HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

BỘ MÔN HỆ ĐIỀU HÀNH

Báo Cáo Bài Tập Lớn

Nhóm 13

Họ và tên	Mã Sinh Viên
Nguyễn Quang Huy	B23DCKH058
Đỗ Đức Mạnh	B23DCKH074
Nguyễn Minh Quân	B23DCKH094
Vũ Văn Quyến	B23DCKH096
Nguyễn Đức Trường	B23DCKH124

Hà Nội 2025

Mục Lục

https://github.com/ducmanh202005/btlhdhnhom13.git

Phần 1 – GIỚI THIỆU ĐỀ TÀI

Trong bối cảnh các hệ thống lưu trữ hiện đại ngày càng phải xử lý lượng dữ liệu khổng lồ với yêu cầu tốc độ cao, việc tối ưu hiệu năng truy xuất dữ liệu trở thành một vấn đề quan trọng. Ổ đĩa cứng truyền thống (HDD) có ưu điểm dung lượng lớn và chi phí thấp, nhưng tốc độ truy cập chậm do sự phụ thuộc vào cơ cấu cơ học. Điều này khiến HDD trở thành nút thắt (bottleneck) trong nhiều hệ thống.

Sự xuất hiện của ổ đĩa thể rắn (SSD) với tốc độ truy xuất nhanh hơn hàng chục lần đã mở ra giải pháp sử dụng SSD như lớp **bộ nhớ đệm (cache)** nhằm tăng tốc độ cho hệ thống HDD. Mô hình **SSD Cache + HDD** giúp kết hợp ưu điểm của cả hai: SSD đảm nhiệm tốc độ, HDD đảm nhiệm dung lượng.

Đề tài của nhóm nhằm mục tiêu:

- **Xây dựng chương trình mô phỏng hoạt động của SSD Cache kết hợp HDD**,
- **Thử nghiệm và so sánh hiệu năng giữa hai chính sách ghi phổ biến: Write-Through và Write-Back**,
- **Đánh giá hiệu quả dựa trên các workload khác nhau** như Random, Sequential, Locality và Write-Heavy.

Việc mô phỏng giúp hiểu sâu hơn cách hệ thống lưu trữ hoạt động, tác động của từng chính sách ghi và lý do tại sao các doanh nghiệp/data center lựa chọn các chiến lược cache khác nhau. Đây là kiến thức nền quan trọng đối với các kỹ sư phần mềm, kỹ sư hệ thống và kỹ sư dữ liệu trong tương lai.

Phần 2 – LÝ THUYẾT SSD–HDD CACHE

2.1. Tổng quan về Cache trong hệ thống lưu trữ

Cache là một lớp bộ nhớ nhanh hơn được đặt giữa người dùng và thiết bị lưu trữ gốc nhằm giảm độ trễ truy xuất dữ liệu. Khi một block dữ liệu được truy cập, hệ thống sẽ:

- **Hit** nếu dữ liệu đã có trong SSD → truy xuất nhanh.
- **Miss** nếu dữ liệu chưa ở SSD → phải đọc từ HDD → chậm.

Mục tiêu của cache:
=> Tối đa hóa hit rate → giảm truy cập HDD → tăng tốc toàn hệ thống.

2.2. SSD làm Cache cho HDD

Trong mô hình SSD–HDD Cache, SSD đóng vai trò lưu tạm các block "nóng" (hot data) thường xuyên truy cập. HDD vẫn là bộ lưu trữ chính (main storage).

Ưu điểm:

- Tăng tốc độ đọc/ghi rõ rệt.
- Giảm tải cho HDD, kéo dài tuổi thọ.
- Tận dụng chi phí thấp của HDD + tốc độ cao của SSD.

Nhược điểm:

- Tốn dung lượng SSD.
- Cần thuật toán quản lý cache hiệu quả (LRU, LFU,…).
- Chính sách ghi ảnh hưởng lớn đến độ an toàn dữ liệu.

2.3. Hai chính sách ghi chính: Write-Through và Write-Back

(1) Write-Through Cache

- Mọi thao tác ghi được ghi **đồng thời** vào SSD cache và HDD.
- Dữ liệu luôn an toàn và đồng bộ.
- Nhưng tốc độ ghi bị giảm, phụ thuộc HDD.

Ưu điểm

- Đảm bảo tính toàn vẹn dữ liệu.
- Dễ cài đặt, ít rủi ro.

Nhược điểm

- Tốc độ ghi thấp hơn Write-Back.
- HDD vẫn bị truy cập nhiều.

(2) Write-Back Cache

- Dữ liệu được ghi **trước vào SSD** → nhanh.
- HDD chỉ được ghi lại khi block bị đẩy ra khỏi cache hoặc định kỳ.

Ưu điểm

- Tốc độ ghi rất cao.
- Giảm truy cập HDD → tăng độ bền HDD.

Nhược điểm

- Có rủi ro mất dữ liệu nếu mất điện trước khi đẩy xuống HDD.
- Cần cơ chế đồng bộ phức tạp hơn.

2.4. Các loại workload ảnh hưởng đến hiệu năng Cache

Các workload mô phỏng hành vi truy cập thực tế:

Workload	Đặc điểm	Tác động lên Cache
Random	Truy cập ngẫu nhiên	Phụ thuộc nhiều vào LRU, hiệu năng cải thiện rõ
Sequential	Tuần tự, không lặp lại block	Cache gần như vô dụng, hit rate thấp
Locality	Dữ liệu truy cập lặp lại	Cache hoạt động tối ưu, hit rate cao
Write-Heavy	Ghi chiếm tỷ lệ lớn	Write-Back vượt trội so với Write-Through

Nhóm sẽ dùng 4 workload này để đánh giá hiệu năng hai chính sách ghi.

Phần 3: THIẾT KẾ MÔ PHỎNG HỆ THỐNG SSD CACHE + HDD

3.1. MÔ HÌNH PHẦN CỨNG GIẢ LẬP

3.1.1. Hard Disk Drive (HDD)

Đặc điểm kỹ thuật:

- **Dung lượng mô phỏng**: 10,000 blocks
- **Kích thước mỗi block**: 4 KB
- **Tổng dung lượng**: 40 MB
- **Độ trễ đọc**: 8 ms
- **Độ trễ ghi**: 10 ms

Cấu trúc lưu trữ:

HDD được mô phỏng bằng mảng tuyến tính:
hdd[0], hdd[1], hdd[2], ..., hdd[9999]

Mỗi phần tử chứa:
- Block ID (định danh)
- Dữ liệu 4KB

Mục đích: Lưu trữ toàn bộ dữ liệu của hệ thống. Tốc độ chậm nhưng dung lượng lớn và chi phí thấp.

