CFLRU: A Replacement Algorithm for Flash Memory*

Seon-yeong Park, Dawoon Jung, Jeong-uk Kang, Jin-soo Kim, and Joonwon Lee

Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea

ACM CASES'06, October 23–25, 2006

Outline

- Introduction
- CFLRU Algorithm
- Simulation
- Implementation
- Conclusion

Introduction

- Flash memory
  - small and light weight
  - solid-state reliability
  - low power consumption
- The paper
  - focus on a cache replacement policy
    - the cache hit rate
    - the replacement cost
  - Clean-First LRU

Introduction

Table 1. The characteristics of flash memory [21]

NAND	**Access Time *(us/4KB)***	Read	284.2
		Write	1833.0
		Erase	499.2
	Energy Consumption *(uJ/4KB)*	Read	9.4
		Write	59.6
		Erase	16.5
NOR	**Access Time *(us/4KB)***	Read	53.8
		Write	14054.4
		Erase	15616.0
	Energy Consumption *(uJ/4KB)*	Read	8.6
		Write	3251.2
		Erase	3609.6

CFLRU Algorithm

- two kinds of replacement costs
  - a requested page is
    - fetched from secondary storage to the page cache in RAM
    - evicted from cache to secondary storage
  - A clean page
    - contains the same copy of the original data
    - can be just dropped from cache when it is evicted
  - Cache hit rate v.s. replacement cost
    - Space is not enough

CFLRU Algorithm

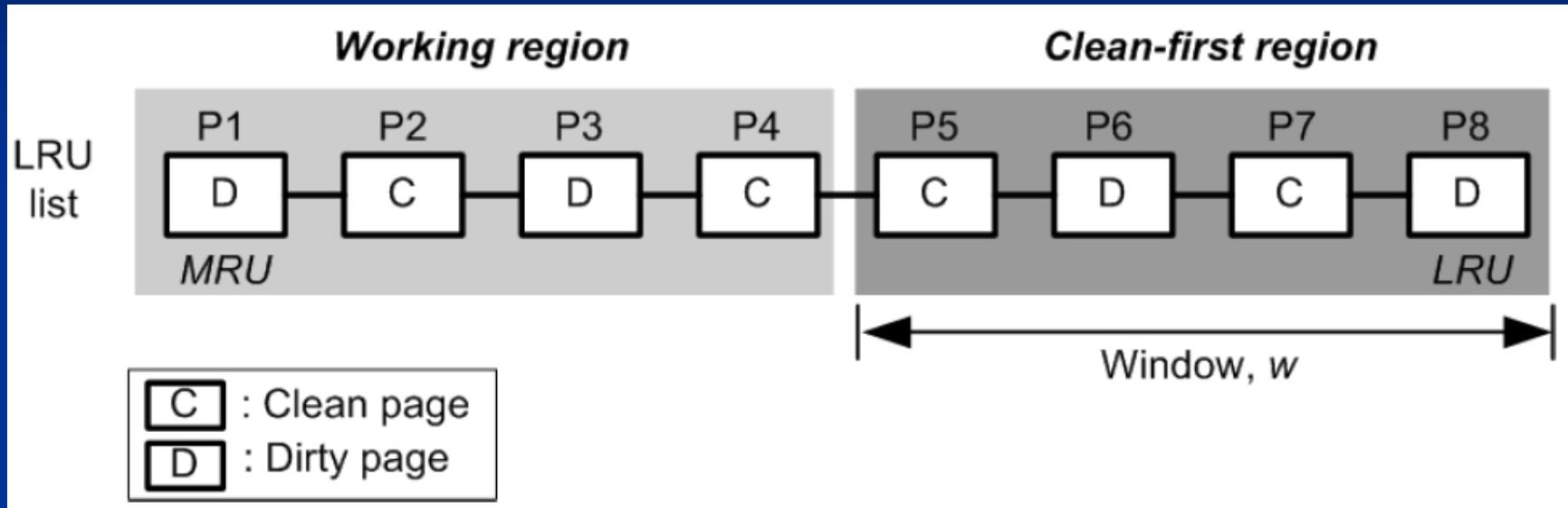


Figure 1. Example of CFLRU algorithm

- The victim page in the order
  - LRU : p8 -> p7 -> p6 -> p5
  - CFLRU: p7 -> p5 -> p8 -> p6

