# Hệ thống lưu trữ hỗ trợ SQLite trên bộ nhớ NAND Flash

Hồ Văn Phi, Nguyễn Phương Tâm, Nguyễn Thị Hạnh, Lương Khánh Tý
Đại học Công nghệ Thông tin và Truyền thông Việt-Hàn, Đà Nẵng, Việt Nam
Tác giả liên hệ: Hồ Văn Phi, email: hvphi@vku@udn.vn
Ngày nhận bài: 28/12/2020, ngày sửa chữa: 03/04/2021, ngày duyệt đăng: 13/04/2021
Định danh DOI: 10.32913/mic-ict-research-vn.v2021.n1.938

**Tóm tắt:** Trong những năm gần đây, các thiết bị di động ngày càng phổ biến. Các thiết bị này thường sử dụng SQLite để làm cơ sở dữ liệu và bộ nhớ NAND Flash để lưu trữ dữ liệu. SQLite là một hệ cơ sở dữ liệu gọn nhẹ nhưng mạnh mẽ. SQLite đáp ứng đầy đủ mọi yêu cầu cơ bản của một hệ quản trị cơ sở dữ liệu. Bộ nhớ NAND Flash ngày càng phổ biến nhờ vào những ưu điểm nổi bật như tốc độ truy xuất nhanh, tiêu thụ điện năng ít, không mất dữ liệu khi mất nguồn. Do đó, việc sử dụng SQLite và NAND Flash là sự lựa chọn tốt cho thiết bị Android và iOS. Tuy nhiên, NAND Flash cũng có những nhược điểm cần khắc phục như không thể ghi đè, vòng đời bị giới hạn bởi số lần xoá trên các khối nhớ. Bên cạnh đó, cơ chế ghi tạm file journal mỗi lần commit của SQLite cũng làm ảnh hưởng đến bộ nhớ Flash. Trước khi thực hiện cập nhật dữ liệu vào cơ sở dữ liệu, SQLite tạo một file tạm gọi là journal chứa dữ liệu cập nhật trên bộ nhớ Flash. Việc này dẫn đến rất nhiều thao tác đọc/ghi trên bộ nhớ Flash làm cho hiệu suất của hệ thống giảm đáng kể. Để giải quyết vấn đề này, chúng tôi đề xuất một hệ thống bộ nhớ lai sử dụng FRAM cho hệ thống gọi là COSS(Commit-Optimized Scheme for SQLite). COSS có thể giảm bớt số lượng lớn thao tác đọc/ghi lên NAND Flash và thời gian thực hiện của cả hệ thống nhờ vào khả năng ghi đè và tốc độ cao của FRAM. Kết quả thực nghiệm cho thấy COSS đạt hiệu suất cao hơn so với hệ thống nguyên thuỷ thông thường.

**Từ khóa:** *SQLite, bộ nhớ Flash, FRAM, cơ sở dữ liệu điện thoại thông minh, thiết bị di động.*

---

**Title:** **A Design Method of Computational Semantics of Linguistic Words for Fuzzy Rule-based Classifier**

**Abstract:** Recently, the mobile devices are popular day by day. These devices use SQLite as a database management system and NAND Flash memeory as a storage medium. SQLite is a lightweight and powerful database engine. It provides all basic features of a database management system. NAND Flash memeories are popular because of the following advantages: fast access speed, nonvolatile, low power consumption. Thus, the combination of NAND Flash memory and SQLite is a good choice for mobile devices. However, NAND Flash memories also have some drawbacks that should be limited such erase-before-write characteristic, the lifecycle is limited by the number of block erasing. Beside that, the performance of NAND Flash is downgraded by temporarily writing journal files mechanism of SQLite. Whenever updating data, SQLite has to create and write the back up data to a file called Journal file. This causes a lot of Flash memory operations leading to reduction of overall performance of the system and shorten the lifecycle of NAND Flash. To address this problem, this study introduces a novel commit policy and hybrid storage system for SQLite based on a Flash memory and an FRAM, called COSS. COSS can minimize the overhead by deploying the hybrid storage of FRAM and NAND Flash memory. The experimental results show that COSS yields a good performance.

**Keywords:** *SQLite, Flash memory, FRAM, smart phone database, mobile devices.*

---

## I. GIỚI THIỆU

Bộ nhớ Flash [1, 4-6] được sử dụng rất phổ biến hiện nay nhờ vào những ưu điểm nổi bật của chúng như tốc độ cao, tiêu thụ ít điện năng, kích thước nhỏ gọn và độ an toàn dữ liệu cao. Tuy nhiên, bên cạnh những điểm mạnh đó, bộ nhớ Flash vẫn có những điểm yếu cần lưu tâm đó là đặc tính xóa trước khi ghi – không thể ghi đè và vòng đời hữu hạn (khoảng từ 10.000 đến 100.000 lần xóa trên mỗi block). Bộ nhớ Flash có cấu tạo và cơ chế hoạt động hoàn toàn khác với đĩa cứng (HDD-Hard Disk Drive) thông thường. Do đó để giúp các hệ điều hành truy cập bộ nhớ Flash một cách tương tự như HDD, một phần mềm trung gian có tên là FTL (Flash Translation Layer)[1] được sử dụng để chuyển đổi địa chỉ logic sang địa chỉ vật lý ánh xạ giữa máy tính và bộ nhớ Flash.