3.1.2. Solid State Drive Cache (SSD Cache)

Đặc điểm kỹ thuật:

- **Dung lượng mô phỏng**: 128 blocks (có thể điều chỉnh: 64, 128, 256)
- **Kích thước mỗi block**: 4 KB
- **Tổng dung lượng**: 512 KB
- **Độ trễ đọc**: 0.1 ms
- **Độ trễ ghi**: 0.2 ms

Cấu trúc lưu trữ:

SSD Cache được mô phỏng bằng mảng cache entries:
ssdCache[0], ssdCache[1], ..., ssdCache[127]

Mỗi entry bao gồm:
- Block ID (block nào từ HDD đang được cache)
- Dữ liệu
- Dirty Bit (cho Write-Back)
- Timestamp (cho LRU)
- Valid Bit (entry có đang được sử dụng không)

Mục đích: Lưu trữ dữ liệu được truy cập thường xuyên (hot data) để tăng tốc độ truy cập.

3.1.3. Bảng so sánh HDD vs SSD Cache

Tiêu chí	HDD	SSD Cache
Dung lượng	10,000 blocks	128 blocks
Tốc độ đọc	8 ms	0.1 ms
Tốc độ ghi	10 ms	0.2 ms
Chi phí	Thấp	Cao
Mục đích	Lưu trữ lâu dài	Tăng tốc truy cập

3.1.4. Tham số cấu hình hệ thống

```
// Cấu hình HDD
#define BLOCK_SIZE 4096          // 4KB
#define HDD_CAPACITY 10000       // 10,000 blocks
#define HDD_READ_LATENCY 8       // 8 milliseconds
#define HDD_WRITE_LATENCY 10     // 10 milliseconds

// Cấu hình SSD Cache
#define CACHE_SIZE 128           // 128 blocks (có thể thay đổi)
#define SSD_READ_LATENCY 0.1     // 0.1 milliseconds
#define SSD_WRITE_LATENCY 0.2    // 0.2 milliseconds

// Chính sách Write
#define WRITE_THROUGH 0
#define WRITE_BACK 1
```

3.2. CẤU TRÚC DỮ LIỆU

3.2.1. CacheEntry - Mục trong SSD Cache

```
typedef struct {
   int blockID;            // Block ID từ HDD (0 đến 9999)
   char data[BLOCK_SIZE];    // Dữ liệu thực tế của block (4KB)
   bool dirtyBit;          // 1 = đã sửa chưa ghi xuống HDD
   long long timestamp;      // Thời điểm truy cập gần nhất
   bool valid;             // 1 = entry đang được sử dụng
} CacheEntry;
```

Giải thích từng trường:

1. **blockID**:
   - Định danh block từ HDD (0 - 9999)
   - Dùng để map cache entry với block trên HDD
   - Ví dụ: blockID = 150 nghĩa là entry này chứa dữ liệu của hdd[150]
2. **data[BLOCK_SIZE]**:
   - Chứa nội dung thực tế của block (4096 bytes)
   - Copy từ HDD hoặc dữ liệu mới từ CPU
3. **dirtyBit**:
   - Chỉ dùng cho chính sách **Write-Back**
   - = 0: Dữ liệu trong cache và HDD giống nhau (đã đồng bộ)
   - = 1: Dữ liệu trong cache mới hơn HDD (chưa đồng bộ)
   - Khi evict entry có dirty = 1, phải ghi xuống HDD trước
4. **timestamp**:
   - Ghi lại thời điểm truy cập gần nhất
   - Dùng cho thuật toán LRU (Least Recently Used)
   - Giá trị lớn = truy cập gần đây, giá trị nhỏ = lâu không dùng
5. **valid**:
   - = 0: Entry trống, chưa sử dụng
   - = 1: Entry đang chứa dữ liệu hợp lệ

3.2.2. HDDEntry - Mục trong HDD

```
typedef struct {
   int blockID;            // Số thứ tự block (0 đến 9999)
   char data[BLOCK_SIZE];    // Dữ liệu 4KB
} HDDEntry;
```

Đơn giản hơn CacheEntry vì:

- Không cần timestamp (HDD không dùng LRU)
- Không cần dirtyBit (HDD là bộ nhớ chính)

- Không cần valid (tất cả block đều hợp lệ)

3.2.3. StorageSystem - Hệ thống tổng thể

```
typedef struct {
   // Phần cứng
   CacheEntry ssdCache[CACHE_SIZE];    // Mảng SSD cache
   HDDEntry hdd[HDD_CAPACITY];         // Mảng HDD

   // Thống kê hiệu năng
   long long cacheHits;              // Số lần trúng cache
   long long cacheMisses;            // Số lần không trúng cache
   long long totalReads;             // Tổng số lần đọc
   long long totalWrites;            // Tổng số lần ghi
   long long totalReadLatency;       // Tổng thời gian đọc (ms)
   long long totalWriteLatency;      // Tổng thời gian ghi (ms)

   // Cấu hình
   int writePolicy;                  // WRITE_THROUGH hoặc WRITE_BACK
   long long currentTime;            // Đồng hồ hệ thống
} StorageSystem;
```

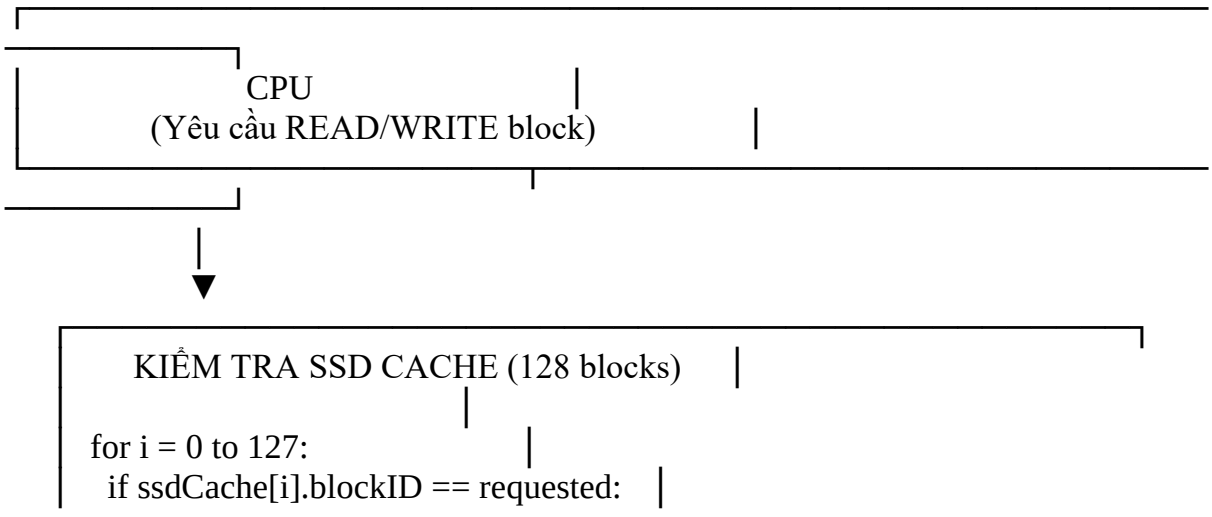
Các chỉ số hiệu năng quan trọng:

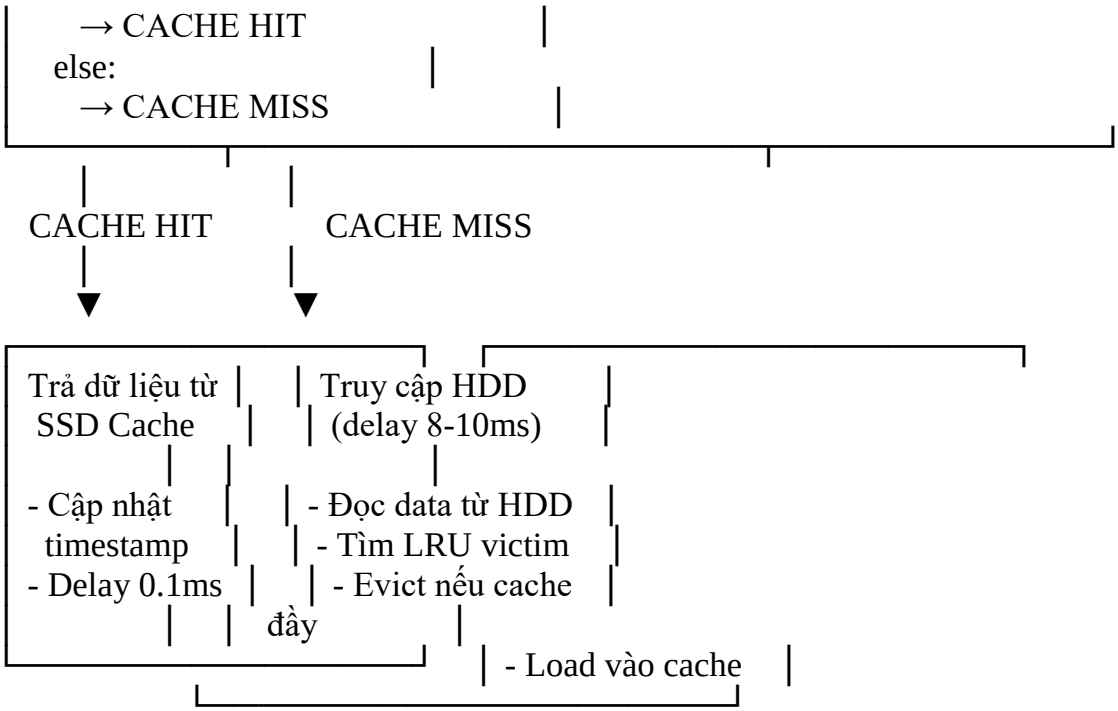
- **Hit Rate** = cacheHits / (cacheHits + cacheMisses) × 100%
- **Miss Rate** = cacheMisses / (cacheHits + cacheMisses) × 100%
- **Average Read Latency** = totalReadLatency / totalReads
- **Average Write Latency** = totalWriteLatency / totalWrites

---

3.3. LUỒNG ĐỌC/GHI DỮ LIỆU

3.3.1. Sơ đồ tổng quan hệ thống





3.3.2. Quy trình READ (Đọc dữ liệu)

INPUT: Block ID cần đọc (ví dụ: block 150)

OUTPUT: Dữ liệu của block

CÁC BƯỚC THỰC HIỆN:

Bước 1: Tìm kiếm trong SSD Cache

```
for (i = 0; i < CACHE_SIZE; i++) {
   if (ssdCache[i].valid && ssdCache[i].blockID == 150) {
      → CACHE HIT
      → Chuyển sang Bước 2
   }
}
→ CACHE MISS
→ Chuyển sang Bước 3
```

Bước 2: Xử lý CACHE HIT

- Cập nhật timestamp = currentTime
- Delay 0.1ms (độ trễ đọc SSD)
- statistics.cacheHits++
- statistics.totalReadLatency += 0.1
- Trả về ssdCache[i].data
- KẾT THÚC

Bước 3: Xử lý CACHE MISS

- statistics.cacheMisses++
- Đọc dữ liệu từ hdd[150]
- Delay 8ms (độ trễ đọc HDD)
- statistics.totalReadLatency += 8

Bước 4: Cập nhật SSD Cache

Case 1: Cache chưa đầy (có entry.valid == false)
- Tìm entry trống đầu tiên
- Load block 150 vào entry đó
- Đặt valid = true, timestamp = currentTime

Case 2: Cache đã đầy (tất cả entry.valid == true)
- Gọi hàm FIND_LRU_VICTIM()
- Nếu victim.dirtyBit == 1:
  + Ghi victim xuống HDD (delay 10ms)
- Thay thế victim bằng block 150
- Cập nhật timestamp

Bước 5: Trả dữ liệu cho CPU

THỜI GIAN THỰC HIỆN:

- Cache Hit: ~0.1 ms
- Cache Miss (cache chưa đầy): ~8.1 ms
- Cache Miss (phải evict): ~8.1 ms (hoặc ~18.1 ms nếu evict dirty block)

3.3.3. Quy trình WRITE với Write-Through

INPUT: Block ID cần ghi (ví dụ: block 200), dữ liệu mới

OUTPUT: Xác nhận ghi thành công

CÁC BƯỚC THỰC HIỆN:

Bước 1: Kiểm tra block có trong cache không

```
found = false
for (i = 0; i < CACHE_SIZE; i++) {
   if (ssdCache[i].valid && ssdCache[i].blockID == 200) {
      found = true
      → Chuyển sang Bước 2
   }
}
if (!found) → Chuyển sang Bước 3
```

