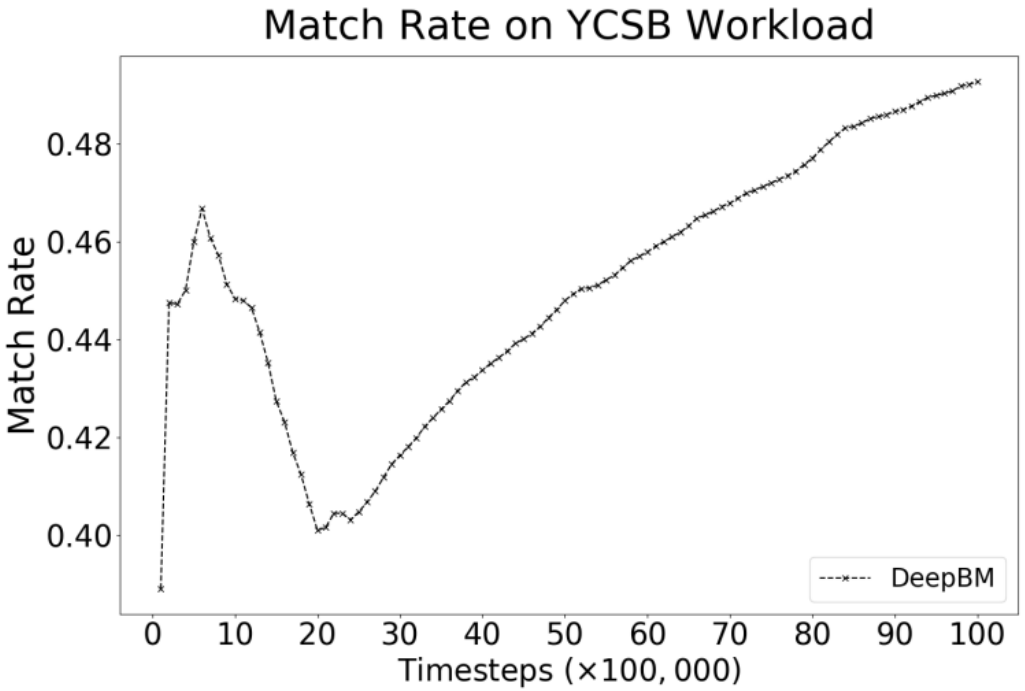
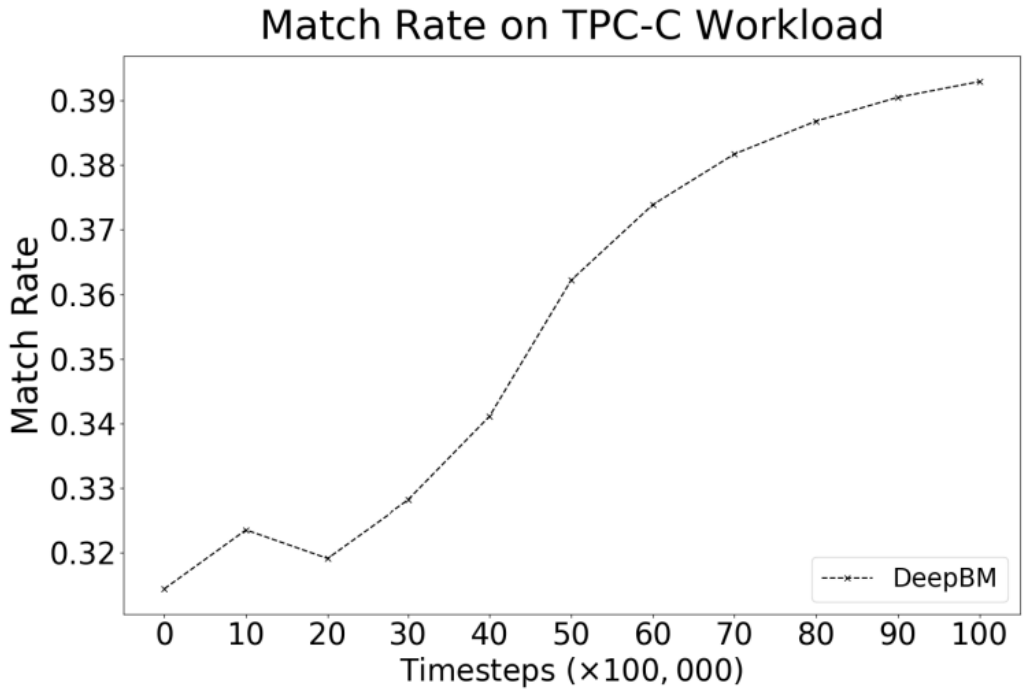


Figure 2: The hit rate of different page eviction policies throughout the workload execution. (a) YCSB workload. (b) TPC-C workload. Note that in (a), the curves of PG and LRU largely overlap with each other.



(a)



(b)

Figure 3: The match rate of DeepBM throughout the workload execution. (a) YCSB workload. (b) TPC-C workload.

Discussion:

- Is it reasonable to use match rate as a evaluation of DeepBM’s performance?
- There is no denying that higher hit rate is what we want. But DeepBM does not outrage other policies so much.
- What is the meaning of comparing to Optimal Algo?

Discussion:

- Can we combine DeepBM with existing buffer replacement algorithms?
- We can observe that the hit rate of DeepBM is pretty low at the beginning. Can we combine with LRU?