CFLRU Algorithm

- The Window Size
  - If windows size is too large, the hit rate falls dramatically
  - If windows size is too small, the number of evicted pages increases
- It is assumed that
  - C_W : the cost of a flash write operation
  - C_R : the cost of a flash read operation
  - N_D : the number of dirty pages that should have been evicted in the LRU order but are kept in the cache
    - The benefit : $C_W * N_D$
  - N_C : the number of clean pages that are evicted instead of dirty pages within the clean-first region
  - $Pi(k)$: the probability of the future reference of a clean page i , but is evicted at kth position
    - The cost : $\sum_{i}^{N_C} C_R \bullet P_i(k)$

CFLRU Algorithm

- The formula

$$\underset{w}{MAX}\left[C_W \bullet N_D - \sum_{i}^{N_C} C_R \bullet P_i(k)\right]$$

  - summarizes the proper window size, w, of the clean-first region

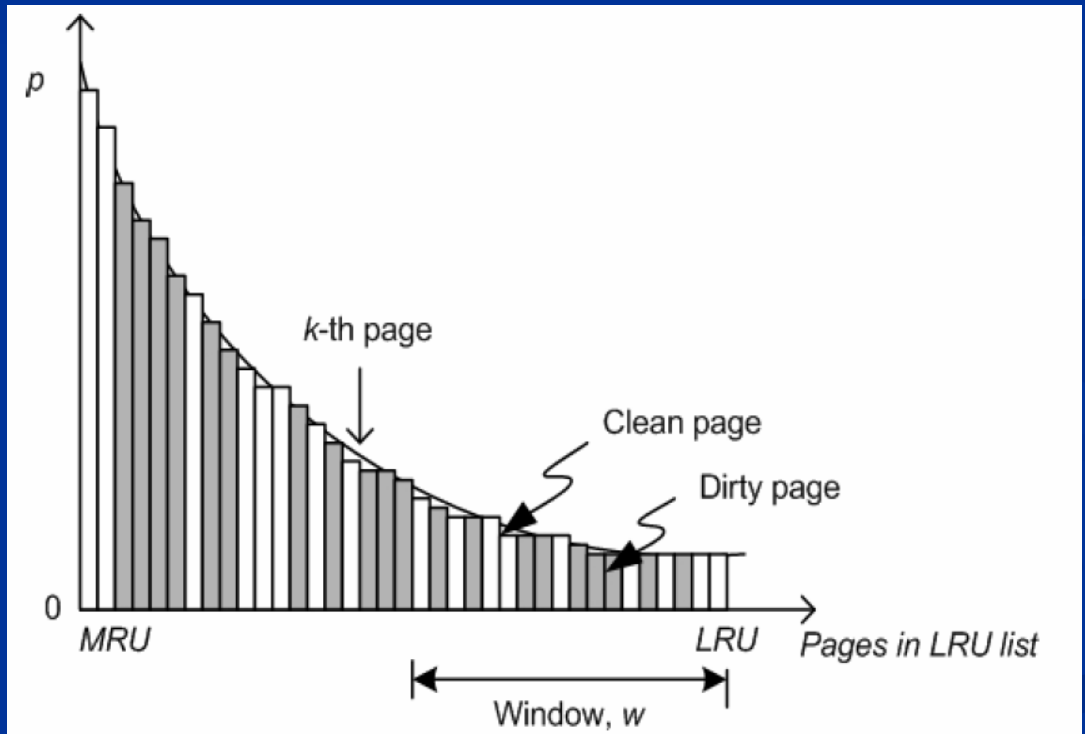


Figure 2. Window size of CFLRU algorithm

Simulation

- The simulation
  - is performed with two different types of real workload traces
    - The trace for swap system
      - gathered virtual memory reference traces using a profiling tool, called Valgrind
    - The trace for file system
      - gathered block reference traces directly from the buffer cache under the ext2 file system
  - is assumed that the system has
    - 32MB SDRAM
    - 128MB flash memory for swap space or file system
    - NAND flash memory

Simulation

- Simulation Result
  - The average replacement costs are reduced
    - In swap system
      - The CFLRU-static：28.4%
      - The CFLRU-dynamic：23.1%
    - In file system buffer cache
      - The CFLRU-static：26.2%
      - The CFLRU-dynamic：23.5%

Table 2. Scenarios used in gathering traces

Workloads	Scenarios
Gqview	Performs a slide show of seven images and adjusts their sizes.
Kword	Edits several lines and saves in a text document.
Kspread	Calculates sum, avg.,min., and max. of numerical data, and sorts them.
Acrobat reader	Views a PDF document.
Mozilla	Browses several web sites such as shopping mall, news center, mail server, etc.