Bước 2: Block có trong cache

- Cập nhật ssdCache[i].data = new_data
- Cập nhật timestamp = currentTime
- Delay 0.2ms (ghi SSD)

Bước 3: Block không có trong cache

- Tìm LRU victim (nếu cache đầy)
- Evict victim (ghi dirty nếu cần)
- Load block 200 vào cache
- Cập nhật data = new_data
- Delay 0.2ms

Bước 4: GHI ĐỒNG BỘ XUỐNG HDD (đặc trưng của Write-Through)

- Ghi hdd[200].data = new_data
- Delay 10ms (ghi HDD)
- Đảm bảo cache và HDD luôn giống nhau

Bước 5: Trả kết quả cho CPU

THỜI GIAN THỰC HIỆN:

- Luôn ~10ms (phụ thuộc vào HDD write latency)

ƯU ĐIỂM:

- Dữ liệu an toàn (luôn có trên HDD)
- Không cần xử lý dirty bit phức tạp
- Dễ implement

NHƯỢC ĐIỂM:

- Tốc độ ghi chậm (phụ thuộc HDD)
- Không tận dụng được tốc độ SSD khi ghi

3.3.4. Quy trình WRITE với Write-Back

INPUT: Block ID cần ghi (ví dụ: block 300), dữ liệu mới

OUTPUT: Xác nhận ghi thành công

CÁC BƯỚC THỰC HIỆN:

Bước 1: Kiểm tra block có trong cache không

(Tương tự Write-Through)

Bước 2: Block có trong cache

- Cập nhật ssdCache[i].data = new_data
- Đặt ssdCache[i].dirtyBit = 1 ← KHÁC Write-Through
- Cập nhật timestamp = currentTime
- Delay 0.2ms (chỉ ghi SSD)

Bước 3: Block không có trong cache

- Tìm LRU victim
- Nếu victim.dirtyBit == 1:
  + GHI victim.data xuống HDD
  + Delay 10ms
- Load block 300 vào cache
- Cập nhật data = new_data
- Đặt dirtyBit = 1
- Delay 0.2ms

Bước 4: TRẢ KẾT QUẢ NGAY (KHÔNG ghi xuống HDD)

- CPU nhận xác nhận ghi thành công
- Dữ liệu chỉ tồn tại trên SSD cache
- HDD chưa được cập nhật

Bước 5: Ghi xuống HDD sau (Lazy Write)

Chỉ xảy ra khi:
- Block bị evict khỏi cache
- Hoặc có lệnh FLUSH cache
- Hoặc tắt hệ thống

THỜI GIAN THỰC HIỆN:

- Cache Hit: ~0.2 ms (rất nhanh!)
- Cache Miss: ~0.2 ms đến ~10.2 ms (tùy dirty victim)

ƯU ĐIỂM:

- Ghi cực nhanh (chỉ phụ thuộc SSD)
- Giảm số lần ghi xuống HDD → tăng tuổi thọ HDD
- Hiệu năng tổng thể cao

NHƯỢC ĐIỂM:

- Mất điện/SSD lỗi → mất dữ liệu chưa đồng bộ
- Cần cơ chế flush cache định kỳ
- Phức tạp hơn trong implementation

3.3.5. So sánh Write-Through vs Write-Back

Tiêu chí	Write-Through	Write-Back
Tốc độ ghi	Chậm (~10ms)	Nhanh (~0.2ms)
Độ an toàn	Cao	Thấp hơn
Tính nhất quán	Luôn đồng bộ	Có thể không đồng bộ
Tuổi thọ HDD	Thấp hơn	Cao hơn
Độ phức tạp	Đơn giản	Phức tạp
Sử dụng dirty bit	Không	Có

3.4. THUẬT TOÁN THAY THẾ LRU

3.4.1. Nguyên lý hoạt động

LRU (Least Recently Used) loại bỏ block **ít được sử dụng gần đây nhất**.

Ý tưởng:

- Mỗi entry có một timestamp ghi thời điểm truy cập cuối
- Khi cache đầy, tìm entry có timestamp NHỎ NHẤT
- Entry đó là "victim" - sẽ bị thay thế

Lý do hiệu quả:

- Dữ liệu được truy cập gần đây → khả năng cao sẽ được truy cập lại
- Dữ liệu lâu không dùng → ít khả năng cần trong tương lai gần

3.4.2. Ví dụ minh họa

Giả sử: Cache size = 4 blocks

Trạng thái ban đầu:

Index	BlockID	Timestamp	Dirty	Valid

0	5	100	0	1
1	12	150	1	1
2	7	80	0	1
3	20	200	0	1

Yêu cầu: Đọc block 33 (không có trong cache) tại time = 220

Phân tích:

- Cache đầy (tất cả valid = 1)
- Cần tìm LRU victim
- Timestamp: {100, 150, 80, 200}
- Timestamp nhỏ nhất = 80 → Index 2 (block 7)

Các bước thực hiện:

1. **Tìm victim**: Index 2 (timestamp = 80)
2. **Kiểm tra dirty**: Block 7 có dirty = 0 → Không cần ghi HDD
3. **Loại bỏ**: Xóa block 7 khỏi cache
4. **Load block mới**: Đọc block 33 từ HDD (delay 8ms)
5. **Cập nhật cache**:
    - ssdCache[2].blockID = 33
    - ssdCache[2].timestamp = 220
    - ssdCache[2].valid = 1

Trạng thái sau:

Index	BlockID	Timestamp	Dirty	Valid
0	5	100	0	1
1	12	150	1	1
2	**33**	**220**	0	1
3	20	200	0	1

3.4.3. Trường hợp đặc biệt: Evict Dirty Block

Giả sử: Tiếp tục từ trạng thái trên, yêu cầu đọc block 50 tại time = 250

Phân tích:

- LRU victim: Index 0 (timestamp = 100)
- Block 5 có dirty = 0 → OK

Bây giờ, yêu cầu đọc block 77 tại time = 300:

Trạng thái hiện tại:

Index	BlockID	Timestamp	Dirty
0	50	250	0
1	12	150	**1**
2	33	220	0
3	20	200	0

Phân tích:

- LRU victim: Index 1 (timestamp = 150)
- Block 12 có **dirty = 1** → Phải ghi xuống HDD trước!

