MITIGATING BÉLÁDY'S ANOMALY: OPTIMIZING MEMORY MANAGEMENT WITH HYBRID PAGE REPLACEMENT STRATEGIES

22BRS1155 RITIK SHEORAN , 22BAI1275 AYUSH BIST , 22BPS1207 LAKSHAY & 22BRS1151 HARSH PUNATAR

This study investigates the alleviation of Bélády's Anomaly through the deployment of Hybrid Page Replacement Strategies within memory management systems. Bélády's Anomaly, a phenomenon characterized by increased page faults with augmented memory allocation, presents a challenge in conventional page replacement algorithms. The proposed hybrid approach combines features of both FIFO (First-In-First-Out) and LRU (Least Recently Used) methodologies to optimize page replacement decisions. Through simulation-based assessments, the hybrid strategy's performance is compared against the individual FIFO and LRU approaches across diverse scenarios. The study aims to analyze their influence on the occurrence of Bélády's Anomaly and the overall efficiency of the system. Initial observations suggest the potential of the hybrid strategy in diminishing instances of the anomaly while concurrently augmenting the overall efficacy of memory management. The research underscores the significance of innovative hybrid methodologies in tackling Bélády's Anomaly, providing a promising avenue for enhancing memory management systems and mitigating the adverse effects of increased page faults associated with traditional page replacement algorithms

# 1.Introduction

The realm of computer science is continually challenged by the enigmatic Bélády's Anomaly, a conundrum that has long plagued memory management systems. As the need for efficient and seamless data handling escalates, the occurrence of this anomaly presents a critical impediment to the optimization of page replacement algorithms. Bélády's Anomaly manifests as an unexpected increase in page faults when additional memory is allocated, confounding traditional FIFO (First-In-First-Out) and LRU (Least Recently Used) page replacement strategies. To address this challenge, this research embarks on a compelling quest, delving into the innovative concept of a Hybrid Page Replacement Strategy.

At the heart of this investigation lies the endeavor to combine the stalwart attributes of FIFO and LRU methodologies. The hybrid strategy aims to intelligently merge the chronological sensitivity of FIFO with the adaptive responsiveness of LRU, potentially offering a groundbreaking solution to mitigate the unwelcome effects of Bélády's Anomaly. By harmonizing the strengths of these distinct approaches, the study seeks to usher in a new era of memory management, one that is not only adept at reducing anomaly instances but also at elevating the overall efficiency and performance of memory systems.

Through comprehensive simulations and meticulous analysis, this research aspires to not only unlock the secrets behind the anomaly's occurrence but also to lay the foundation for a novel and dynamic approach to page replacement algorithms. The resulting insights could herald a paradigm shift in memory management, paving the way for more robust and adaptive systems capable of navigating the complexities of modern data processing. This investigation thus stands as a beacon of innovation in the pursuit of conquering the elusive Bélády's Anomaly and revolutionizing memory management paradigms

# 2.LITERATURE REVIEW

***How Anomalous Is Bélády's Anomaly? By -Kirby McMaster ,Samuel Sambasivam , Nicole Anderson***

The study explored Bélády's anomaly under varying conditions using computer simulations with FIFO and Random Page algorithms, focusing on process size, reference string length, and allocated memory frames. Findings revealed that Bélády's anomaly occurred up to 58.6% of the time under FIFO and in 100% of cases for Random Page. With the highest anomaly bump counts at 869 for FIFO and 32152 for Random Page, it was noted that the average memory frame allocation was proportional to process size, approximately 75% for FIFO and just over 50% for Random Page at peak anomaly conditions. Across diverse process sizes and reference string lengths, the anomaly was so prevalent, especially under Random Page, that it was no longer seen as an exceptional occurrence but rather as a common phenomenon, signifying its frequent manifestation and emphasizing the susceptibility of Random Page to this anomaly

