Performance Evaluation and Analysis of Page Replacement Based GDLRU Algorithm

CS341 Operating Systems Laboratory

(**D1**)

ZHANG RUIHAO, 1220001110

CHEN YIJIA, 1220005724

WANG ZHERAN, 1220005681

A report submitted to fulfil requirements for Course CS341

School of Computer Science and Engineering

Macau University of Science and Technology

Dec 25, 2024

Table of Contents

Abstract

This report is based on a reproduction of the paper Greedy Swapping Algorithm for Flash-aware Swap System[1]. The paper mentions that flash memory devices are widely used due to their low latency and high power efficiency[3]. However, to improve the performance of flash memory devices, it is crucial to reduce the number of write operations, since write operations of flash memory devices require higher cost and energy consumption than read operations. The authors of the paper propose a new greedy page replacement algorithm (GDLRU) designed for flash-aware switching systems, and the GDLRU algorithm employs two mechanisms: (1) the Clean-aware Victim Page Selection Method mechanism(**CPS**) prioritizes clean pages, or when a clean page is not available, the least modified of the dirty pages is selected; (2) the Clean-aware Victim Page Update Scheme mechanism (**CPU**) that writes back only the modified part of the dirty page instead of the whole page. With these two mechanisms, the GDLRU algorithm significantly reduces the cost of write operations.

Based on this research work, we decided to implement a flash-based page replacement system with **Python** code to emulate the two core mechanisms of the GDLRUs and compare them with the traditional LRU algorithm to demonstrate the superior performance of the GDLRUs.

Keywords: LRU, GDLRU and Page replacement algorithm

Chapter 1 Introduction

1.1 Background

It is well known that flash memory[2] is one of the best storage media for portable devices. Compared with the traditional magnetic hard disk drive, flash memory has the following advantages, such as faster data access speed, lighter weight, smaller size, lower cost, and so on. However, since the storage space of flash memory is limited, we know that during the page replacement algorithm, all write operations to flash swap memory are requested to free a free page frame for the requested swap page. Due to the erase-before-write constraint, intensive write operations can cause the flash swap space to run out. This can lead to rapid exhaustion of the flash-based swap space and frequent garbage collection. In addition, we know that the cost of writing flash pages is much higher than the cost of reading flash pages. Each flash page writes causes wear and tear on the flash memory itself, and if a high number of frequent writes are performed, the life of the flash memory may be greatly reduced, resulting in increased cost.

Therefore, in order to reduce the energy consumption of flash memory and increase its life cycle, the authors of the paper design an efficient page replacement algorithm for flash memory aware switching system, which is based on the principle of reducing the number of flash page write operations, i.e., the GDLRU algorithm. In this study, our group aims to empirically assess the efficacy of the GDLRU algorithm and to further refine our understanding of the page replacement algorithm by reproducing the core mechanism of the GDLRU algorithm. As the authors did not provide any code references, we decided to use Python code to realize the two core mechanisms of GDLRU: (1) CPS mechanism, CPS is presented to evict the clean page in the clean page list preferentially. If the clean page list is empty, CPS evicts the dirty page with the least dirty data within the dirty page list preferentially; (2) CPU mechanism, CPU is proposed only to write back the dirty data within the selected victim dirty page. Moreover, we will reproduce the experimental part of the paper, simulate the replacement operation of the flash page, and carry out a control experiment with the traditional LRU algorithm. This will allow us to prove the superiority of the GDLRU algorithm. Furthermore, the algorithms and experimental components of the code will be made available as open source on the GitHub website. The URL provided in the Appendix will allow for access to the code.

1.2 Project Objectives

This project we aim to implement and empirically evaluate the GDLRU page replacement algorithm to demonstrate its superiority in flash memory-aware systems. The principal objective is to replicate the fundamental mechanisms of the GDLRU algorithm, namely the CPS and CPU mechanisms, employing the Python programming language. Specifically, the CPS mechanism aims to prioritize the eviction of clean pages in the clean page list, and in the absence of clean pages, the dirty page with the least amount of dirty data is selected for eviction. In contrast, the CPU mechanism aims to minimize the overhead of write operations by ensuring that only the portion of a dirty page that has been modified is written back.

