Adjacent LSTM-Based Page Scheduling for Hybrid DRAM/NVM Memory Systems

Manolis Katsaragakis
Microprocessors and Digital Systems Laboratory, National Technical University of Athens, Greece

Konstantinos Stavrakakis
Microprocessors and Digital Systems Laboratory, National Technical University of Athens, Greece

Dimosthenis Masouros
Microprocessors and Digital Systems Laboratory, National Technical University of Athens, Greece

Lazaros Papadopoulos
Microprocessors and Digital Systems Laboratory, National Technical University of Athens, Greece

Dimitrios Soudris
Microprocessors and Digital Systems Laboratory, National Technical University of Athens, Greece

Abstract
Recent advances in memory technologies have led to the rapid growth of hybrid systems that combine traditional DRAM and Non Volatile Memory (NVM) technologies, as the latter provide lower cost per byte, low leakage power and larger capacities than DRAM, while they can guarantee comparable access latency. Such kind of heterogeneous memory systems impose new challenges in terms of page placement and migration among the alternative technologies of the heterogeneous memory system. In this paper, we present a novel approach for efficient page placement on heterogeneous DRAM/NVM systems. We design an adjacent LSTM-based approach for page placement, which strongly relies on page accesses prediction, while sharing knowledge among pages with behavioral similarity. The proposed approach leads up to 65.5% optimized performance compared to existing approaches, while achieving near-optimal results and saving 20.2% energy consumption on average. Moreover, we propose a new page replacement policy, namely clustered-LRU, achieving up to 8.1% optimized performance, compared to the default Least Recently Used (LRU) policy.

2012 ACM Subject Classification
Computer systems organization → Heterogeneous (hybrid) systems; Hardware → Emerging tools and methodologies

Keywords and phrases
Page Placement, Long Short-Term Memory, LSTM, Prediction, NVM, DRAM

Digital Object Identifier 10.4230/OASIcs.PARMA-DITAM.2023.7

Funding
This work has been partially funded by EU Horizon 2020 program under grant agreement No 101015922 AI@EDGE (https://aiatedge.eu/).

1 Introduction
Over the last years, the rapid growth of applications that utilize, process and handle large data sets, while following the in-memory processing paradigm and pursuing sustainability is increasing in high pace and has exploded the number of data generated [10]. Applications derived from alternative domains, such as Machine Learning (ML) deployed on Cloud/HPC systems for training and inference [2], Online Analytical Processing (OLAP) applications and big-data analytics [9], significantly increase the demand for efficient storage and processing of data generated. This leads to I/O, memory bottlenecks and high pressure in the main memory of existing computing paradigms and performance degradation [11].
Such kind of applications are traditionally deployed on HPC and (pre-)exascale systems that integrate DRAM. However, despite their fast access latency, existing DRAM technologies face significant scalability issues [19], while leading to increased energy requirements, due to high leakage and refresh power, in order to maintain the data active inside the memory technology [17]. Therefore, performing more complex simulations and analytics by integrating more DRAM modules on each computing node, is neither a scalable nor a sustainable solution. To overcome these limitations, recent computing paradigms adopt Non-Volatile Memory (NVM), such as Spin-Transfer Torque RAM (STT-RAM), Phase Change Memory (PCM), Resistive-RAM (ReRAM) technologies and other. These technologies are inferior to existing DRAMs in terms of performance, however they achieve near-DRAM access latency, while they consume less energy, provide lower cost per GB, as well as persistence and higher scalability [10]. Modern commercial state-of-the-art platforms integrate Intel Optane DC Persistent Memory (DCPM) in the same memory layer with DRAM, providing a hybrid memory system, which consists of a large, energy efficient, but high latency (and/or low bandwidth) NVM and a fast DRAM of limited capacity, increased energy requirements and higher cost per byte.

Aiming to identify trade-offs between performance and energy consumption over hybrid DRAM/NVM systems, the clustering of pages is a promising solution [8]. Traditional page scheduling approaches tend to store frequently accessed (hot) data on the fast DRAM, in order to optimize performance and least accessed (cold) data on the slow, but energy efficient NVM, in order to provide energy gains [5]. Nevertheless, providing an efficient page clustering and placement scheme still remains non-trivial. Several prior state-of-the-art researches have investigated the problem of page placement [5, 20, 7]. However, existing approaches do not take advantage of spatio-temporal characteristics of the application’s pages, but their decision making is limited on the behavior of each individual page behavior, thus restricting the potential performance and energy gains that can be derived.