Table 3. Workloads used in simulations

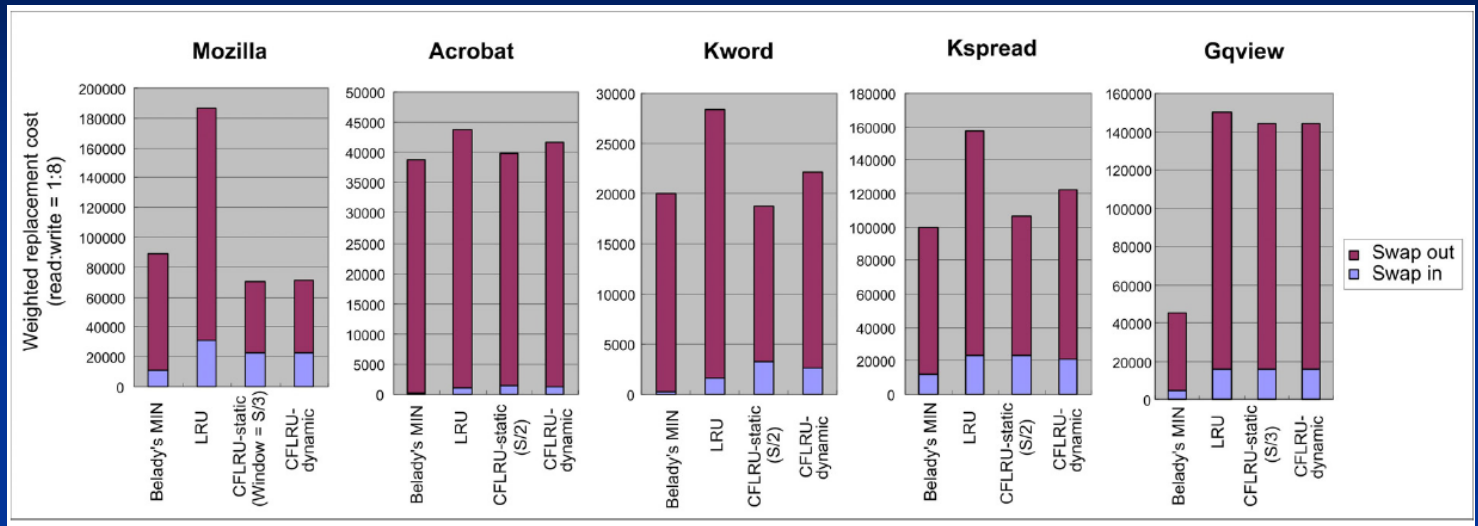
Workloads	Memory used (MB)	Memory references			
		Total	Instruction	Data	
			Read	Read	Write
Gqview	21.49	28,940,881	896,355 (3.1 %)	11,437,768 (39.5 %)	16,606,758 (57.4 %)
Kword	28.57	4,779,386	1,025,217 (21.4 %)	2,837,916 (59.4 %)	916,253 (19.2 %)
Kspread	40.24	11,366,261	1,515,521 (13.3 %)	6,264,476 (55.1 %)	3,586,264 (31.6%)
Acrobat reader	37.28	10,815,848	2,062,732 (19.1 %)	2,256,161 (20.9 %)	6,496,955 (60.0 %)
Mozilla	39.64	41,533,372	20,243,704 (48.7 %)	14,893,383 (35.9 %)	6,396,285 (15.4 %)