***And Page Replacement Algorithms: Anomaly Cases. By - Najeeb A. Al-Samarraie***

Extensive research in the domain of operating systems and memory management has continually delved into the intricate realm of page replacement algorithms and their impact on system performance. Central to this exploration is the pervasive issue of Bélády's Anomaly, a phenomenon where increased memory allocation paradoxically leads to heightened page faults, challenging the effectiveness of traditional page replacement strategies. Key algorithms such as First In First Out (FIFO), Optimal (OPT), and Least Recently Used (LRU) have been at the forefront of numerous studies examining their efficiencies and vulnerabilities. The distinct behaviors of these algorithms, such as FIFO's simplicity yet susceptibility to Bélády's Anomaly, OPT's theoretical optimality, and LRU's reliance on past usage patterns, have been scrutinized through empirical experiments simulating various reference strings and page frame configurations. These investigations highlight anomalies and unexpected outcomes, notably Bélády's Anomaly cases, where increasing the number of available frames results in augmented page faults—a scenario contradicting the notion that more available memory should decrease faults. The findings consistently showcase OPT as the algorithm exhibiting the lowest page fault rates across diverse scenarios, emphasizing its theoretical optimality, while offering insight into anomalies and system behavior. This comprehensive literature amalgamation underscores the multifaceted nature of page replacement algorithms, their distinct behaviors, and the persistent enigma of Bélády's Anomaly, steering the focus towards optimizing memory management and enhancing system efficiencies.

***Removing Bélády’s Anomaly from Caches with Prefetch Data. By- Elizabeth Varki***

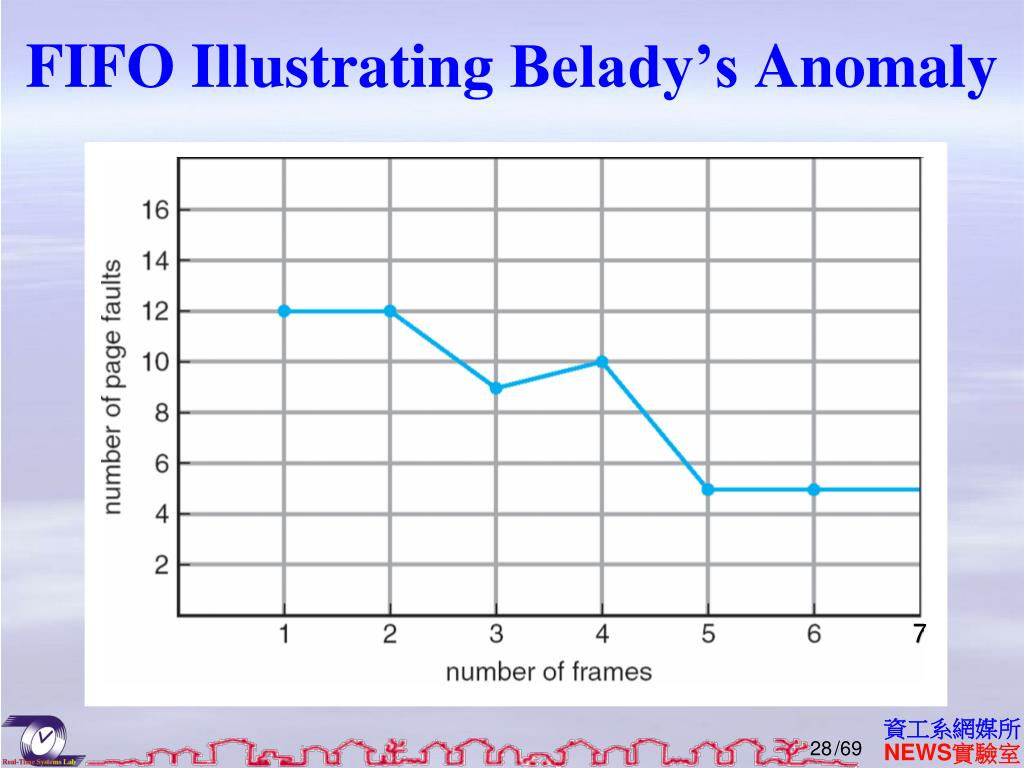
The study presented in the paper delves into the intricate relationship between prefetching mechanisms and cache replacement policies within file systems, highlighting the challenges in evaluating the resulting cache system due to its erratic performance. The research underscores that the introduction of prefetching often disrupts the stack property, a fundamental characteristic for efficient cache systems, consequently leading to instances of Bélády's anomaly. The entanglement of prefetch techniques with replacement policies complicates the isolation of the causes for this violation.