In order to fully exploit the hybrid memory systems characteristics, the Machine Learning paradigm is widely integrated in operating systems [8], as it can combine near-optimal results, low-overhead and real-time decision making. In this work, we introduce a page scheduling approach, which integrates a Critical Page Selector mechanism, a History-based Predictor and an Adjacent Long Short Term Memory (LSTM)-based Predictor. Page placement algorithms can strongly take advantage of spatio-temporal locality among pages [18]. Thus, we use LSTM networks to enable fast and accurate prediction of the memory accesses per page, respectively, in order to take advantage of temporal locality. Moreover, we enhance our predictor with knowledge of neighboring pages, aiming to take advantage of spatial locality and the behavioral correlation of neighboring pages [18]. The essential tuning knob of the system is the designation of the pages to be placed on a DRAM/NVM system, aiming to optimize performance and/or reduce energy consumption.

The novel contributions of this work are the following:

- **We propose a Critical Page Selector**, for efficiently identifying and clustering critical pages based on read and write access operations over the hybrid memory system.

- **We propose an Adjacent LSTM-based Predictor mechanism** for accurately predicting the number of page memory accesses. We integrate knowledge sharing among application pages with behavioral similarity based on k-nearest neighbors clustering and we evaluate our approach against state-of-the-art solutions, showing that our framework achieves up to 65.5% optimized performance and 20.2% less energy consumption on average.
We integrate a novel page eviction policy on main memory, namely *clustered-LRU*, which extends the default, widely used, Least Recently Used (LRU) policy, based on the K-means clustering algorithm.

The rest of this paper is organized as follows: Section 2 presents related work. In Section 3 we present the core principles of proposed page scheduler, while in Section 4 we provide an experimental evaluation and analysis of our scheduler and discuss key findings of our research. Finally, Section 5 concludes this paper.

## 2 Related Work

Several works have been conducted, aiming to target the problem of data placement over hybrid DRAM/NVM systems. Authors of [4] propose a runtime framework for adaptive granularity data placement for graph-based applications. Similarly, authors of [7] implement a set of libraries for object placement on heterogeneous memory systems. In [12] a memory management scheme for disaggregated memory systems is presented, aiming to support migration and hot/cold pages identification. Moreover, authors of [6] propose NUMA-aware hybrid allocation strategies for latency-sensitive and bandwidth-sensitive applications. In [13] authors design an object placement mechanism to reduce write traffic on NVM.

ML-based approaches have been proposed, in order to tackle the problem of object/page characterization and placement. Authors of [8] propose leveraging Recurrent Neural Networks (RNNs) for predicting memory access patterns, in order to improve memory prefetching. In a similar perspective, the authors of [20] propose a neural network based hardware prefetcher by predicting the access patterns of applications with linked data structures, compressed data formats and data dependent control flows. In [5], authors design a page scheduler for heterogeneous DRAM/NVM systems, based on deep learning techniques, namely *Kleio*.

Although research has illuminated the potential impact of efficient page scheduling for heterogeneous DRAM/NVM systems, no study to date, according to our knowledge, has evaluated a scheduling policy based on LSTMs, boosted by neighboring knowledge sharing. *Kleio* [5] is the most similar approach to ours, however there exist several fundamental differences:

- We propose and implement an adjacent LSTM-based predictor, enhanced with knowledge of pages with behavioral similarity in terms of memory access patterns, based on k-nearest neighbors clustering.
- We integrate a more sophisticated main memory replacement policy, namely *clustered-LRU*, aiming to cluster pages based on their access pattern.
- We propose a custom loss function evaluation for optimizing the training phase, in order to incorporate alternative weights according to each page criticality on the target application’s performance.

## 3 Proposed Methodology

The key idea of our proposed approach is to provide a run-time decision making scheduler for efficiently placing pages over heterogeneous DRAM/NVM memory systems, in order to optimize performance and energy consumption. More specifically, the main objective is to maximize the number of application’s memory requests that are served from DRAM, thus to maximize performance, while reducing the energy consumption by placing data on low-power NVM when they are not frequently processed. Our proposed scheduler consists of three major components: (i) the *Critical Page Selector*, (ii) the *Adjacent LSTM-based Predictor*
and (iii) the History-based Predictor. Critical Page Selector is responsible for efficiently identifying the critical pages that have significant impact on the overall performance, based on past memory accesses. The Adjacent LSTM-based Predictor is responsible for providing memory accesses predictions based on LSTM networks, aiming to co-exist and optimize History-based Predictor, which provides memory accesses predictions, however its effectiveness in providing applications with fast (i.e., in-DRAM) data accesses inherently depends on the application data access behavior [5]. Typical page placement approaches mostly rely on history-based approaches, resulting to sub-optimal placement decisions. Our proposed scheduler is illustrated in Fig. 1. As input to our framework, we provide the target application with the corresponding workload. The application consists of memory pages. Page scheduling occurs on every scheduling epoch. As scheduling epoch, we define the interval between two consecutive placement decisions.