(a) Virtual memory references

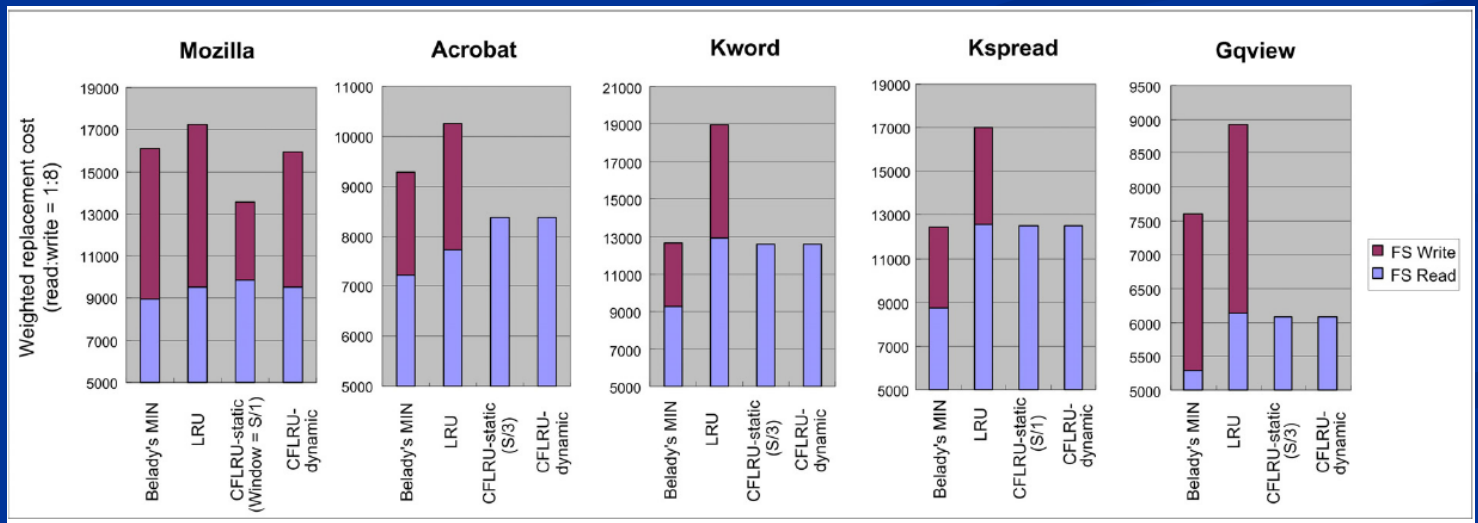
Workloads	Buffer used (MB)	Buffer block references		
		Total	Read	Write
Gqview	21.16	84,226	82,792 (98.3 %)	1,434 (1.7 %)
Kword	34.71	33,614	29,402 (87.5 %)	4,212 (12.5 %)
Kspread	33.63	34,719	33,115 (95.4 %)	1,604 (4.6 %)
Acrobat reader	28.86	37,884	36,244 (95.7 %)	1,640 (4.3 %)
Mozilla	35.89	39,141	25,557 (65.3 %)	13,584 (34.7 %)

(b) Buffer block references

Simulation



(a) Replacement cost for swap system



(b) Replacement cost for file system buffer cache

Implementation

- The Original Linux kernel 2.4 Page reclamation
  - the page cache that consists of two pseudo-LRU lists
    - the active list and the inactive list
  - 1. Reclaiming phase
    - Scanning the pages in the inactive list , then free pages
  - 2. Swap out phase
    - Round Robin fashion