Các bước thực hiện:

1. **Tìm victim**: Index 1 (block 12, timestamp = 150)
2. **Kiểm tra dirty**: dirty = 1
3. **Flush dirty block**:
    - Ghi hdd[12].data = ssdCache[1].data
    - Delay 10ms
4. **Load block mới**: Đọc block 77 từ HDD (delay 8ms)
5. **Cập nhật cache**: Load block 77 vào index 1

Tổng thời gian: 10ms (flush) + 8ms (read) = 18ms

Phần 4: Cài đặt chương trình.

- Áp dụng lý thuyết và mô hình của phần 3, tiến hành cài đặt 2 chương trình mô phỏng:

1)Write_throught.py

⇨ Sử dụng cách ghi write_through mỗi khi ghi sẽ tiến hành ghi đồng thời vào cả SSD và HDD.

2)Write_back.py

⇨ Sử dụng cách ghi write_back mỗi khi ghi thì sẽ chỉ ghi vào mỗi SSD, và từ từ đẩy xuống HDD qua thuật toán LRU khi SSD đã đầy.

Phần 5: Kết quả, đánh giá và kết luận.

5.1. Bảng kết quả

BẢNG SO SÁNH 4 LOẠI WORKLOAD (WRITE-THROUGH)

Chỉ số	Random	Sequential	Locality	Write-Heavy
Hit Rate (%)	27.08%	0.00%	63.77%	52.63%
Miss Rate (%)	72.92%	100.00%	36.23%	47.37%
Số lần truy cập HDD (Read)	78	150	40	38
Số lần truy cập HDD (Write)	52	37	31	81
Thời gian Read (ms)	281.30	904.00	204.40	73.00
Thời gian Write (ms)	530.40	377.40	316.20	826.20
Tổng thời gian xử lý (ms)	811.70	1281.40	520.60	899.20

BẢNG SO SÁNH 4 LOẠI WORKLOAD (WRITE-BACK)

Chỉ số	Random	Sequential	Locality	Write-Heavy
Hit Rate (%)	27.08%	0.00%	63.77%	52.63%
Miss Rate (%)	72.92%	100.00%	36.23%	47.37%
Số lần truy cập HDD (Read)	78	150	40	38
Số lần truy cập HDD (Write)	44	37	23	32
Thời gian Read (ms)	281.30	904.00	204.40	73.00
Thời gian Write (ms)	450.40	377.40	236.20	336.20
Tổng thời gian xử lý (ms)	731.70	1281.40	440.60	409.20

Sự khác biệt về hiệu năng giữa Write-Through và Write-Back chủ yếu xoay quanh hai yếu tố: Số lần ghi xuống HDD và Tổng thời gian xử lý.

5.1.1 Tỷ lệ Hit Rate và HDD Read

- Hit Rate/Miss Rate: Tỷ lệ này hoàn toàn giống nhau ở cả hai cách ghi . Điều này xác nhận rằng thuật toán thay thế (LRU) và tính chất của workload quyết định khả năng Cache Hit/Miss, không phải chính sách ghi.
    - Sequential (0.00% Hit Rate): Đây là kịch bản thất bại của cache. Do mọi truy cập đều là Miss, hiệu năng bị chi phối bởi độ trễ HDD Read (8ms), khiến tổng thời gian xử lý rất cao và không có sự khác biệt giữa hai chính sách.
    - Locality (63.77% Hit Rate): Cache hoạt động hiệu quả nhất.
- Số lần truy cập HDD (Read): Cũng giống nhau vì cả hai chính sách đều sử dụng chiến lược Write-Allocate (phải đọc block từ HDD vào cache khi Write Miss để block đó có thể phục vụ cho các lần truy cập sau).

5.1.2. Số lần Ghi HDD

Đây là minh chứng rõ ràng nhất cho hiệu quả của Dirty Bit trong chính sách Write-Back.

- Write-Through (WT): Số lần ghi HDD luôn bằng số lần ghi của ứng dụng (trừ trường hợp ghi trùng lặp trong bộ đệm HDD). Ví dụ, Write-Heavy có 81 thao tác ghi → 81 lần ghi HDD.
- Write-Back (WB): Giảm đáng kể số lần ghi HDD nhờ Gộp ghi (Write Coalescing).
    - Trong Write-Heavy, 81 thao tác ghi chỉ dẫn đến 32 lần ghi HDD (flush), giảm 60.5%. Tức là 49 lần cập nhật dữ liệu trên SSD đã được gộp lại.
    - Trong Locality, 31 thao tác ghi chỉ dẫn đến 23 lần ghi HDD, giảm 26%.

5.1.3. Phân tích Độ trễ Tổng thể (Tổng thời gian xử lý)

Workload	Độ trễ Ghi WT/op	Độ trễ Ghi WB/op	Tốc độ tăng của WB
Random	10.2 m	458.40/52≈8.8 ms	9.8%
Locality	10.2 ms	236.20/31≈7.6 ms	15.3%
Write-Heavy	826.20/81≈10.2 ms	336.20/81≈4.15 ms	54.5%

- Write-Heavy: Sự giảm 60.5% số lần ghi HDD (từ 81 xuống 32) và việc thay thế chi phí 10.2 ms/op bằng chi phí 0.2 ms/op (Write Hit) đã giúp Write-Back vượt trội, nhanh hơn gần nửa giây.
- Locality: Write-Back cũng mang lại lợi ích đáng kể vì các thao tác ghi được trì hoãn và gộp lại.
- Sequential: Không có lợi ích nào vì hiệu năng bị chi phối hoàn toàn bởi 150 lần đọc HDD (Read Misses).

5.1.4. Kết luận

Chính sách	Ưu điểm	Nhược điểm
Write-Through (WT)	An toàn dữ liệu tuyệt đối. Dữ liệu luôn nhất quán	Hiệu năng Ghi kém. Luôn bị giới hạn bởi độ trễ