SQLite là một hệ cơ sở dữ liệu được sử dụng phổ biến

trên các thiết bị di động sử dụng hệ điều hành Android và iOS. SQLite không cần máy chủ, không cần cấu hình, khép kín và nhỏ gọn. SQLite engine không phải là một quy trình độc lập như các cơ sở dữ liệu khác, chúng có thể liên kết một cách tĩnh hoặc động tùy theo yêu cầu của ứng dụng. SQLite truy cập trực tiếp các file lưu trữ của nó trên bộ nhớ do đó. Mặc dù hiệu suất của các thiết bị di động tăng đáng kể nhờ vào việc sử dụng SQLite và bộ nhớ Flash. Tuy nhiên việc ghi tạm file journal trên bộ nhớ Flash làm giảm đáng kể hiệu suất hoạt động do một số lượng lớn các thao tác (read/write/erase) trên bộ nhớ Flash được thực hiện mỗi khi cập nhật dữ liệu trên SQLite. Đây là một vấn đề cần được giải quyết để nâng cao hiệu suất của các hệ thống Android và iOS. Bài báo này giới một hệ thống lưu trữ kết hợp cho các ứng dụng SQLite sử dụng Flash và FRAM (Ferroelectric Random Access Memory)[2, 3, 7] gọi là COSS nhằm tối ưu hoá thao tác Commit cho các hệ thống sử dụng NAND Flash và SQLite. Mục tiêu của hệ thống này là làm giảm số lượng số lượng thao tác trên bộ nhớ Flash và làm chậm quá trình lão hóa của bộ nhớ Flash, đồng thời hệ thống này cũng giúp dữ liệu của SQLite không bị mất khi mất điện hoặc hệ thống bị treo. FRAM đóng vai trò là một bộ đệm nơi lưu trữ tạm thời các file journal của SQLite. Khi bộ nhớ FRAM đầy, tất cả các file journal trên FRAM sẽ được ghi vào bộ nhớ Flash. Khả năng ghi đè dữ liệu và ghi từng byte của FRAM giúp các ứng dụng giảm chi phí thực hiện và số lượng thao tác phụ phát sinh trên bộ nhớ Flash do các đặc tính vật lý của Flash. Kết quả thử nghiệm cho thấy hệ thống này đạt hiệu quả tốt và đảm bảo dữ liệu của SQlite luôn được an toàn. Phần còn lại của bài bào có cấu trúc như sau: Phần II giới thiệu các kiến thức căn bản và liên quan đến SQLite, Flash và FRAM. Phần III trình bày kiến trúc của hệ thống. Kết quả thực nghiệm và đánh giá hệ thống được trình bày trong Phần IV và cuối cùng, kết luận bài báo được trình bày trong Phần V.