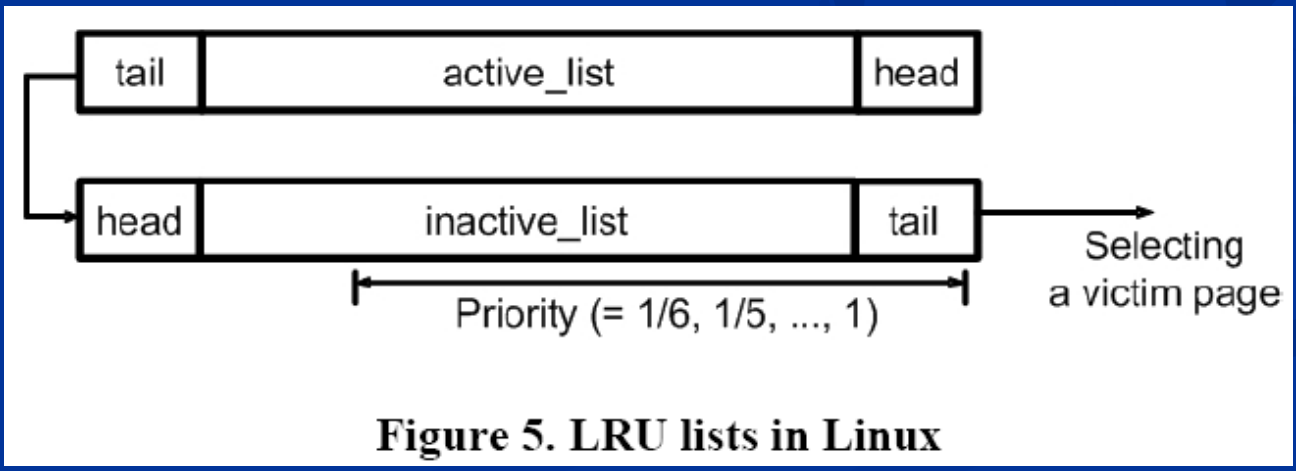


Figure 5. LRU lists in Linux

Implementation

- CFLRU page reclamation
  - The additional reclamation function consists of two phases
    - first：clean pages in the inactive list are evicted first until the enough number of pages is freed
      - Dirty pages are skipped
    - second：clean pages that belong to the process region are swapped out
  - A kernel daemon
    - periodically checks the replacement cost
  - Optimizations for flash memory
    - Remove sequential read-ahead function from Linux kernel

Implementation

- A system with
  - a Pentium IV processor
  - 32MB SDRAM
  - Linux kernel 2.4.28
  - Flash memory
    - 64MB for swap space
    - 256MB for file system
    - NAND flash memory
- Window size
  - The window size of the CFLRU static algorithm is 1/4 of the inactive list
- the performance measure
  - five applications; gcc, tar, diff, encoding, and file system benchmarks

Implementation

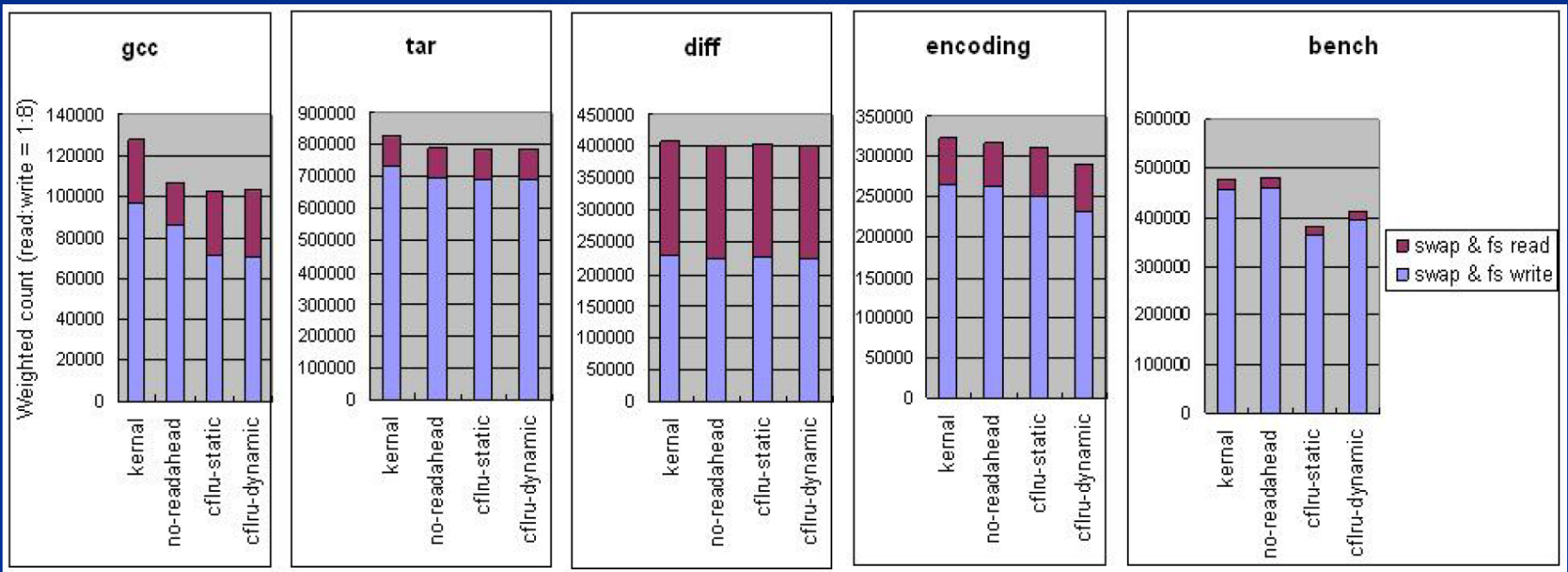
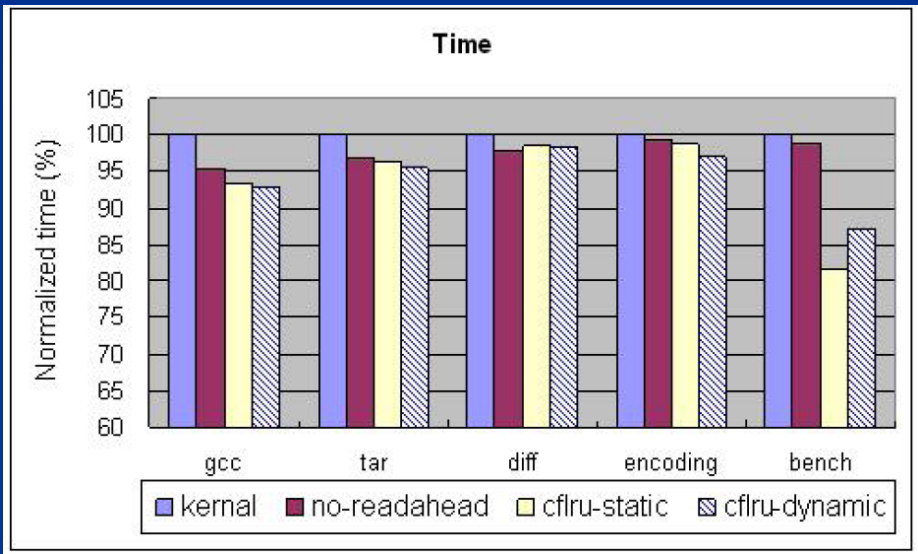