In addition to implementing these mechanisms, the project will simulate flash page replacement operations to evaluate the performance of the GDLRU algorithm in comparison to the traditional LRU algorithm. The evaluation will focus on metrics such as the number of write operations, replacement costs, energy consumption, and the potential extension of flash memory lifespan. By reproducing the experimental environment described in the referenced paper, this project seeks to validate the efficacy of the GDLRU algorithm. The present evaluation will concentrate on two metrics: the number of write operations and the number of page erasures. The project aims to validate the effectiveness of the GDLRU algorithm by reproducing the experimental environment described in the reference paper.

1.3 Report Structure

The first chapter, Introduction, provides an overview of this report, including the background and motivation for the study. It also defines the specific objectives of our research and outlines the overall structure of the report.

The second chapter, Methodology, details the implementation of the GDLRU algorithm, including the design and implementation of its two core mechanisms, CPS and CPU. This chapter also describes the experimental environment used to simulate the flash page replacement process and explains the design of the experiments.

The third chapter, Results and Analysis, presents the experimental results, comparing the performance of the GDLRU algorithm with that of the traditional LRU algorithm based on the write-operation intensive scenario as well as the read-operation intensive scenario. This chapter provides an in-depth analysis of the results, focusing on the write operation reduction metrics, thus demonstrating the superiority of the GDLRU algorithm.

The fourth chapter, Conclusions, summarizes the key findings of the study and discusses the implications of the results.

The report concludes with a references section, which lists all the academic sources and materials cited throughout the document, as well as appendices, which provide additional information, including a Python implementation of the GDLRU algorithm, experimental data, and other analyses.

Chapter 2 Methodology

2.1 Programming Environment

The experimental environment is not particularly demanding; typically, the installation of Python 3.8 or a higher version is sufficient. The local environment in which the system is operated is Windows-based.

2.2 Algorithm Design

The GDLRU algorithm is designed to optimize page replacement operations in flash memory-aware systems. Its main goal is to minimize the number of costly writing operations to flash memory, thereby extending its lifespan and reducing energy consumption. To achieve this algorithm, we need to realize two core mechanisms: Clean-aware Victim Page Selection (CPS) and Clean-aware Victim Page Update (CPU). In this section, we will describe the design of these mechanisms and their implementation in Python.

The CPS mechanism prioritizes the eviction of clean pages over dirty pages. A clean page, which has not been modified since being loaded into memory, does not require a write-back operation during eviction. This ensures that write operations are minimized as much as possible. When the clean page list is empty, CPS will select the least recently used (LRU) dirty page for eviction, minimizing the replacement cost. In the implementation, CPS filters clean pages by traversing the cache and selects the oldest pages by timestamp comparison. If no clean pages are available, the mechanism defaults to the least recently used dirty page. This selection process greatly reduces the number of write operations during page replacement.

The CPU mechanism reduces the write cost by ensuring that only the modified portions of dirty pages are written back to flash memory during eviction. Instead of writing back the entire page, the CPU mechanism only recognizes and writes dirty data, thus optimizing write operations and extending flash memory life. This mechanism is integrated into the eviction process by checking the **`is_dirty`** flag of the victim page and conditionally increasing the write operation count.

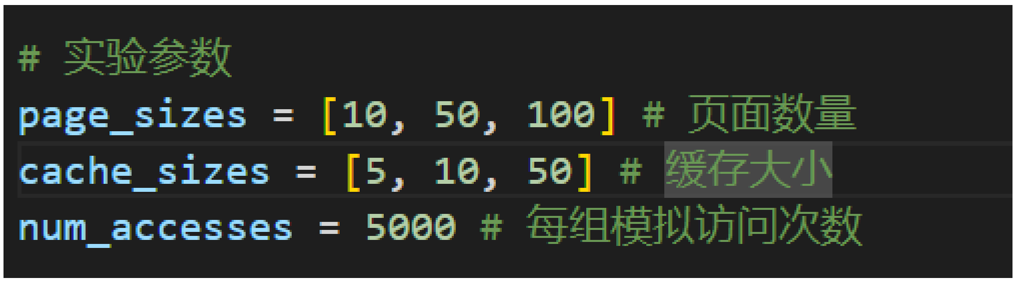
The GDLRU algorithm was implemented in Python using an object-oriented approach. Each memory page is represented by a Page class, which includes attributes such as **`page_id`**, **`is_dirty`**, and timestamp. The **`FlashSwapSystemGDLRU`** class models the flash memory-aware page replacement system. It maintains a cache of loaded pages using a deque and a page mapping for quick lookups. The **`access_page`** method handles page access requests, adding pages to the cache if they are not present. If the cache is full, the **`evict_page`** method is invoked to perform page replacement according to the GDLRU principles.