## II. KIẾN THỨC CƠ SỞ

Flash là một loại thiết bị bộ nhớ thứ cấp được sử dụng rất phổ biến hiện nay. Chúng có mặt trong các thiết bị điện tử như máy tính, điện thoại thông minh, các thiết bị nhúng, thẻ nhớ, USB, SSD,... Khác với đĩa cứng thông thường, bộ nhớ Flash gồm một dãy các thẻ nhớ NAND Flash. Bộ nhớ Flash được tổ chức thành các khối nhớ; mỗi khối nhớ chứa một số lượng cụ thể các trang nhớ (32, 64, 128 trang). Trang nhớ là đơn vị nhỏ nhất của thao tác đọc và ghi trong khi đó khối nhớ là đơn vị nhỏ nhất của thao tác xóa. Bộ nhớ Flash hỗ trợ 3 thao tác cơ bản là đọc, ghi và xóa [8-9]. Đọc là thao tác có tốc độ nhanh nhất tiếp đến là thao tác ghi. Một thao tác ghi mất khoảng 2ms; chậm hơn 10 lần so với thao tác đọc. Chậm nhất là thao tác xóa, chậm hơn 10 lần so với ghi. Hiện nay, bộ nhớ Flash có vòng đời khoảng 10.000-100.000 lần xóa khối. Nếu số lần xóa của một khối vượt quá ngưỡng này thì khối đó không còn khả năng sử dụng. Để phù hợp với các hệ điều hành khác nhau và đạt hiệu quả tối ưu, Flash cần phần mềm trung gian FTL. FTL có 3 chức năng chính đó là: ánh xạ địa chỉ (Address Mapping), thu hồi khối nhớ không hợp lệ và điều phối ghi dữ liệu trên các khối nhớ (Wear Leveling). Chức năng ánh xạ địa chỉ cung cấp các thuật toán ánh xạ giữa địa chỉ logic và địa chỉ vật lý trên bộ nhớ Flash. Đã có rất nhiều thuật toán ánh xạ địa chỉ [1, 4-6] được giới thiệu. Chúng có thể được gom thành 3 nhóm chính: Ánh xạ sector (Sector Mapping), ánh xạ khối (Block Mapping) và ánh xạ lai (Hybrid Mapping). Bộ thu hồi khối nhớ không hợp lệ có chức năng thu hồi các khối nhớ không hợp lệ để tái sử dụng. Khi số lượng khối nhớ hợp lệ trên bộ nhớ Flash thấp hơn ngưỡng cho phép, bộ thu hồi khối nhớ không hợp lệ sẽ được tự động kích hoạt để thu hồi và tái sử dụng các khối không hợp lệ. Wear Leveling được sử dụng để điều tiết các khối nhớ để đảm bảo các khối nhớ có tần suất được sử dụng là tương đương nhau. Wear leveling sẽ quyết định khối nhớ nào được sử dụng khi có yêu cầu ghi dữ liệu vào bộ nhớ Flash. FRAM là một công nghệ bộ nhớ mới sử dụng tính chất điện sắt. FRAM có các tính chất tương tự như DRAM nhưng dữ liệu lưu trên FRAM không bị mất khi không có nguồn cung cấp điện. Khác với Flash, FRAM cho phép truy xuất ngẫu nhiên đến từng đơn vị bit. Thuật ngữ “ferroelectric - điện sắt” không có nghĩa là FRAM chứa các phần tử sắt cũng không có nghĩa là FRAM bị ảnh hưởng bởi từ trường. FRAM cho phép phân vùng một cách linh hoạt cho cả mục đích chứa mã thực thi (Execution Code Partition) và dữ liệu. Có nghĩa là FRAM có thể được sử dụng để lưu trữ mã thực thi chương trình và dữ liệu đồng thời mà với các loại bộ nhớ khác phải sử dụng 2 loại riêng biệt: ROM để lưu mã chương trình và bộ nhớ ngoài lưu trữ dữ liệu. SQLite là một hệ quản trị cơ sở dữ liệu được sử dụng rộng rãi trong các thiết bị di động nhờ vào tính nhỏ gọn và mạnh mẽ của nó. Tuy nhiên, tần suất cập nhật và ghi dữ liệu ngẫu nhiên trên SQLite rất lớn nên hiệu quả của SQLite trên bộ nhớ Flash sẽ giảm đáng kể. Mỗi khi cập nhật dữ liệu, SQLite tạo một file gọi là file journal để lưu trữ tạm thời dữ liệu cập nhật. Sau khi thao tác cập nhật đã được commit, file journal sẽ bị xóa. Quá trình thực hiện atomic commit trong SQLite được thực hiện như sau:

1. Chiếm giữ khóa:

Trước khi SQLite có thể ghi dữ liệu vào cơ sở dữ liệu, nó phải nắm giữ khóa đọc (read lock) để đảm bảo cơ sở dữ liệu đã sẵn sàng. Sau đó sẽ chiếm giữ khóa chia sẻ (shared lock) để ngăn không cho giao dịch khác ghi vào mục dữ liệu hiện hành.

2. Đọc thông tin từ cơ sở dữ liệu:

Những dữ liệu cần thay đổi được đọc từ cơ sở dữ liệu.

3. Chiếm giữ khóa dành riêng (Reserved Lock)
Mục đích của việc chiếm giữ khóa reserved là nhằm ngăn chặn các giao dịch khác ghi dữ liệu và giao dịch hiện hành sẽ sớm ghi dữ liệu vào mục dữ liệu hiện hành.

4. Tạo file Journal:
Trước khi thực hiện thay đổi dữ liệu, SQLite tạo và ghi dữ liệu cũ vào file journal. Lúc này journal file được lưu tạm thời ở bộ đệm.

5. Thay đổi dữ liệu ở phía người dùng:
Sau khi dữ liệu cũ được khi vào file journal, dữ liệu được cập nhật ở phía người dùng.

6. Ghi journal file vào đĩa:
Để đảm bảo an toàn dữ liệu, file journal được ghi vào đĩa cứng.

7. Chiếm giữ khóa độc quyền (Exclusive Lock)
Trước khi ghi dữ liệu, SQLite cần chiếm giữ khóa độc quyền để đảm bảo không có giao dịch nào đang đọc hay ghi dữ liệu lên mục dữ liệu hiện hành.

8. Ghi dữ liệu thay đổi vào đĩa:
Toàn bộ dữ liệu thay đổi sẽ được ghi vào đĩa.

9. Xóa file journal:
Sau khi hoàn tất ghi dữ liệu vào file, hệ thống sẽ xóa file journal. Sau khi journal file đã bị xóa thành công, các khóa sẽ được giải phóng.