### 3.1 Critical Page Selector

The core functionality of the Critical Page Selector component is the identification of performance-critical pages of the input workload, based on their actual memory accesses over the past N scheduling epochs. Ideally, if computational resources and real-time requirements were not a bottleneck, the maximum prediction accuracy could be achieved by deploying a single LSTM for each application page, respectively. However, this would lead to increased demands in terms of required resources, while the decision latency would skyrocket. Thus, the Critical Page Selector is designed to classify the pages on two major categories: a) Critical, which require sophisticated decision making through LSTM-based solution and a miss-placement can be significant for the application’s performance and b) Non-Critical, which represent those pages, whose influence is not that significant for the application’s performance, thus they are tolerant to miss-placements and a typical lightweight History-based Predictor can handle this decision.

The classification criteria of pages to Critical and Non-Critical is strongly related to the number of past read and write accesses of each page, respectively. Due to the inherent asymmetry of read and write access latency on NVMs and their limited write endurance, the impact of write operations on the application’s performance and energy consumption is more dominant. Thus, we consider weighted accesses of page \( p \) for the scheduling epoch \( i \), as shown in Eq. 1. Constants \( \alpha \) and \( \beta \) are set to 0.75 and 0.25, respectively.

\[
\text{Accesses}_i(p) = \alpha \times \text{writes}_i(p) + \beta \times \text{reads}_i(p)
\]  

(1)
For the scheduling epoch $i$, we define a binary function $MP_i(p)$ (Eq. 2), based on the placement decision, which indicates whether the page $p$ was misplaced on scheduling epoch $i$. A page is defined as misplaced by the Critical Page Selector, when the page was not placed in the optimal memory technology, due to page hotness inaccurate prediction in the past. A page is characterized as hot when the number of past accesses exceeds a user-defined threshold $h$ and thus is expected to maintain this behavior in the next scheduling epoch.

$$MP_i(p) = \begin{cases} 
1 & \text{if } p \text{ was misplaced on epoch } i \\
0 & \text{if } p \text{ was properly placed on epoch } i
\end{cases} \quad (2)$$

For every individual page $p$ of the input application and for $N$ past scheduling epochs, we define profit function (Eq. 3), which is the essential factor for the final classification of the misplaced pages and indicates impact of the corresponding misplacement, measured on the summary of accesses of $N$ previous epochs.

$$\text{Profit}(p) = \sum_{i=0}^{N} \text{Accesses}_i(p) \ast MP_i(p) \quad (3)$$

In order to classify the pages, we define $\lambda$, a user-defined constant. If the profit function of page $p$ is higher than $\lambda$, then the page $p$ is considered as Critical, therefore it is fed as input to the LSTM-based Predictor\(^{(3a)}\), otherwise it is considered as Non-Critical and given as input to History-based Predictor\(^{(3b)}\).

### 3.2 Adjacent LSTM-based Predictor

This component is responsible for accurately predicting the number of Critical page accesses (as derived from Critical Page Selector) for the next scheduling epoch. We utilize a single LSTM for every individual page, consisting of two layers with 256 neurons per layer, followed by one Dense layer. Furthermore, we deploy k-nearest neighbors algorithm, in order to encapsulate the spatio-temporal locality of our application without adding significant computation overhead\[18\]. We cluster the $M$ application pages on groups of $k$ pages, according to behavioral similarity on previous scheduling epochs. Constant $k$ is set to 8. The input of the LSTM for page $p$ on scheduling epoch $i$ is a 2D-feature vector of size $k \times l$, where $l$ denotes the history length, which is set to 20 through experimentation.