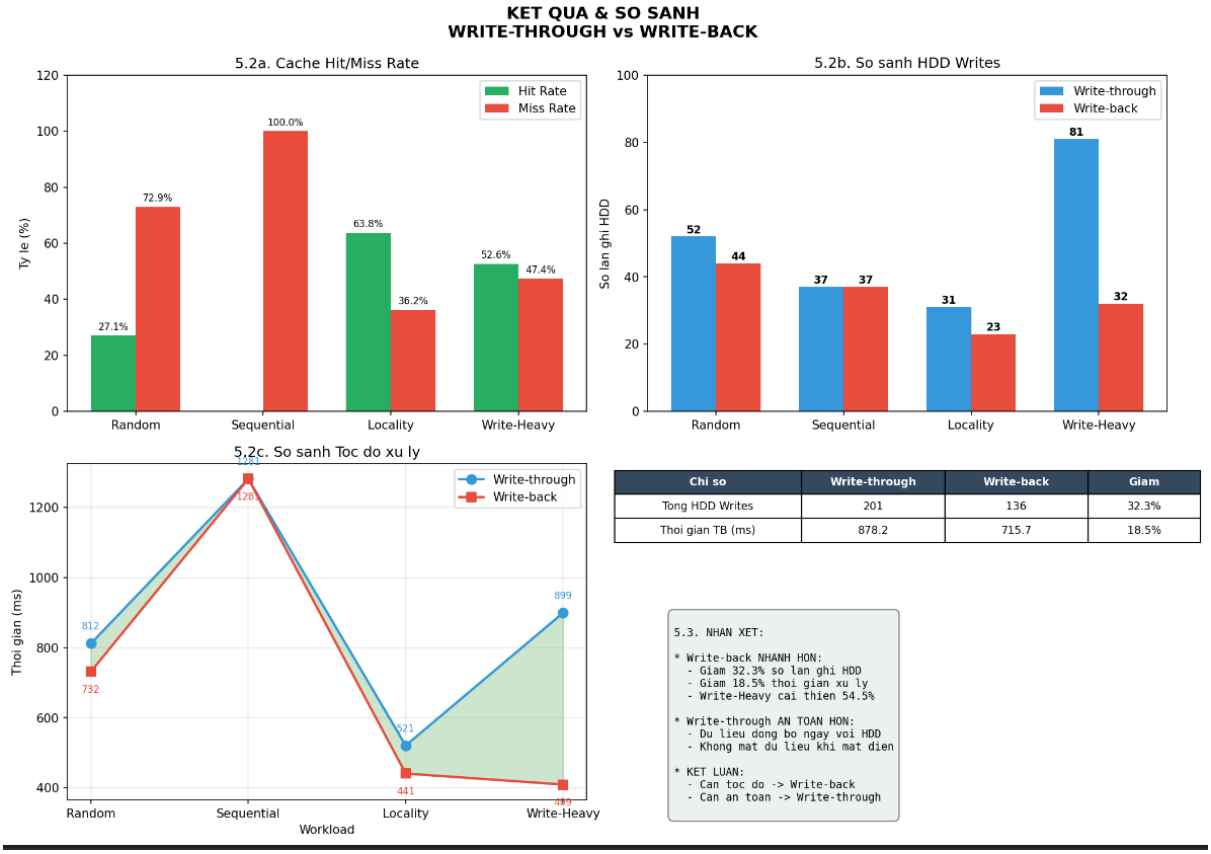
	giữa cache và HDD.	HDD (10ms) cho mỗi thao tác ghi.
Write-Back (WB)	Hiệu năng cao vượt trội. Giảm độ trễ ghi (đặc biệt cho Write-Heavy) và giảm tải cho HDD (giảm 60.5% Write).	Rủi ro mất dữ liệu. Dữ liệu chưa đồng bộ (Dirty Bit) sẽ bị mất nếu hệ thống gặp sự cố mất điện.

Đề xuất:

1. Sử dụng Write-Back: Khi ưu tiên tốc độ và giảm tải cho HDD, đặc biệt với workload có tính cục bộ tốt hoặc khối lượng ghi lớn (Write-Heavy).
2. Sử dụng Write-Through: Khi tính toàn vẹn dữ liệu là yêu cầu bắt buộc và hệ thống không có biện pháp bảo vệ nguồn điện (UPS) đáng tin cậy.

5.2. Biểu đồ

- **Biểu đồ cột so sánh hit/miss**



5.2.1. Biểu đồ 5.2a: Tỷ lệ Cache Hit/Miss

Biểu đồ cột này so sánh Tỷ lệ Hit (màu xanh) và Tỷ lệ Miss (màu đỏ) trên 4 loại workload.

- Không phụ thuộc vào Chính sách Ghi: Tỷ lệ Hit/Miss Rate hoàn toàn giống nhau cho cả Write-Through (WT) và Write-Back (WB). Điều này khẳng định một nguyên tắc cơ bản: Chính sách ghi không ảnh hưởng đến khả năng tìm thấy dữ liệu (Hit Rate). Hit Rate chỉ phụ thuộc vào thuật toán thay thế (LRU) và tính cục bộ (Locality) của workload.
- Tác động của Workload:
    - Sequential (0.0% Hit): Thể hiện thất bại hoàn toàn của cache (cache thrashing). Mọi truy cập là Miss, buộc hệ thống phải đọc từ HDD, không tận dụng được cache.
    - Locality (63.8% Hit): Lý tưởng nhất. LRU hoạt động hiệu quả khi dữ liệu được truy cập lặp lại.
    - Write-Heavy (52.6% Hit): Tỷ lệ Hit tốt, cho thấy các thao tác ghi tập trung vào một tập dữ liệu nóng (Hot Data Set) được giữ lại trong cache.

5.2.2. So sánh Số lần Ghi xuống HDD

Biểu đồ này là minh chứng rõ ràng cho lợi ích giảm tải của cơ chế Dirty Bit trong Write-Back.

Workload	WT (HDD Writes)	WB (HDD Writes)	Tỷ lệ giảm	Tác động
Write-Heavy	81	32	↓60.5%	Giảm tải tối đa. Write-Back gộp 49 lần ghi (Write Coalescing), chỉ cần 32 lần Flush.
Locality	31	23	↓25.8%	Gộp ghi hiệu quả, giảm 8 lần truy cập HDD Write 10ms.

Sequential	37	37	0%	Không giảm. Mọi Write là Miss, block bị đẩy ra ngay lập tức ->phải Flush.

Write-Through luôn là 1:1 (một Write operation ->một HDD Write), thể hiện sự kém hiệu quả trong việc sử dụng HDD.

Write-Back tận dụng Dirty Bit để trì hoãn và gộp các thao tác ghi. Điều này kéo dài tuổi thọ của HDD và là cơ sở để giảm độ trễ tổng thể.

5.2.3. Biểu đồ 5.2c: So sánh Tốc độ Xử lý

Biểu đồ minh họa tăng tốc độ do WB mang lại.