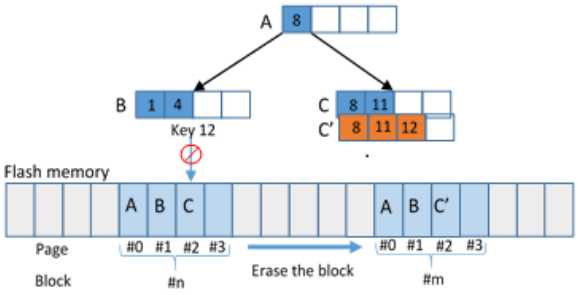
Nếu quá trình commit không thành công, hệ thống sẽ thực hiện rollback bằng cách đọc file journal vào ghi dữ liệu cũ vào cơ sở dữ liệu. Quá trình rollback bao gồm các bước như sau:

1. Tìm file journal "nóng". File journal "nóng" là file journal có các đặc điểm sau:
- Đang tồn tại hợp lệ
- Không rỗng
- Không chứa khóa reserved trên file cơ sở dữ liệu chính
- Tiêu đề đúng cấu trúc
- Không chứa tên file supper-journal (dùng trong commit nhiều giao dịch đồng thời).

2. Đọc thông dữ liệu cũ từ file journal và ghi vào cơ sở dữ liệu.

3. Xóa file journal "nóng": Sau khi tất cả mọi thông tin được ghi vào cơ sở dữ liệu, file journal cần được xóa để hoàn tất giao dịch.

Như đã đề cập ở trên, bên cạnh những điểm mạnh nổi bật, bộ nhớ Flash có hai nhược điểm lớn đó là đặc tính xóa trước khi ghi và hạn chế số lần xóa khối. Do đó, thực hiện thao tác commit của SQLite nguyên thủy trên bộ nhớ Flash sẽ dẫn đến hiệu quả hoạt động thấp. Hình 1 trình bày một ví dụ về việc cập nhật dữ liệu trên bộ nhớ Flash sử dụng thuật toán ánh xạ địa chỉ Block mapping.




Hình 1. Ví dụ cập nhật dữ liệu trên bộ nhớ Flash

Giả sử cần cập nhật trên cây chỉ mục trong cơ sở dữ liệu SQLite được lưu trữ trên bộ nhớ Flash. Nếu một bản ghi với giá trị khóa 12 được chèn vào cơ sỡ dữ liệu thì bản ghi đó được cập nhật trên nút C và ghi dữ liệu trên trang nhớ của bộ nhớ Flash. Vì đặc tính xóa trước khi ghi (không thể ghi đè) nên quá trình chèn 12 được thực hiện như sau: Đầu tiên các file hợp lệ (A, B) trong block #n được sao chép sang block #m; ghi file C' vào block #m và cuối cùng block #n sẽ bị xóa hoặc đánh dấu block không hợp lệ. Quá trình này phát sinh rất nhiều các thao tác trên Flash dẫn đến giảm hiệu suất của toàn hệ thống. Lưu ý rằng, vòng đời của NAND Flash bị giới hạn bởi số lần xoá trên các block. Do đó, việc giảm thao tác xoá giúp cho tuổi thọ của NAND Flash ít bị ảnh hưởng. Nhằm hạn chế việc phát sinh các thao tác như đã trình bày ở ví dụ trên, một giải pháp mới được đề xuất ở Phần III.

## III. THIẾT KẾ VÀ CÀI ĐẶT COSS

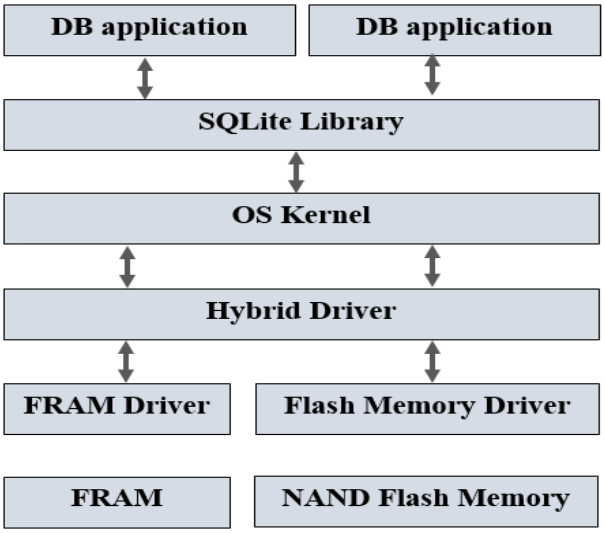
### 1. Thiết kế COSS

Phần này sẽ trình bày một cơ chế tối ưu hóa thao tác commit trên SQLite và hệ thống lưu trữ cho SQLite trên nền tảng NAND Flash và bộ nhớ FRAM. Mục tiêu của hệ thống này là nhằm giảm tối đa chi phí thực hiện cập nhật file journal mỗi khi cập nhật dữ liệu trong cơ sở dữ liệu SQLite và giảm số lượng thao tác trên NAND Flash. Để đạt được mục tiêu này, chúng tôi xây dựng một hệ thống lưu trữ lai sử dụng FRAM và NAND Flash. Trong hệ thống này, FRAM được dùng để lưu trữ file journal và NAND Flash được dùng để lưu trữ dữ liêu.