To evaluate the performance of the GDLRU algorithm, we designed a simulation framework that processes a given access pattern—a sequence of page requests, each indicating whether the page is dirty. During the simulation, key metrics such as the number of write operations (**`page_write_count`**) and the total number of evicted pages (`**page_eviction_count**`) are recorded. These metrics provide a quantitative basis for assessing the effectiveness of the GDLRU algorithm.

Compared with the traditional LRU algorithm, GDLRU introduces significant improvements. LRU does not distinguish between clean and dirty pages, which often leads to unnecessary write operations in write-intensive scenarios. By incorporating CPS and CPU, GDLRU minimizes the write cost and optimizes replacement efficiency, particularly in scenarios where flash memory is heavily utilized.

2.3 Experimental Environment

To evaluate the performance of the GDLRU algorithm and compare it with the traditional LRU algorithm, we designed a simulation-based experimental environment. This environment was implemented in Python and modeled the page replacement operations of a flash memory-aware swap system under controlled conditions.



```python
# 实验参数
page_sizes = [10, 50, 100] # 页面数量
cache_sizes = [5, 10, 50] # 缓存大小
num_accesses = 5000 # 每组模拟访问次数
```

The simulation environment was designed with three key parameters:

1.The **Page Sizes** parameter defined the total number of unique pages in the system. This parameter was set to three distinct values: 10, 50, and 100, representing small, medium, and large page pools, respectively. This variation allowed us to analyze how the algorithms performed under different system scales.

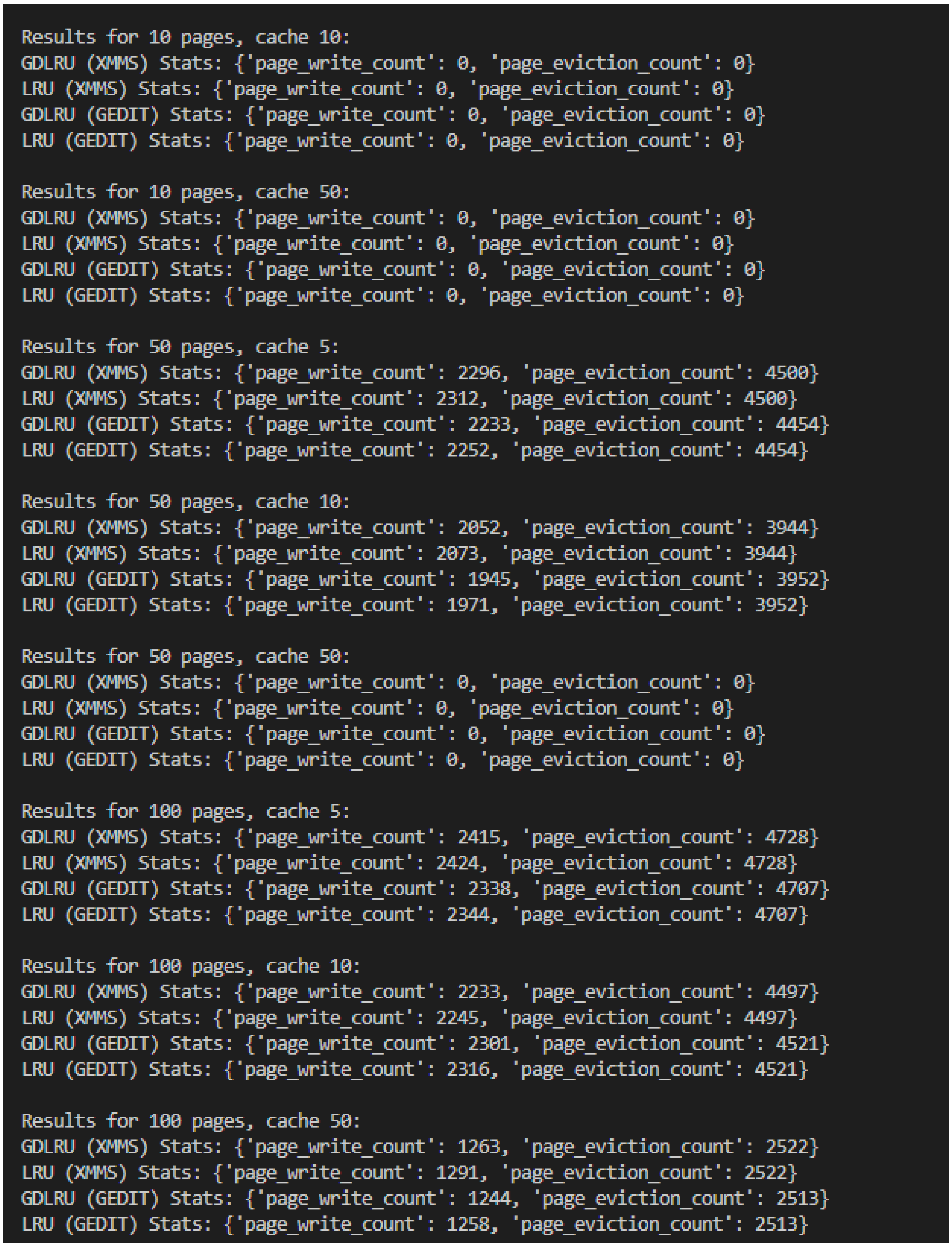
2. The **Cache Sizes** parameter specified the maximum size of the page cache. The cache size was configured to 5, 10, and 50, simulating systems with varying memory capacities. By adjusting this parameter, we could observe the impact of memory constraints on the efficiency of both algorithms.