- Write-Heavy (WT 899 ms vs WB 409 ms):
  - WB nhanh hơn 54.5%. Đây là sự khác biệt lớn nhất. Chi phí tiết kiệm được là 490 ms (tương đương 49 lần ghi HDD bị loại bỏ), chứng minh Write-Back là giải pháp tối ưu cho Workload Write-Heavy.
- Locality (WT 521 ms vs WB 441 ms):
  - WB nhanh hơn 15.3%. Lợi ích đến từ Hit Rate cao (giảm Read Latency) và gộp Write (giảm 8 lần ghi HDD).
- Random (WT 812 ms vs WB 732 ms):
  - WB nhanh hơn 9.8%. Lợi ích nhỏ hơn do Hit Rate thấp.
- Sequential (WT 1281 ms vs WB 1281 ms):
  - Không chênh lệch. Hiệu năng bị chi phối bởi chi phí Read Miss Latency (8ms/lần), vượt qua mọi lợi ích của chính sách ghi.

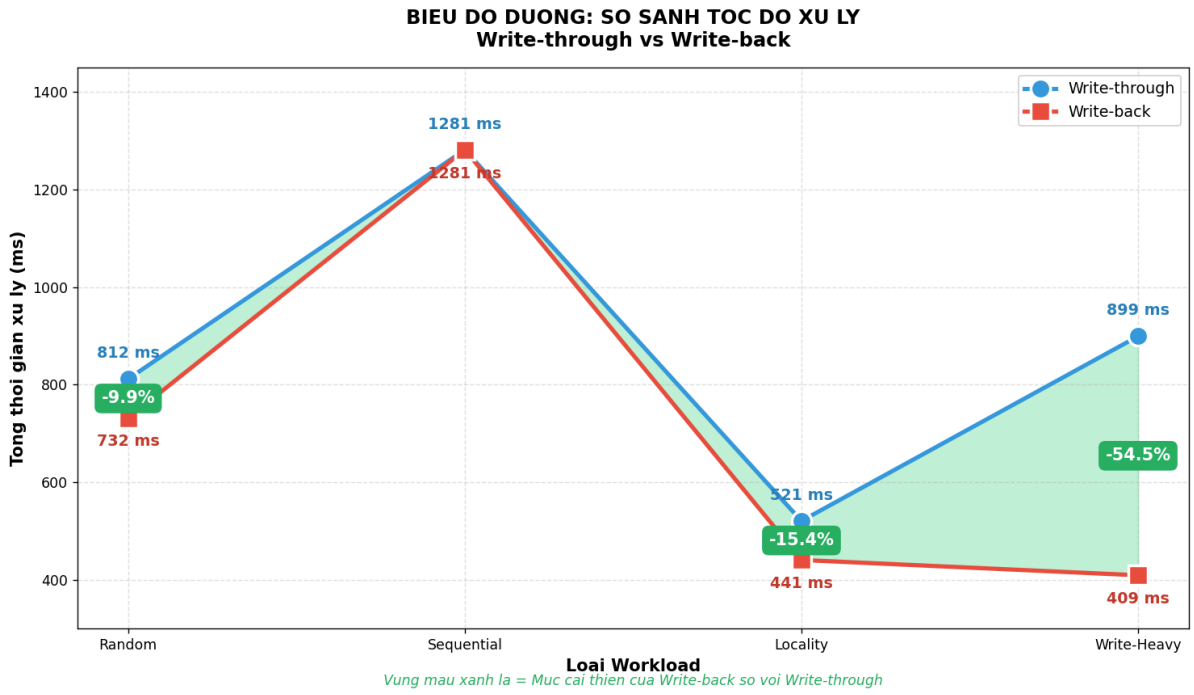
5.2.4. Khối Bảng và Khối Nhận xét

Chỉ số	Write-Through	Write-Back	Giảm
Tổng HDD Writes	201	136	↓32.3%
Tổng thời gian (ms)	878.2	715.7	↓18.5%

- Phân tích: Các con số này đại diện cho hiệu năng trung bình của WB trên cả 4 workload, cho thấy khả năng tăng tốc tổng thể là 18.5%

- Nhận xét: Khối nhận xét tóm tắt chính xác sự cân bằng:
    - Tốc độ -> Write-Back: Tăng tốc 54.5% cho Write-Heavy, giảm 32.3% tổng HDD Writes.
    - An toàn -> Write-Through: Dữ liệu luôn đồng bộ ngay lập tức, loại bỏ rủi ro mất dữ liệu chưa ghi (Dirty Data).

5.3 Biểu đồ đường so sánh tốc độ



5.3.1. Tổng quan Biểu đồ

Biểu đồ so sánh Tổng thời gian xử lý (ms), trong đó:

- Đường màu Xanh dương (Blue) là Write-Through (WT).
- Đường màu Đỏ/Cam là Write-Back (WB).
- Vùng màu Xanh lá thể hiện mức độ cải thiện hiệu suất (thời gian tiết kiệm) của Write-Back so với Write-Through.

Nguyên tắc: Đường WB càng thấp hơn WT, hiệu suất WB càng cao.

5.3.2. Chi tiết Từng Workload (Tác động của Workload)

Workload	WT (ms)	WB (ms)	Chênh lệch (WB - WT)	Tăng tốc/Cải thiện	Phân tích Nguyên nhân
Sequential	1281	1281	0 ms	0.0%	Không có sự khác biệt. Hiệu năng bị chi phối bởi 100%Read Miss (chi phí đọc HDD 8 ms/lần), làm lu mờ mọi lợi ích về Write.
Random	812	732	-80 ms	↑9.9%	Lợi ích đến từ việc giảm số lần ghi HDD (Write Coalescing), tiết kiệm 80 ms (tương đương 8 lần ghi HDD bị loại bỏ: 8 x 10 ms).

Locality	521	441	-80 ms	↑15.4%	Cải thiện tốt do Hit Rate cao (giảm chi phí Read) và giảm 8 lần ghi HDD.
Write-Heavy	899	409	-490 ms	↑54.5%	Vượt trội tuyệt đối. WB loại bỏ Write Bottleneck của HDD, chuyển 49 lần ghi 10 ms thành 49 lần ghi 0.2 ms.

5.3.3. Kết luận

3.1. Sự Thất bại của Cache (Sequential Workload)

- Workload Sequential là điểm hiệu suất kém nhất cho cả hai chính sách. Thời gian 1281 ms cho thấy rằng khi tính cục bộ (locality) bằng không (0%), hiệu năng của hệ thống bị giới hạn bởi tốc độ chậm nhất là HDD Read Latency (8.0 ms). Mọi lợi ích từ SSD Cache đều bị mất đi.