Hình 2 trình bày kiến trúc tổng thể hệ thống COSS. Hệ thống COSS bao gồm tất cả các thành phần của một hệ thống thông thường, driver cho hệ thống lai và các thành phần lưu trữ FRAM và NAND Flash.

Thành phần driver cho hệ thống lai sẽ quản lý và quyết định thiết bị lưu trữ nào (FRAM hay Flash) sẽ được sử dụng để ghi dữ liệu. Nếu file journal được tạo ra hoặc được cập nhật thì driver hệ thống lai sẽ ghi dữ liệu file journal lên FRAM. Ngược lại, dữ liệu của SQLite sẽ được ghi vào Flash. FRAM driver và Flash driver sẽ quản lý việc ghi dữ liệu lên FRAM và Flash tương ứng theo cơ chế của nó.

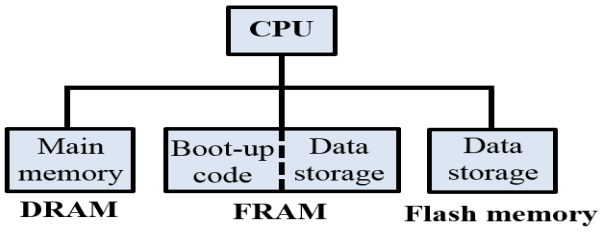
FRAM được sử dụng kết hợp với bộ nhớ Flash nhằm nâng cao hiệu suất của SQLite trên các hệ thống Android và iOS. Bởi vì dữ liệu có thể được ghi đè trên FRAM nên có thể tránh được nhược điểm xóa trước khi ghi của Flash. Mỗi khi file journal được tạo ra, COSS sẽ ghi file journal vào FRAM thay vì ghi vào Flash như bình thường. Điều này làm giảm đáng kể số lượng thao tác trên bộ nhớ Flash, đặc biệt là thao tác ghi và xóa trên Flash tiêu tốn nhiều thời gian và năng lượng.




Hình 2. Kiến trúc của COSS

Bằng cách sử dụng hệ thống lưu trữ lai này, COSS làm tăng đáng kể hiệu suất của SQLite trên các hệ thống Android/iOS của các thiết bị thông minh đồng thời làm chậm quá trình lão hoá của NAND Flash.

## 2. Cơ chế hoạt động của COSS

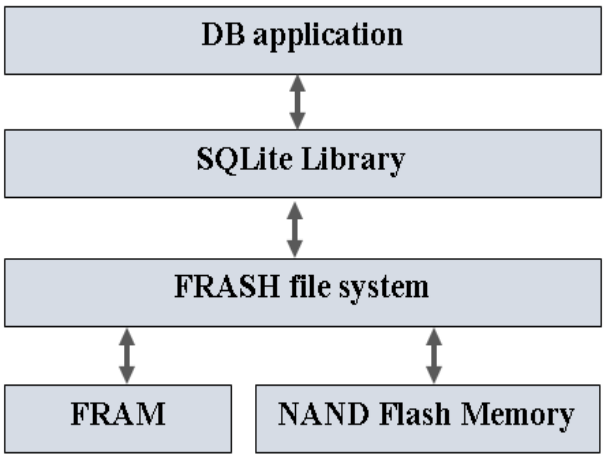



Hình 3. Hệ thống bộ nhớ kết hợp

Với hệ thống này, chúng tôi kết hợp một FRAM và một NAND Flash để lưu trữ dữ liệu. FRAM đã được sử dụng như một thiết bị lưu trữ trong các hệ thống nhúng. FRAM cho phép truy xuất từng byte và ghi đè dữ liệu do đó nó có thể hạn chế nhược điểm xóa trước khi ghi của Flash.

Như trình bày trong Hình 3, hệ thống sử dụng một FRAM cho hai mục đích là lưu trữ dữ liệu và mã thực thi chương trình. Trong trường hợp này hệ thống vẫn hoạt động tốt vì FRAM có thể đóng vai trò như cả ROM và thiết bị lưu trữ. Bằng cách sử dụng một phần của FRAM để lưu dữ liệu, hệ thống có thể lưu file journal tạm thời vào FRAM và cơ sở dữ liệu vào Flash. Với cơ chế này, hệ thống sẽ hoạt động với hiệu suất tốt hơn các hệ thống thông thường bởi vì FRAM nhanh hơn Flash, hạn chế các thao tác xóa trước khi ghi đồng thời giúp kéo dài tuổi thọ của Flash.

Để cài đặt hệ thống lưu trữ này, hệ thống cần một phương thức riêng để truy xuất cả Flash và FRAM. Như trình bày trong Hình 4, hệ thống sử dụng FRASH[10] là một hệ thống file đơn giản được thiết kế cho hệ thống lưu trữ kết hợp FRAM và Flash.