A significant accomplishment of this paper lies in its endeavor to elucidate the reasons behind the violation of the stack property by prefetching mechanisms and proposing potential remedies. It sheds light on the inherent difficulty in disentangling the effects of prefetch actions from replacement policy decisions, contributing to the unpredictability in system performance.

An insightful highlight of this research is the proposal for a prefetch technique that operates independently of the replacement policy, achieved through the utilization of a fixed stream degree. This novel approach has the potential to aid in the evaluation of replacement policies for prefetch caches, providing a valuable avenue to better understand and assess the performance of these systems.

Overall, the paper's contributions lie in both its explanations for the violation of the stack property induced by prefetching and its proposed remedies, offering a pathway towards disentangling prefetching from replacement policies for more structured and predictable cache system evaluations.

# 3.METHODOLOGIES



***Implementation of Hybrid Page Replacement***

**Page Fault Counter Setup:** The creation of a page fault counter involves developing a mechanism within the system that systematically tracks and counts every instance of a page fault occurring during system operation. This counter serves as a pivotal metric to measure system performance and progression. As each page fault happens, the counter increments, accumulating a numerical representation of system inefficiencies due to page replacements.

**FIFO Algorithm Integration:** The initial implementation of the FIFO algorithm serves as the primary page replacement strategy during the system's operation. Initially, the system employs FIFO to manage the first X page faults encountered. FIFO's simplicity in replacing the oldest page in memory ensures a straightforward start for the system’s page replacement logic.

**Transition Logic Development:** The transition logic involves the creation of a decision-making process guided by the page fault count. When the accumulated page fault count aligns with the predetermined threshold (X), a specific transition from using FIFO to LRU is triggered. This transition signifies a critical shift in the system's page replacement strategy from a straightforward FIFO approach to a more adaptive LRU scheme.

**Incorporating LRU:** Upon reaching the set threshold of page faults, the system integrates the LRU algorithm into its operation. LRU, which replaces the least recently used page when a new page needs allocation, takes over the page replacement process from FIFO. LRU's adaptability to the system's recent memory access history aims to enhance page replacement efficiency post-transition point.

***Simulation and Testing***

**Scenario Configuration:** Generating a diverse set of simulated scenarios is crucial for testing the system's robustness under varying conditions. These scenarios include different workloads, reference strings, and memory sizes, enabling comprehensive testing and analysis of the hybrid approach's adaptability to diverse operational conditions.

**Simulating Page Faults:** System simulations are executed to closely monitor and analyze the occurrences of page faults. The observed page fault counter plays a critical role, specifically to identify the moment when the predetermined threshold (X) of page faults is met. This observation is a pivotal event that triggers the transition from FIFO to LRU, marking a crucial shift in the system's page replacement strategy.

**Performance Comparison:**

**Before Transition Point:**

* **Page Fault Analysis:** Analyze the frequency and pattern of page faults while the system operates under the FIFO algorithm. Measure the number and distribution of page faults occurring within this phase.
* **System Efficiency:** Assess the overall system efficiency, considering factors such as response times, resource utilization, and any bottlenecks observed during this period.