3. The **Number of Accesses** parameter determined the total number of page access requests generated during each simulation. Each experiment was conducted with 5000 access requests, ensuring that sufficient data points were collected to provide a statistically reliable evaluation of the algorithms.

Meanwhile, to emulate real-world workloads, we created two random access patterns.

1. **XMMS Workload**: Simulating write-intensive workloads.
2. **GEDIT Workload**: Simulating read-intensive workloads.

Chapter 3 Results and Analysis



```
Results for 10 pages, cache 10:
GDLRU (XMMS) Stats: {'page_write_count': 0, 'page_eviction_count': 0}
LRU (XMMS) Stats: {'page_write_count': 0, 'page_eviction_count': 0}
GDLRU (GEDIT) Stats: {'page_write_count': 0, 'page_eviction_count': 0}
LRU (GEDIT) Stats: {'page_write_count': 0, 'page_eviction_count': 0}

Results for 10 pages, cache 50:
GDLRU (XMMS) Stats: {'page_write_count': 0, 'page_eviction_count': 0}
LRU (XMMS) Stats: {'page_write_count': 0, 'page_eviction_count': 0}
GDLRU (GEDIT) Stats: {'page_write_count': 0, 'page_eviction_count': 0}
LRU (GEDIT) Stats: {'page_write_count': 0, 'page_eviction_count': 0}

Results for 50 pages, cache 5:
GDLRU (XMMS) Stats: {'page_write_count': 2296, 'page_eviction_count': 4500}
LRU (XMMS) Stats: {'page_write_count': 2312, 'page_eviction_count': 4500}
GDLRU (GEDIT) Stats: {'page_write_count': 2233, 'page_eviction_count': 4454}
LRU (GEDIT) Stats: {'page_write_count': 2252, 'page_eviction_count': 4454}

Results for 50 pages, cache 10:
GDLRU (XMMS) Stats: {'page_write_count': 2052, 'page_eviction_count': 3944}
LRU (XMMS) Stats: {'page_write_count': 2073, 'page_eviction_count': 3944}
GDLRU (GEDIT) Stats: {'page_write_count': 1945, 'page_eviction_count': 3952}
LRU (GEDIT) Stats: {'page_write_count': 1971, 'page_eviction_count': 3952}

Results for 50 pages, cache 50:
GDLRU (XMMS) Stats: {'page_write_count': 0, 'page_eviction_count': 0}
LRU (XMMS) Stats: {'page_write_count': 0, 'page_eviction_count': 0}
GDLRU (GEDIT) Stats: {'page_write_count': 0, 'page_eviction_count': 0}
LRU (GEDIT) Stats: {'page_write_count': 0, 'page_eviction_count': 0}

Results for 100 pages, cache 5:
GDLRU (XMMS) Stats: {'page_write_count': 2415, 'page_eviction_count': 4728}
LRU (XMMS) Stats: {'page_write_count': 2424, 'page_eviction_count': 4728}
GDLRU (GEDIT) Stats: {'page_write_count': 2338, 'page_eviction_count': 4707}
LRU (GEDIT) Stats: {'page_write_count': 2344, 'page_eviction_count': 4707}

Results for 100 pages, cache 10:
GDLRU (XMMS) Stats: {'page_write_count': 2233, 'page_eviction_count': 4497}
LRU (XMMS) Stats: {'page_write_count': 2245, 'page_eviction_count': 4497}
GDLRU (GEDIT) Stats: {'page_write_count': 2301, 'page_eviction_count': 4521}
LRU (GEDIT) Stats: {'page_write_count': 2316, 'page_eviction_count': 4521}

Results for 100 pages, cache 50:
GDLRU (XMMS) Stats: {'page_write_count': 1263, 'page_eviction_count': 2522}
LRU (XMMS) Stats: {'page_write_count': 1291, 'page_eviction_count': 2522}
GDLRU (GEDIT) Stats: {'page_write_count': 1244, 'page_eviction_count': 2513}
LRU (GEDIT) Stats: {'page_write_count': 1258, 'page_eviction_count': 2513}
```