Hình 4. Hệ thống file

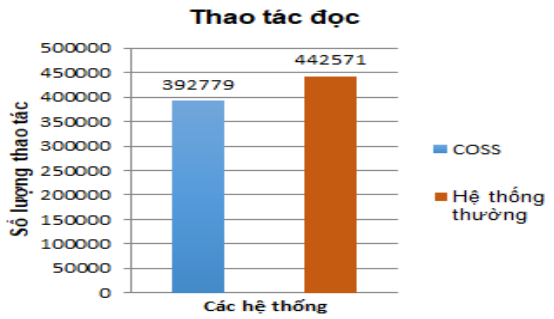
Về cơ bản, khi thực hiện quá trình commit, COSS ghi các file journal vào FRAM. Ngay khi dữ liệu được cập nhật và ghi vào Flash thành công thì các file journal trong phân vùng dữ liệu của FRAM được xóa để hoàn tất quá trình commit.

## IV. ĐÁNH GIÁ HIỆU SUẤT

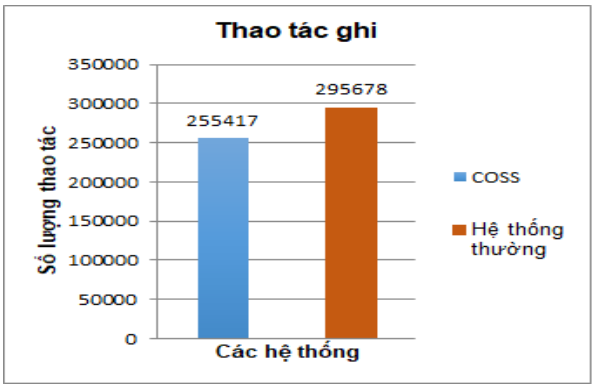
Phần này trình bày một số kết quả thực nghiệm của COSS và so sánh hiệu suất với hệ thống nguyên thuỷ. Tất cả các thực hiện được triển khai trên bộ nhớ FRAM giả lập được lấy từ RAM có dung lượng 8MB. Hiệu suất của các hệ thống được đánh giá dựa trên số lượng thao tác và hiệu suất sử dụng không gian nhớ trong các khối nhớ của NAND Flash. Trong các thử nghiệm, chúng tôi sử dụng bộ đếm thao tác và bộ đếm thời gian hệ thống của bộ nhớ Flash để đánh giá kết quả thực hiện. Để đảm bảo công bằng khi so sánh, chúng tôi thử nghiệm tất cả thực nghiệm trên cùng một môi trường là Windows 10 Pro N với vi xử lý Intel Core i5-7400, 8GB DDR và đĩa cứng SSD 512GB.

## 1. Hiệu suất đọc/ghi

Trong phần này, chúng tôi đánh giá hiệu quả của các hệ thống thông qua việc cập nhật 100.000 bản ghi trong cơ sở dữ liệu.




Hình 5. Số lượng thao tác đọc



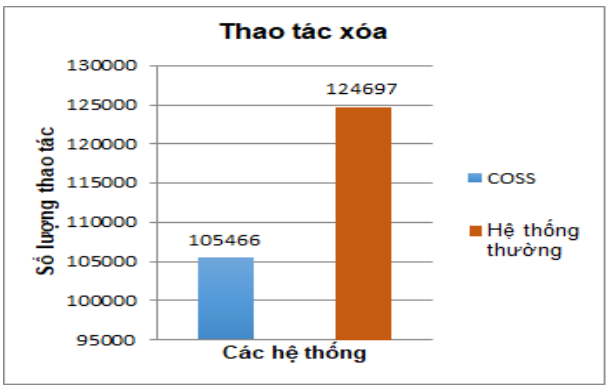

Hình 6. Số lượng thao tác ghi

Như Hình 5 đã thể hiện, số lượng thao tác đọc của COSS ít hơn khoảng 14,1% so với hệ thống nguyên thuỷ bởi vì COSS sử dụng FRAM có khả năng ghi đè. Nhờ vào khả năng ghi đè của FRAM, COSS tránh được số lượng lớn thao tác ghép bộ nhớ Flash dẫn đến một số lượng thao tác đọc được giảm bớt. Đối với thao tác ghi và thao tác xoá, COSS thực hiện ít hơn hệ thống nguyên thuỷ khoảng 17,9% và 20,5% như trong Hình 6 và Hình 7 sau đây.

Mặc dù COSS cần thêm một số thao tác ghi để ghi dữ liệu vào FRAM trước khi ghi vào Flash nhưng COSS không thực hiện các thao tác ghép trên Flash làm giảm đáng kể số lượng thao tác ghi và xoá. Điều này rất có ích cho bộ nhớ Flash bởi vì nó có thể kèo dài vòng đời của bộ nhớ Flash. Vòng đời của bộ nhớ Flash được tính dựa trên số lần xoá các khối nhớ. Do đó, việc giảm số lượng thao tác xoá trên các khối sẽ làm chậm lại quá trình lão hoá của bộ nhớ Flash.