3.2. Lợi thế Cốt lõi của Write-Back (Write-Heavy Workload)

- Độ trễ Ghi (Write Latency): Đây là nơi WB thể hiện sức mạnh.
    - WT: Mỗi thao tác ghi (dù là Hit) đều phải chờ HDD 10 ms.
    - WB: Nhờ Dirty Bit, WB có thể thực hiện phần lớn thao tác ghi với tốc độ 0.2 ms (SSD), chỉ cần Flush 10 ms khi cần Evict block Dirty.

- Minh họa: Việc giảm 490 ms (tăng tốc 54.5%) cho Workload Write-Heavy chính là kết quả của việc tránh được 49 lần ghi 10 ms ngay lập tức (từ dữ liệu của biểu đồ HDD Write trước đó). Điều này chứng minh rằng WB là giải pháp tối ưu để loại bỏ Write Bottleneck trong hệ thống lưu trữ kết hợp.

3.3. Hiệu suất Kết hợp (Random & Locality)

- Sự cải thiện 9.9% và 15.4% cho Random và Locality cho thấy ngay cả khi không phải là Write-Heavy, WB vẫn mang lại lợi ích rõ rệt nhờ khả năng gộp ghi trung bình và cho phép Write Hit được hoàn thành gần như ngay lập tức.
- Locality đạt được hiệu suất cao nhất (441 ms) vì nó kết hợp được Hit Rate cao nhất (giảm chi phí Read) và Gộp ghi (giảm chi phí Write).

5.3.4. Kết luận

Biểu đồ chứng minh một cách mạnh mẽ sự cân bằng giữa hiệu suất và an toàn trong thiết kế hệ thống cache:

1. Write-Back (WB) là lựa chọn hiệu năng vượt trội: WB mang lại hiệu suất cao hơn trong mọi Workload có Hit Rate dương, đặc biệt là tăng tốc 54.5% cho Workload Write-Heavy.
2. Write-Through (WT) là lựa chọn an toàn: Mặc dù chậm hơn, WT đảm bảo tính nhất quán và an toàn dữ liệu tuyệt đối.
3. Tốc độ SSD không đảm bảo hiệu suất: Nếu Workload không có tính cục bộ (Sequential), tốc độ xử lý vẫn bị giới hạn bởi thiết bị chậm nhất (HDD).

5.4 Nhận xét

5.4.1. Nhận xét về Hiệu năng (Performance)

Kết luận: Write-Back vượt trội hoàn toàn về mặt hiệu năng nhờ khả năng giảm đáng kể chi phí cho các thao tác ghi.

5.4.1.1. Tổng thời gian xử lý (Biểu đồ 5.2c)

- **Tăng tốc độ:** Write-Back (WB) nhanh hơn Write-Through (WT) trung bình 18.5% trên 4 Workload.
- **Cải thiện Tối đa:** Trong Workload **Write-Heavy** (81 thao tác ghi), WB chỉ mất 409 ms so với 899 ms của WT. Tức là, WB tăng tốc độ 54.5%.
- **Điểm giới hạn:** Trong Workload **Sequential**, cả hai chính sách đều mất 1281 ms. Điều này chứng minh rằng khi **Hit Rate bằng 0%**, hiệu năng bị giới hạn bởi **HDD Read Latency** (8.0 ms/lần), làm lu mờ mọi lợi ích của chính sách ghi.

5.4.1.2. Giảm tải Ghi HDD (Biểu đồ 5.2b)

- **Giảm tải Ghi:** WB giảm 32.3% tổng số lần ghi HDD (201 lần của WT xuống còn 136 lần của WB).
- **Hiệu quả Gộp ghi:** Trong Workload Write-Heavy, WT phải thực hiện 81 lần ghi HDD. WB chỉ thực hiện 32 lần ghi HDD, tương đương giảm 60.5%. Việc này là do cơ chế **Dirty Bit** cho phép **Gộp ghi (Write Coalescing)**: nhiều lần cập nhật dữ liệu trên SSD (0.2 ms) chỉ cần một lần Flush xuống HDD (10 ms).

⇒ WB chuyển chi phí đắt đỏ (10 ms/lần ghi) thành chi phí rẻ (0.2 ms/lần ghi) và giảm tần suất sử dụng HDD, từ đó giảm đáng kể độ trễ phản hồi ứng dụng.

5.4.2. Nhận xét về An toàn Dữ liệu và Tính nhất quán

Kết luận: Write-Through an toàn dữ liệu hơn vì nó đảm bảo tính nhất quán mạnh mẽ.

Tiêu chí	Write-Through (WT)	Write-Back (WB)	Căn cứ số liệu
Tính nhất quán	**Hoàn hảo (Strong Consistency).** Dữ liệu luôn đồng bộ.	**Tạm thời (Eventual Consistency).** Dữ liệu có thể Dirty.	Dữ liệu Write-Heavy cho thấy 49 lần ghi bị trì hoãn (Dirty).
An toàn Dữ liệu	**Rủi ro 0%.** Dữ liệu được ghi bền vững ngay lập tức.	**Rủi ro cao.** Dữ liệu Dirty sẽ bị mất nếu mất điện đột ngột.	Cần cơ chế phục hồi phức tạp hơn (Journaling/UPS).

⇒ WT ưu tiên **toàn vẹn dữ liệu** ở mọi thời điểm. WB đánh đổi rủi ro mất dữ liệu chưa đồng bộ để đạt được tốc độ.

5.4.3. Nhận xét Tổng kết và Nguyên tắc Lựa chọn

Lựa chọn chính sách phải căn cứ vào yêu cầu chính của hệ thống, dựa trên phân tích hiệu năng:

Yêu cầu Chính	Lựa chọn Chính sách	Bằng chứng Số liệu
Ưu tiên Tốc độ	**WRITE-BACK**	WB nhanh hơn 54.5% cho Workload Write-Heavy

		(nơi tốc độ ghi là quan trọng nhất).
Ưu tiên An toàn	**WRITE-THROUGH**	WT đảm bảo tính nhất quán (không có rủi ro mất 49 lần cập nhật dữ liệu).
Mô hình I/O	**WRITE-BACK**	WB hiệu quả hơn trong việc xử lý **tính cục bộ** và **Write-Heavy**.

- **Sử dụng WB** khi Workload có khối lượng ghi lớn hoặc tính cục bộ cao và hệ thống có các cơ chế đảm bảo an toàn dữ liệu (như UPS).
- **Sử dụng WT** khi hệ thống không thể chấp nhận bất kỳ rủi ro mất dữ liệu nào, bất chấp sự suy giảm hiệu năng.