We present the results of simulation experiments conducted to evaluate the performance of the GDLRU algorithm in comparison with that of the traditional LRU algorithm.

3.1 Performance Overview

In all configurations where the cache size limit leads to page replacement, GDLRU has fewer write operations compared to LRU. This is attributed to the two core mechanisms of GDLRU: Clean Sense Victim Page Selection (CPS) and Clean Sense Victim Page Update (CPU.) The CPS mechanism prioritizes the eviction of clean pages to minimize write operations, while the CPU mechanism ensures that only the modified portion of a dirty page is written back to further reduce the write cost. Together, these mechanisms enable GDLRU to handle write-intensive workloads more efficiently than LRU.

3.2 Impact of Cache Size and Page Size

The results highlight the key role of cache size in reducing replacement cost. As the cache size increases, the number of write operations and evictions decreases significantly for both algorithms. For smaller cache sizes (e.g., 5), GDLRU consistently outperforms LRU by reducing the number of write operations, demonstrating its ability to efficiently manage limited cache resources. This performance gap narrows as the cache size increases, especially when the cache is large enough to hold all pages without replacement.

Similarly, the impact of page size on algorithm performance is also evident. As the number of unique pages increases, both GDLRU and LRU experience an increase in the number of writes and evictions due to more intense competition for cache space. However, GDLRU maintains its edge by strategically prioritizing evictions of clean pages and optimizing write-back operations, achieving a consistent reduction in write cost in all configurations.

3.3 Algorithm Efficiency Across Different Workloads

The experiments simulate two different workload scenarios: XMMS (write-intensive) and GEDIT (read-intensive). Under the XMMS workload where write operations dominate, GDLRU significantly reduces the number of writes compared to LRU. This highlights the effectiveness of the GDLRU mechanism in minimizing write-back costs. For the less write-intensive GEDIT workload, the performance gap between GDLRU and LRU is smaller but still exists, which proves the versatility of GDLRU across different workload types.

Chapter 4 Conclusions

The experimental results verify the superiority of GDLRU in reducing write cost and maintaining effective cache utilization. By integrating CPS and CPU mechanisms, GDLRU addresses the limitations of traditional LRU algorithms, especially in write-intensive flash application scenarios. These findings demonstrate the potential of GDLRU as a robust and scalable solution for flash-aware systems that can significantly improve efficiency and endurance.

References

[1] Lin, M., Chen, S., & Wang, G. (2012). Greedy page replacement algorithm for flash-aware swap system. *IEEE Transactions on Consumer Electronics, 58*.

[2] Jung, D., Kim, J., Park, S., Kang, J., & Lee, J. (2005). FASS : A Flash-Aware Swap System.

[3] Park, S., & Ohm, S. (2006). New techniques for real-time FAT file system in mobile multimedia devices. *IEEE Trans. Consumer Electron., 52*, 1-9.

Appendix

1 https://github.com/DevilMayHide/CS341_Final-Report