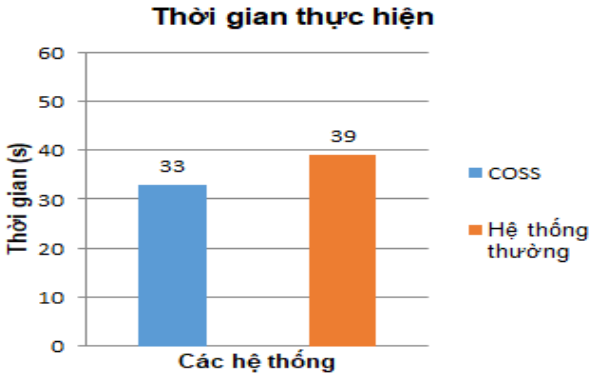
## 2. Thời gian thực hiện

Hình 8 trình bày thời gian thực hiện 100.000 thao tác cập nhật vào cơ sở dữ liệu. Bởi vì COSS giảm một số lượng lớn các thao tác trên Flash và tốc độ cao của FRAM nên COSS thực hiện nhanh hơn rất nhiều so với hệ thống nguyên thủy. Tốc độ truy xuất của FRAM tương tự như DRAM, nhanh hơn nhiều so với Flash. Mặt khác, không có thao tác ghép xảy ra trong FRAM - một thao tác mà tốn rất nhiều thời gian trên Flash. Điều này giúp cho COSS thực hiện cập nhật nhanh hơn khoảng 218% so với hệ thống nguyên thủy.



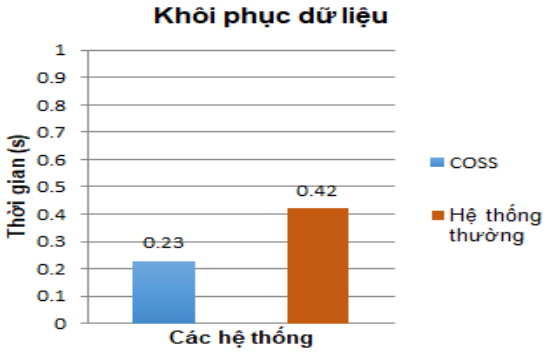

Hình 7. Số lượng thao tác xoá

## 3. Khôi phục dữ liệu




Hình 8. Thời gian thực hiện

Như đã trình bày ở phần trước, khả năng khôi phục dữ liệu là một điểm mạnh của FFB. Dữ liệu của COSS luôn được đảm bảo và khả năng khôi phục dữ liệu sau sự số rất nhanh. Hình 9 trình bày thời gian khôi phục dữ liệu sau sự cố. Nhìn chung, COSS khôi phục dữ liệu trong thời gian trung bình khoảng 0,23 giây; nhanh hơn hệ thống bình thường khoảng 82,6%. Bởi vì tất cả dữ liệu của COSS đều được ghi vào Flash hoặc FRAM (không mất dữ liệu) do đó dữ liệu có thể được khôi phục một cách dễ dàng bằng cách đọc từ FRAM và ghi vào Flash sau khi sự cố xảy ra.




Hình 9. Thời gian khôi phục dữ liệu

Thực tế, thời gian khôi phục dữ liệu phụ thuộc vào kích thước vùng nhớ của FRAM dùng để làm bộ đệm. Kích

thước vùng này càng lớn thì thời gian khôi phục càng lâu vì có nhiều dữ liệu lưu trữ tạm thời trong đó. Tuy nhiên, tốc độ truy xuất của FRAM khá cao nên thời gian khôi phục dữ liệu là chấp nhận được.

## V. KẾT LUẬN

Bằng cách kết hợp 2 thiết bị lưu trữ để tạo thành một hệ thống lưu trữ kết hợp, COSS có thể nâng cao hiệu suất của SQLite và bộ nhớ NAND Flash trong các hệ thống di động. Trong hệ thống COSS này, với khả năng ghi đè từng byte và tốc độ cao của FRAM đã khắc phục được nhược điểm vật lý của NAND Flash đó là ghi từng trang (page) và xóa khối (block) trước khi ghi (tính chất làm ảnh hưởng xấu đến tốc độ và vòng đời của Flash). COSS không những giúp các hệ thống của sử dụng SQLite nâng cao hiệu suất mà còn đảm bảo dữ liệu luôn được an toàn. Tuy nhiên, một điểm hạn chế của COSS đó là phải sử dụng thêm một FRAM (một thiết bị lưu trữ mới chưa được sử dụng phổ biến và còn hạn chế về dung lượng).

Do đó, COSS có thể được áp dụng trong các ứng dụng mà ở đó yêu cầu về mặt tốc độ và an toàn dữ liệu đặt lên hàng đầu.

Trong tương lai gần, chúng tôi sẽ tập trung nghiên cứu xây dựng hệ thống drive cho COSS để nâng cao hiệu quả hoạt động của hệ thống và đô an toàn của dữ liệu.