**After Reaching Transition Point:**

* **Transition Point Verification:** Confirm the triggering of the transition from FIFO to LRU by verifying that the specified threshold of page faults (X) has been met.
* **Page Fault Behavior:** Examine the alteration in page fault patterns post-transition to LRU. Compare the number, sequence, and distribution of page faults after the transition, aiming to observe any fluctuations or improvements in the page replacement process.
* **System Efficiency Post-Transition:** Evaluate the system’s overall performance and efficiency under the LRU algorithm. Consider response times, resource allocation, and overall workload management to gauge any noticeable improvements or changes.

**Comparison:** Compare the frequency of page faults experienced before and after the transition point. Analyze whether the LRU algorithm has a different impact on the number of page faults compared to FIFO, examining whether there’s a decrease, increase, or stability in the occurrence of faults.

**Overall System Efficiency:**

* **Evaluation Metrics:** Utilize various metrics like throughput, response times, and resource utilization to evaluate the overall system efficiency before and after the transition point.
* **Comparative Analysis:** Compare system performance metrics under both FIFO and LRU to assess whether the transition results in an overall improvement or decline in system efficiency.

**Adjusting the Threshold:**

Experiment with different threshold values to determine the most effective transition point. Modify the transition logic accordingly, retest the system, and analyze the effects on system performance.

# 4.ALGORITHM

Input: Number of frames, total number of pages, transition point for switching from FIFO to LRU

Output: Page replacement simulation and total page faults using FIFO

1. Begin main function

2. Declare variables: total\_pages, num\_frames, transitionPoint as integers

3. Print "Enter the number of frames: "

4. Read and store user input for num\_frames

5. Print "Enter the total number of pages: "

6. Read and store user input for total\_pages

7. Print "Enter the transition point for FIFO to LRU: "

8. Read and store user input for transitionPoint

9. Declare an array of structures: pages of type struct Page with size total\_pages

10. Print "Enter the page sequence:"

11. Loop from i = 0 to total\_pages - 1

a. Print "Page i + 1: "

b. Read and store user input for pages[i].pageNo

12. Call the FIFO function passing arguments: pages, total\_pages, num\_frames, transitionPoint

13. End main function

Function FIFO(pages[], total\_pages, num\_frames, transitionPoint):

1. Declare variables: page\_faults, frames[MAX\_FRAMES], frame\_count as integers

2. Initialize page\_faults, frame\_count to 0

3. Loop from i = 0 to total\_pages - 1

a. Retrieve current\_page from pages[i]

b. Initialize page\_found to 0

c. Loop from j = 0 to frame\_count - 1

i. If frames[j] is equal to current\_page

- Set page\_found to 1 and break the loop

d. If page\_found is false (page not found in frames)

i. If frame\_count is less than num\_frames

- Add current\_page to frames at position frame\_count

- Increment frame\_count

ii. Else, replace the oldest page in frames

- Shift the frames to simulate FIFO behavior

iii. Increment page\_faults

iv. Print "Page current\_page referenced: " with current frame contents

v. If page\_faults is equal to transitionPoint

- Print "Transitioning from FIFO to LRU..."

- Exit the loop and the function

4. Print "Total page faults using FIFO: " followed by page\_faults

5. End FIFO function

# 5.COMPARISON WITH EXISTING METHODS

**1. Hybrid Approach (Combining FIFO and LRU):**

**Bélády's Anomaly Mitigation:** The hybrid approach adapts the transition from FIFO to LRU based on a specific number of page faults. This transition aims to prevent sudden increases in page faults when additional memory frames are introduced. It's tailored to mitigate Bélády's Anomaly, ensuring more predictable and stable performance when more memory frames are added. By shifting to LRU strategically, it intends to avoid scenarios where increased memory leads to more page faults, which is a characteristic of Bélády's Anomaly.

**Balanced Performance:** This strategy aims to create a balance by incorporating FIFO's simplicity and LRU's efficiency. It's designed to perform well in diverse workloads where neither FIFO nor LRU might be optimal. The method is intended to adapt to varying workloads and memory configurations, providing a more dynamic and balanced performance.