The output of the per-page LSTM is the predicted number of main memory accesses for the next scheduling epoch. The input sequence is normalized between 0 and 1 using $\text{softmax()}$ function, to stabilize the gradient descent step and achieve faster convergence. In contrast to existing approaches\,[8, 5\], the only information required from the model are the relative accesses of the pages, as the absolute prediction of accesses would lead to extra computational overhead and potential fault predictions. Significant role for the efficiency of our proposed scheduler has the selection of the loss function. In contrast to existing approaches, such as\,[5\] which utilizes the mean squared error loss function, we design and implement a custom weighted loss function, aiming to weight each prediction’s contribution to the cost function. The custom loss function for page $p$ on scheduling epoch $i$ is defined in 4.

$$\text{Loss}_i(p) = \text{real}_i(p) \ast \text{log(predicted}_i(p)) \ast w_i(p) \quad (4)$$

The variables $\text{real}_i(p)$ and $\text{predicted}_i(p)$ denote the normalized real and predicted value of page $p$ on epoch $i$, respectively, while $w_i(p)$ indicates the corresponding weight. The neural
network aims to minimize the custom loss value between the predicted and actual values, using the Adam optimizer and a learning rate equal to 0.01. The training stops if the loss for the validation dataset is not reduced for $s$ consecutive training epochs. We set $s$ equal to 30 epochs.

Regarding the training phase of our proposed scheduler, the training time of an LSTM for every individual page can skyrocket due to huge overhead. Thus, inspired by [8], we detect the most dominant pages over the $c$ most performance-critical pages. Variable $c$ is a user-defined threshold derived through extensive experiments. 75% of the dataset is utilized for training and 25% for validation.

### 3.3 History-based Predictor

The Non-Critical pages, as derived by the Critical Page Selector are fed as input to the History-based Predictor, which strongly relies on existing approaches, such as [16]. The core functionality of this component is the selection of page accesses based on past traffic patterns and strongly relies on the assumption that both cold and hot pages will maintain their pattern based on the latest scheduling epochs, e.g. a highly-accessed page will remain highly-accessed. This kind of scheduler is vulnerable to mis-predictions. However, due to the Critical Page Selector strategy, such faults will not significantly affect the performance of a page misplacement on DRAM/NVM system.

### 3.4 Page Grouping, Placement and Migration

The predictions of the LSTM-based Predictor and the History-based Predictor are finally accumulated and sorted in ascending order based on the number of the predicted memory accesses. Our grouping policy aims to split the pages into two distinct groups, i.e. hot pages and cold pages, based on a user-defined threshold $t\epsilon[0,1]$. The $t\%$ is selected to be placed on the DRAM, while the $(1-t)\%$ is placed on the NVM. Through this classification the hot pages can benefit from the fast DRAM, while cold pages that are not accessed frequently take advantage of the low-power storage of persistent memory.

However, the state of each individual page can be modified in consecutive scheduling epochs, thus creating the necessity for page reorganization between the DRAM and NVM memory technologies. Based on the prediction of the upcoming epoch, each page is either maintained on the same memory technology or migrated to the other technology. We target engines that allow seamless page migration, which is overlapped with the computation, thus overhead of the migration between DRAM and NVM is negligible [5, 14].

### 3.5 Clustered-LRU Replacement Policy

Aiming to further optimize the performance of our proposed framework, we integrate a page eviction policy from the main memory, namely clustered-LRU. Our proposed replacement policy extends the LRU replacement policy, which is the state-of-the-art replacement policy on main memories. Inspired by the clustering approach implemented in [8], we integrate a k-means clustering approach, in order to cluster the pages. The number of clusters is chosen through extensive evaluation based on inspecting the distribution of addresses of the whole address space. Each page is clustered with respect to the number of read and write accesses on the main memory (DRAM or NVM). Through this clustering, the sparsity of the address space of an application is avoided. Every individual page is assigned with a unique cluster ID, which represents the group of pages that the corresponding page belongs to.
During every scheduling epoch, we maintain information about the most active address clusters, i.e., the clusters that have the highest number of page read and write accesses on the main memory. Clusters are then sorted in descending order based on the number of accesses, thus pages that belong to the most active clusters will be prioritized to retain their position in DRAM during the eviction process. Through this policy, we take advantage of the spatial and temporal locality of the data, based on the observation that a cluster that is highly active during a scheduling epoch will probably remain active within the next scheduling epoch as well. Moreover, by maintaining the most highly-accessed clusters on the main memory, the number of page accesses on the slow disk are avoided, thus performance degradation is avoided.