Figure 6. Weighted flash read and write counts under swap and file system cache

Implementation



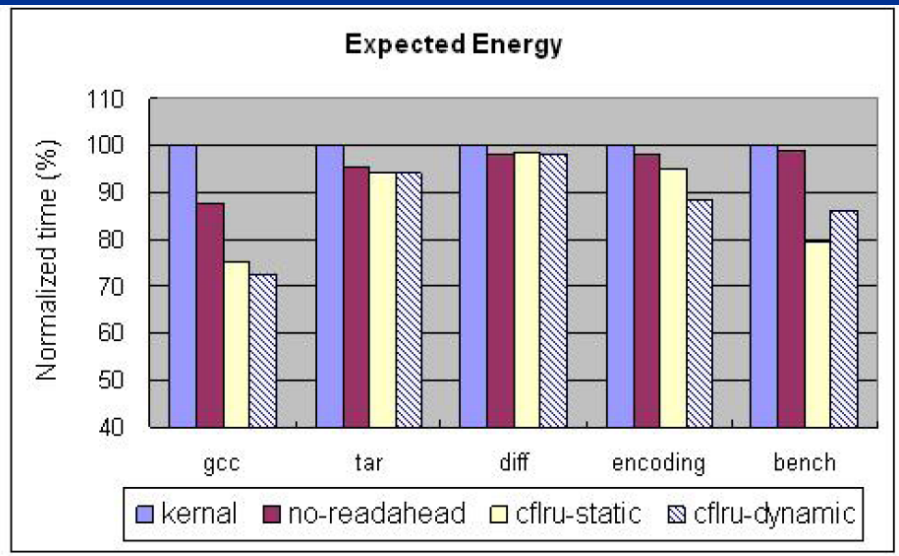


Figure 7. Time delay and expected energy

Conclusion

- The paper presents the Clean-First LRU (CFLRU) replacement algorithm for flash memory
- CFLRU tries to reduce the number of costly write and potential erase operations
- In the simulation
  - Static (dynamic) methods of CFLRU reduce the replacement cost
    - in swap system , reduced by 28.4% (23.1%)
    - In file system buffer cache , reduced by 26.2% (23.5%)
- In the implementation
  - Static (dynamic) methods of CFLRU
    - average time delays ：reduced by 6.2% (5.7%)
    - energy savings ：reduced by 11.4% (12.1%)