**Adaptability and Flexibility:** The hybrid method's approach offers adaptability based on system behavior and observed patterns. By adjusting to the system's requirements, it can dynamically switch between FIFO and LRU, allowing it to operate efficiently in different scenarios.

**2. Comparative Analysis on Time and Space Complexity:**

**Time Complexity:**

* **FIFO:** With a time complexity of O(n), where 'n' is the total number of pages, FIFO has a linear time complexity.
* **LRU:** LRU can vary between O(1) to O(n), depending on its implementation with data structures like a queue or stack.
* **Hybrid Approach:** The hybrid method might have slightly higher time complexity due to the transition calculations between FIFO and LRU. The exact time complexity could depend on the transition point and the overall execution flow of the hybrid logic.

**Space Complexity:**

* **FIFO and LRU:** Both FIFO and LRU use data structures to hold page information. The space complexity depends on the size of frames or pages, utilizing arrays, lists, or other data structures.
* **Hybrid Approach:** The hybrid strategy includes a minimal additional space complexity, mainly attributed to transition calculations.

In summary, the hybrid method aspires to blend the advantages of both FIFO and LRU while aiming to circumvent issues like Bélády's Anomaly by strategically transitioning to LRU. It does so by maintaining a balanced performance between simplicity and efficiency, with minor overhead for transition calculations. The method's design allows it to cater to varied workloads, memory configurations, and system demands, showcasing adaptability and balanced performance in various scenarios.

# 6.RESULT AND DISCUSSIONS

**1. Hybrid Approach (Combining FIFO and LRU):**

**Bélády's Anomaly Mitigation:** The hybrid approach adapts the transition from FIFO to LRU based on a specific number of page faults. This transition aims to prevent sudden increases in page faults when additional memory frames are introduced. It's tailored to mitigate Bélády's Anomaly, ensuring more predictable and stable performance when more memory frames are added. By shifting to LRU strategically, it intends to avoid scenarios where increased memory leads to more page faults, which is a characteristic of Bélády's Anomaly.

**Balanced Performance:** This strategy aims to create a balance by incorporating FIFO's simplicity and LRU's efficiency. It's designed to perform well in diverse workloads where neither FIFO nor LRU might be optimal. The method is intended to adapt to varying workloads and memory configurations, providing a more dynamic and balanced performance.

**Adaptability and Flexibility:** The hybrid method's approach offers adaptability based on system behavior and observed patterns. By adjusting to the system's requirements, it can dynamically switch between FIFO and LRU, allowing it to operate efficiently in different scenarios.

**2. Comparative Analysis on Time and Space Complexity:**

**Time Complexity:**

* **FIFO:** With a time complexity of O(n), where 'n' is the total number of pages, FIFO has a linear time complexity.
* **LRU:** LRU can vary between O(1) to O(n), depending on its implementation with data structures like a queue or stack.
* **Hybrid Approach:** The hybrid method might have slightly higher time complexity due to the transition calculations between FIFO and LRU. The exact time complexity could depend on the transition point and the overall execution flow of the hybrid logic.

**Space Complexity:**

* **FIFO and LRU:** Both FIFO and LRU use data structures to hold page information. The space complexity depends on the size of frames or pages, utilizing arrays, lists, or other data structures.
* **Hybrid Approach:** The hybrid strategy includes a minimal additional space complexity, mainly attributed to transition calculations.

In summary, the hybrid method aspires to blend the advantages of both FIFO and LRU while aiming to circumvent issues like Bélády's Anomaly by strategically transitioning to LRU. It does so by maintaining a balanced performance between simplicity and efficiency, with minor overhead for transition calculations. The method's design allows it to cater to varied workloads, memory configurations, and system demands, showcasing adaptability and balanced performance in various scenarios.

## 7.CONCLUSION

The project centered on simulating a page replacement scheme using the First-In-First-Out (FIFO) approach within a system with a defined number of memory frames and a sequence of pages. The FIFO algorithm mimics the behavior of a cache in managing pages and replaces the oldest page upon encountering a page fault. The simulation allowed users to enter the number of frames and pages and set a transition point for switching from FIFO to another replacement algorithm like LRU (Least Recently Used).