4 Experimental Setup and Evaluation

4.1 Experimental Setup

Our experiments were conducted on a single node with a 2x20 core Intel Xeon Gold 5218R CPU @2.10GHz with 4x32GB DDR4 DIMMs. In order to derive the required tracing data, we utilize the Intel Pintool [15], which is the standard tool for dynamic binary instrumentation framework for x86-64 instruction-set architecture that enables the creation of dynamic program analysis custom tools. We consider 4 KB virtual page ID, that corresponds to the virtual memory address and we group memory accesses into scheduling epochs interval similar to [5].

Similar to existing approaches [5] and in order to evaluate the impact of our proposed scheduler, we design an in-house lightweight heterogeneous DRAM/NVM memory system simulator. We simulate a memory system that consists of a fast memory (i.e., DRAM) and one with high access latency (i.e., NVM). The specific memory characteristics regarding the access latency per memory technologies are derived by [5]. We integrate the Adjacent LSTM-based Scheduler on our simulator, which is responsible for placing, migrating and monitoring the behavior of pages. Last but not least, for our evaluation we utilize a set of benchmarks from alternative application domains, derived by Rodinia 3.1 [3] and PARSEC [1] benchmark suites. Table 1 summarizes the benchmarks used for our experimental evaluation and technical characteristics regarding the page read and write page accesses.

4.2 Experimental Evaluation

Comparative Performance Analysis. We evaluate our Adjacent LSTM-based page scheduler against existing scheduling algorithms. Initially, in order to quantify the performance of our approach against the optimal solution, we augment the comparison by adding an Oracle prediction mechanism, which possesses all the page information a priori and provides the optimal solution based on exhaustive search method. Moreover, we implement from scratch
Adjacent LSTM-Based Page Scheduling for Hybrid DRAM/NVM Memory Systems

**Figure 2** Normalized performance comparison of alternative placement scheduling policies. Results are normalized based on Oracle prediction.

**Figure 3** Root Mean Square Error (RMSE) comparison of alternative placement policies for all benchmarks. Y-axis is in log scale.

and compare our proposed scheduler against a history-based approach [16], which is a straightforward solution for page placement on heterogeneous memory systems, based on the pattern of previous history epochs. Finally, we implement from scratch and compare against a state-of-the-art page scheduler which utilizes machine intelligence, namely Kleio [5]. As evaluation metric we utilize the DRAM hit-rate metric, indicating the percentage of requests served from DRAM (higher DRAM hit-rate corresponds to higher performance).

Figure 2 illustrates the comparative analysis of the alternative scheduling policies. The X-axis denotes the corresponding benchmark ID, as derived from Table 1, while Y-axis represents the DRAM hit-rate. We observe that our approach outperforms both history-based scheduler and Kleio in all experiments, by achieving up to 65.5% and 32.7% higher DRAM hit-rate, respectively. As expected, history-based predictor behaves worse compared to other approaches, as it does not have any intelligence for placement. Regarding the comparison with Kleio, our Adjacent LSTM-based scheduler performs better due to the fact that we achieve optimized placement and migration decisions. Moreover, our proposed scheduler converges to the optimal solution (RD1, RD3, RD4, RD5 derived by the Oracle prediction, by achieving down to 3.9% miss from the optimal solution(RD3). However, in a subset of benchmarks (PS1, PS2, PS3, RD2), there is still space for further optimization in order to approximate near-optimal results. This is due to the fact that automatic memory pattern discovery is not straightforward and cannot be effectively determined by Recurrent Neural Networks (RNNs), as it has already been indicated in [8].
Comparative Accuracy Analysis. The functionality of our scheduler is directly linked to the efficiency of our proposed *Adjacent LSTM-based predictor* for page accesses prediction. Thus, we evaluate the Root Mean Square Error (RMSE) of our predictor and compare with the prediction mechanisms of the history-based approach and *Kleio*, respectively. Figure 3 illustrates the RMSE comparison of alternative approaches. The X-axis denotes the corresponding benchmark ID and the Y-axis reports the RMSE in logarithmic scale. As expected, the history-based predictor does not provide high accuracy, as it only relies on past epochs. The predictor integrated on our scheduler outperforms both the history-based scheduler and *Kleio*. More specifically, our proposed scheduler achieves 74.8% less RMSE compared to history-based approach on average and 54.9% less RMSE compared to *Kleio* on average. The former is due to the fact that history-based scheduler does not provide significant machine intelligence, thus there exists high randomness, while the latter is due to the fact that our *Adjacent LSTM-based predictor* takes advantage of the knowledge of pages with behavioral similarity in terms of access pattern, in contrast to *Kleio*, which does not exchange any knowledge among pages.

Comparative Energy Consumption Analysis. Efficient page scheduling policies for heterogeneous memory systems should be able to provide sustainable behavior. Thus, we perform experiments for evaluating the overall energy consumption of both DRAM and
NVM. We utilize the energy models presented in [21]. Figure 4 depicts the normalized energy consumption of the alternative schedulers over the benchmark workloads. **Adjacent LSTM-based** approach performs significantly better in the energy consumed. More specifically, our proposed scheduler achieves up to 38.7% and 24.3% less energy consumption compared to history-based and *Kleio* scheduler, respectively.

The key insight for the reduced energy consumption of the **Adjacent LSTM-based scheduler** is the efficient page migration between DRAM and NVM. Through the efficient prediction mechanism, pages remain in the DRAM as long as it is predicted to have increased number of accesses, i.e. characterized as hot, while for the rest of the predictions are migrated and stored in the low-power NVM. Once they become hot again they migrate again to DRAM, for the period of the increased number of accesses, thus combining increased performance and low energy consumption.

Aiming to further exploit the behavior of energy consumption, we first observe that there is a correlation between performance and energy. For instance, the RD2 benchmark provides increased energy consumption, while it provides comparatively low DRAM Hit-rate. This is due to the fact that both DRAM and NVM are utilized for higher period of time. Moreover, we observe that there exist benchmarks which even though they provide low DRAM Hit-rate they also provide reduced energy consumption (e.g. PS3). This is due to the fact that the PS3 benchmark is read-heavy. Through the memory traces collected, the write/read ratio is measured to 0.19. Thus energy-expensive write operations are avoided. Similar observations can be derived for the other benchmarks.

**Replacement Policy Evaluation.** Last but not least, we evaluate the impact of the underlying page replacement policy on main memory over our LSTM-based framework. We compare our proposed replacement policy, namely clustered-LRU, against the default replacement LRU policy. Figure 5 indicates the DRAM Hit-rate optimization derived by our proposed replacement policy. We observe that the DRAM hit-rate is optimized up to 8.1% compared to the default LRU policy. This is due to the fact that highly-accessed pages are mainly accessed directly from the main memory and not from the low-latency disk. Moreover, through our proposed approach, the replacement of clusters of pages instead of single pages reduces the number of replacement requests both on DRAM and NVM.

### 4.3 Discussion

In this subsection, we indicate key findings derived from our experimental analysis and evaluation. **Machine intelligence methods can be effectively utilized and combined for page placement.** Throughout our observations on subsection 4.2 we observe that the combination of alternative machine learning models, such as LSTMs and k-nearest neighbors in our case, can boost the performance(Fig. 2), as the most crucial factor for optimized performance is the accuracy of the corresponding predictor. Thus, the exploration and integration of alternative machine learning approaches is of major research interest, as the page placement can provide further optimizations, especially in application workloads that have still performance gap compared to the Oracle scheduler.

Furthermore, **poor replacement policy cannot allow an heterogeneous memory system to fully benefit from an efficient page scheduler approach.** As indicated in Figure 5 widely utilized approaches, such as the LRU policy have still significant space for optimization. Therefore, the investigation of alternative replacement policies can further optimize the performance of existing placement algorithms.
Efficient data placement and migration can exploit the advantages of the sustainable nature of recent NVM technologies. As indicated in 4 an efficient page placement scheduler provides energy optimizations, despite the fact that the main objective of our Adjacent LSTM-based scheduler is performance-oriented. Further energy optimizations can be potentially achieved through the design of page placement policies aiming to optimize energy/power consumption.

5 Conclusion

In this paper, we present a novel approach for efficient page placement on heterogeneous DRAM/NVM systems. We design an Adjacent LSTM-based approach for placing pages based on page accesses prediction through knowledge of pages with behavioral similarity. Our proposed approach leads up to 65.5% optimized performance compared to existing approaches, while achieving near-optimal results in some cases and saving 20.2% energy consumption on average. Moreover, we propose a new page replacement policy, namely clustered-LRU, achieving up to 8.1% optimized performance, compared to the default LRU policy.

References

Adjacent LSTM-Based Page Scheduling for Hybrid DRAM/NVM Memory Systems