## TÀI LIỆU THAM KHẢO


[1] Shinde Pratibha et al. "Efficient Flash Translation layer for Flash Memory," *International Journal of Scientific and Research Publications*, Volume 3, Issue 4, April 2013

[2] Hồ Văn Phi "FFB: Hệ thống lưu trữ kết hợp cho các ứng dụng B-tree trên bộ nhớ NAND Flash". *Hội thảo quốc gia lần thứ XXI: Một số vấn đề chọn lọc của Công nghệ thông tin và truyền thông – Thanh Hóa*, 7/2018, Trang 97-102.

[3] VanPhi Ho, Park, Dong-Joo. "An Efficient B-tree Index Scheme for Flash Memory". *International Journal of Software Engineering and Its Applications*, Vol.11, No.2 pp. 51-64, (2017)

[4] VanPhi Ho, Park, Dong-Joo, "WPCB-Tree: A Novel Flash-Aware B-Tree Index Using a Write Pattern Converter," *Symmetry* 2018, 10, 18.

[5] Chin-Hsien Wu et al. "An Efficient B-Tree Layer Implementation for Flash Memory Storage Systems," *ACM Transactions on Embedded Computing Systems*, Vol. 6, No. 3, Article 19, 2007.

[6] Xiaona Gong et al. "A Write-Optimized B-Tree Layer for NAND Flash," *Proceeding of the 7th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM)*, pp.1-4, 2011

[7] Volker Rzehak, "Low-Power FRAM Microcontrollers and Their Applications", *White Paper*, Texas Instruments Deutschland, June 2011.

[8] Yinan Li, Bingsheng He, Robin Jun Yang, Qiong Luo and Ke Yi. "Tree Indexing on Solid State Drives," *Proceedings of International Conference on Very Large Data Bases*, 2009.

[9] Hongchan Roh, Woo-Cheol Kim, Seungwoo Kim and Sanghyun Par.k "A B-Tree Index Extension to Enhance Response Time and The Life Cycle of Flash Memory," *Information Sciences*, vol. 179, no. 18, 2009, pp. 3136-3161.

[10] Kim, E.-K., Shin, H., Jeon, B.-G., Han, S., Jung, J., Won, Y. "FRASH: Hierarchical File System for FRAM and Flash". *ICCSA 2007*, Part I. LNCS, vol. 4705, pp. 238–251. Springer, Heidelberg (2007).

[11] Jung, Myoungsoo, Choi, Wonil, Shuwen Gao, Ellis Herbert Wilson III, David Donofrio, John Shalf and Mahmut Taylan Kandemir. "NANDFlashSim: High-Fidelity, Microarchitecture-Aware NAND Flash Memory Simulation". *ACM Transactions on Storage*, Vol.12, No.2, Article 6, 1.2016.


## SƠ LƯỢC VỀ TÁC GIẢ


**Hồ Văn Phi**

Sinh ngày: 06/06/1980. Nhận bằng Thạc sỹ Khoa học máy tính năm 2009 tại Đại học Đà Nẵng. Nhận bằng Tiến sỹ KH máy tính năm 2017 tại Trường Đại học Soongsil, Hàn Quốc. Hiện công tác tại Trường ĐH CNTT và Truyền thông Việt Hàn, Đại học Đà Nẵng. Lĩnh vực nghiên cứu: Cơ sở dữ liệu trên Flash.
Điện thoại: 0905702411
E-mail: hvphi@vku.udn.vn

**Nguyễn Phương Tâm**

Sinh ngày: 03/06/1980. Nhận bằng Thạc sỹ Khoa học máy tính năm 2009 tại Đại học Đà Nẵng. Nhận bằng Tiến sỹ KH máy tính năm 2017 tại Trường Đại học Soongsil, Hàn Quốc. Hiện công tác tại Trường ĐH CNTT và Truyền thông Việt Hàn, Đại học Đà Nẵng. Lĩnh vực nghiên cứu: Cơ sở dữ liệu trên Flash.
Điện thoại: 0905702411
E-mail: hvphi@vku.udn.vn

**Nguyễn Thị Hạnh**

Sinh ngày: 01/10/1981. Nhận bằng Thạc sỹ Khoa học máy tính năm 2010 tại Đại học Đà Nẵng. Hiện đang công tác tại Trường ĐH CNTT và Truyền thông Việt Hàn, Đại học Đà Nẵng. Lĩnh vực nghiên cứu: Bảo mật dữ liệu.
Điện thoại: 0935688515.
E-mail: hanhnt@vku.udn.vn

**Lương Khánh Tý**

Sinh ngày: 07/08/1984. Nhận bằng Thạc sỹ KH máy tính năm 2011 tại Đại học Đà Nẵng. Hiện đang công tác tại Trường ĐH CNTT và Truyền thông Việt Hàn, Đại học Đà Nẵng. Lĩnh vực nghiên cứu: Cơ sở dữ liệu.
Điện thoại: 0974883609
E-mail: lkty@vku.udn.vn