The project's objective was to implement a simplified version of FIFO-based page replacement, demonstrating how pages are managed within a limited number of frames and how the system replaces pages as the memory fills up.

The code's structure allowed users to input page sequences and the number of frames to observe the page replacement sequence and the final count of page faults based on the FIFO algorithm. This practical simulation aided in understanding and visualizing the behavior of a basic page replacement technique and how the transition from FIFO to LRU could be triggered based on a defined threshold of page faults.

In conclusion, this project offered a simplified implementation of the FIFO page replacement scheme. However, its functionality is limited and does not cover complex scenarios and real-time system behavior. Yet, it provided an interactive and educational simulation, offering an entry-level understanding of the FIFO algorithm's behavior and the handling of page faults. This project could be further extended to include more advanced page replacement algorithms, additional system complexities, and a more comprehensive user interface for educational purposes

Project.

# 8.REFERENCES

Textbooks:

Operating System Concepts" by Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne.

Modern Operating Systems" by Andrew S. Tanenbaum and Herbert Bos.

Operating Systems: Three Easy Pieces" by Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau.

WEBSITES

* [GeeksforGeeks](https://www.geeksforgeeks.org/)
* [IEEE Xplore](https://ieeexplore.ieee.org/)
* [Springer Link](https://link.springer.com/)

RESEARCH PAPERS

How Anomalous Is Bélády's Anomaly? By -*Kirby McMaster ,Samuel Sambasivam , Nicole Anderson*

And Page Replacement Algorithms: Anomaly Cases. By - *Najeeb A. Al-Samarraie*

Removing Bélády’s Anomaly from Caches with Prefetch Data. By- *Elizabeth Varki*

# 9.Code with I/O

Code:

#include <stdio.h>

#define MAX\_FRAMES 10

struct Page {

int pageNo;

};

void FIFO(struct Page pages[], int total\_pages, int num\_frames, int transitionPoint) {

int page\_faults = 0;

int frames[MAX\_FRAMES];

int frame\_count = 0;

for (int i = 0; i < total\_pages; i++) {

int current\_page = pages[i].pageNo;

int page\_found = 0;

for (int j = 0; j < frame\_count; j++) {

if (frames[j] == current\_page) {

page\_found = 1;

break;

}

}

if (!page\_found) {

if (frame\_count < num\_frames) {

frames[frame\_count] = current\_page;

frame\_count++;

} else {

for (int j = 0; j < num\_frames - 1; j++) {

frames[j] = frames[j + 1];

}

frames[num\_frames - 1] = current\_page;

}

page\_faults++;

printf("Page %d referenced: ", current\_page);

for (int j = 0; j < num\_frames; j++) {

if (j < frame\_count) {

printf("%d ", frames[j]);

} else {

printf("- ");

}

}

printf("\n");

if (page\_faults == transitionPoint) {

printf("Transitioning from FIFO to LRU...\n");

// Call LRU function with the remaining pages

// LRU(pages + i + 1, total\_pages - i - 1, num\_frames);

break;

}

}

}

printf("Total page faults using FIFO: %d\n", page\_faults);

}

int main() {

int total\_pages, num\_frames, transitionPoint;

printf("Enter the number of frames: ");

scanf("%d", &num\_frames);

printf("Enter the total number of pages: ");

scanf("%d", &total\_pages);

printf("Enter the transition point for FIFO to LRU: ");

scanf("%d", &transitionPoint);

struct Page pages[total\_pages];

printf("Enter the page sequence:\n");

for (int i = 0; i < total\_pages; i++) {

printf("Page %d: ", i + 1);

scanf("%d", &pages[i].pageNo);

}

FIFO(pages, total\_pages, num\_frames, transitionPoint);

return 0;

}

I/O:

