

2012 Hội thảo quốc tế IEEE 20 trên mô hình hóa, phân tích và mô phỏng của máy tính và viễn thông Hệ thống

Đề án lập bản đồ mức độ cho Flash Memory Devices

Young-Kyoon Suh, Bongki Moon, Alon Efrat

*Bộ môn Khoa học máy tính*

# Đại học Arizona ở Tucson, Arizona, Mỹ, 85.721

*{* yksuh, bkmoon, alon}@cs.arizona.edu

Jin-Soo Kim, Sang-Won Lee

# Trường Info. & Comm. Engr.

*Sungkyunkwan Đại học Suwon, 440-746, Korea*

*{* jinsookim, swlee}@skku.edu

***Tóm tắt:* Các thiết bị bộ nhớ Flash thường dựa trên các phương án mapping địa chỉ truyền thống như lập bản đồ trang ( page mapping ), lập bản đồ khối ( block mapping) hoặc kết hợp cả hai (hybrid mapping). Page Mapping thì linh hoạt hơn so với block mapping hoặc hybrid mapping mà không bị giới hạn bởi ranh giới khối. Tuy nhiên bảng mapping của nó có khuynh hướng phát triển nhanh chóng nhờ khả năng của các thiết bị bộ nhớ flash. Để khắc phục hạn chế này, chúng tôi đề xuất một chương trình mapping mới mà về cơ bản là khác nhau với các các chiến lược mapping hiện có. Chúng tôi gọi kế hoạch mới này *Virtual Extent Trie (VET),* như nó quản lý thông tin mapping bằng cách xử lý mỗi yêu cầu I/O như một *phạm vi* và bằng cách sử dụng phạm vi như các đơn vị mapping cơ bản chứ không phải là các page hoặc các block. Bằng cách lưu trữ phạm vi thay vì địa chỉ cá nhân, VET tiêu thụ ít bộ nhớ để lưu trữ thông tin mapping và vẫn linh hoạt như page mapping. Chúng tôi quan sát thấy trong các thí nghiệm của chúng tôi rằng VET giảm tiêu thụ bộ nhớ lên đến một bậc so với phương án mapping truyền thống cho một số khối lượng công việc thực tế. Phương án VET cũng mở rộng tốt với việc tăng không gian địa chỉ bằng khối lượng công việc tổng hợp. Với một cơ chế tìm kiếm nhị phân, VET hạn chế thời gian để lập bản đồ** O(log log(U) .**Ở đây U****là tập tất cả các địa chỉ logic có thể. Mặc dù chi phí về thời gian mapping tiệm cận của VET cao hơn** *O (* **1) của một chương trình lập bản đồ trang web, số lượng tăng lên trên cao hơn là gần như không đáng kể hoặc nó vẫn đủ thấp để được che giấu bằng một hoạt động I/O kèm theo .**

1. Giới thiệu

Ổ đĩa bộ nhớ thể rắn (Solid State Drives - SSD) ngày càng được sử dụng như là một giải pháp thay thế cho các ổ đĩa cứng truyền thống. Việc không có bộ phận chuyển động trong các thiết bị bộ nhớ flash đã làm giảm được độ trễ và điện năng tiêu thụ quá mức của nó, và cho phép thiết kế nhẹ, có khả năng chống sốc.

Vì bộ nhớ flash không cho phép bất kỳ dữ liệu được cập nhật tại chỗ, hầu hết các thiết bị bộ nhớ flash đi kèm với một lớp phần mềm được gọi là Flash Translation Layer (FTL) có trách nhiệm về logic tranh chấp của ổ đĩa trừu tượng thông thường(conventional disk abstraction) [1]. Nói chung, FTL hướng dẫn ghi một yêu cầu đến từ một máy chủ vào một vùng trống trong bộ nhớ flash, và quản lý thông tin mapping từ mộtđịa chỉ logic được công nhận bởi các máy chủ đến một địa chỉ *vật lý* trong một thiết bị bộ nhớ flash. Đối với FTL để thực hiện chuyển đổi địa chỉ logical-to-physical (L2P) này, một mapping table cần phải được duy trì trong một thiết bị bộ nhớ flash.

Các thiết bị flash bộ nhớ đương đại thường được dựa trên một chương trình mapping địa chỉ sử dụng một *page (* thường 4KB) hoặc một *block (* thường 256KB) là một đơn vị mapping. Trong page mapping [2], các chi tiết là một page, và tính linh hoạt là lợi thế quan trọng nhất kể từ khi địa chỉ logic của một page có thể được ánh xạ tới bất kỳ vị trí vật lý trên một thiết bị bộ nhớ flash. Như

công suất của các thiết bị bộ nhớ flash phát triển, tuy nhiên, page mapping đòi hỏi họ phải cung cấp dung lượng RAM lớn cho việc duy trì các bảng mapping lớn. Trong trường hợp bộ nhớ flash SSD 4TB, ví dụ, kích thước của bảng mapping của nó có thể được lớn như 4GB.

Trong block mapping [3], các chi tiết là một khối, và kích thước của một bảng mapping nhỏ hơn nhiều vì các chi tiết lớn hơn của mapping. Trong ví dụ trên, kích thước bảng sẽ chỉ nhỏ như 64MB. Tuy nhiên, block mapping có một thiếu sót quan trọng. Các vị trí vật lý của một trang logic là cố định đến một trang nào đó bù đắp trong một khối.Thập chí cập nhật một trang có thể cần toàn bộ khối chứa trang phải được sao chép vào khối rỗng. Điều này làm cho một đề án block mapping thuần không khả thi cho hầu hết khối lượng công việc thực tế.

Đề án Hybrid mapping ( *ví dụ,* Fmax [4], FAST [5], Superblock-FTL [6], LAST [7]) đã được đề xuất để tận dụng những thế mạnh có chiến lược của cả hai page và block mapping. Các lược đồ lai này sử dụng các khối bộ nhớ flash bổ sung như một không gian được cung cấp quá mức, nơi mà các trang được cập nhật gần đây được lưu trữ mà không bị giới hạn bởi các ranh giới khối, do đó các địa chỉ của các trang này được quản lý một cách linh động hơn bằng cách lập bản đồ trang. Chiến lược lai này giúp giảm kích thước của một table mapping, nhưng điều này chỉ khả thi với chi phí bộ nhớ over-provisioned flash (thường khoảng 30% khả năng sử dụng được) và tăng độ trễ lập bản đồ

Trong bài báo này, chúng tôi đề xuất một sơ đồ lập bản đồ mới mà về mặt cơ bản khác với các chiến lược lập bản đồ hiện có. Chúng tôi gọi mô hình này là Virtual Extent Trie (VET) vì nó quản lý thông tin lập bản đồ sao cho một yêu cầu I / O cho trước được xử lý như một phạm vi và phạm vi được sử dụng làm đơn vị lập bản đồ cơ bản. Bằng cách lưu trữ mức độ thay vì các địa chỉ riêng lẻ trong yêu cầu, VET tiêu tốn ít bộ nhớ hơn để lưu trữ thông tin lập bản đồ và vẫn linh hoạt như việc lập bản đồ trang. Ngoài ra, chương trình VET hoạt động không phụ thuộc vào kiến trúc flash bên dưới bằng một hoặc nhiều kênh [8], vì bất kỳ loại thông tin địa chỉ vật lý nào được trả về sau khi viết flash có thể được lưu trữ đơn giản bằng các mức mở rộng. Kết quả sơ bộ cho thấy VET có thể giảm mức sử dụng bộ nhớ lên đến một mức độ so với các sơ đồ lập bản đồ thông thường cho khối lượng công việc thực tế.

Với một cơ chế tìm kiếm nhị phân, VET giới hạn thời gian tìm kiếm ở mức O (log log | U |), trong đó U biểu thị tập hợp tất cả các địa chỉ hợp lý. Mặc dù chi phí lập bản đồ của VET ít hơn thời gian O (1) của bản dịch địa chỉ



1526-7539 / 12 $ 26,00 © 2012 IEEE DOI

10,1109 / MASCOTS.2012.45



331

bằng page mapping, nhưng chúng tôi hy vọng rằng số lượng chi phí tăng lên sẽ gần như không đáng kể hoặc đủ thấp để được ẩn đi bởi một hoạt động I/O kèm theo

Phần còn lại của bài viết này được tổ chức như sau. Phần II phác thảo thiết kế của chương trình VET. Phần III đề xuất các thuật toán của chương trình VET. Sau đó, Mục IV trình bày các kết quả đánh giá hiệu quả sử dụng thế giới thực và công việc tổng hợp. Phần V cung cấp công việc liên quan. Cuối cùng, mục VI kết luận bài báo này.

1. Nguyên lí thiết kế
2. *Dựa trên bản đồ mở rộng*

Trong hầu hết các đề án lập bản đồ địa chỉ truyền thống, một trang hay một khối được sử dụng như là các đơn vị lập bản đồ. Đối với yêu cầu cho đọc hoặc viết thư từ máy chủ, FTL là chịu trách nhiệm cho tất cả các dịch trang lôgíc (hoặc khối) mà các I / O yêu cầu muốn đọc hoặc ghi vào trang vật lý (hoặc khối) thông qua bảng vẽ bản đồ của nó. Do đó, FTLs phải biết trang logic (hoặc khối) được ánh xạ mà trang vật lý (hoặc khối) ở tất cả các lần bằng cách duy trì các thông tin bản đồ trong bảng vẽ bản đồ.

Đề án VET chúng tôi đề xuất trong bài báo này lợi dụng thực tế là một I

/ O yêu cầu từ máy bao gồm một địa chỉ bắt đầu hợp lý và số lượng của các thành phần để đọc hoặc viết. Vì vậy, mỗi yêu cầu I / O có thể được coi là một *mức độ*

de fi định nghĩa trong không gian địa chỉ logic và có thể được lưu trữ trong các bảng vẽ bản đồ như một đơn vị toàn bộ mà không bị phá vỡ cho nhiều trang hoặc các khối.

Khi một yêu cầu ghi mới đến, nó tạo ra thông tin bản đồ mới hoặc cập nhật hiện có. Speci fi biệt, nếu yêu cầu viết vào một khu vực hợp lý mà không bị chiếm đóng bởi các dữ liệu có giá trị, một mức độ mới, thay cho viết sẽ được tạo ra và chèn vào các bảng vẽ bản đồ. Các thông tin địa chỉ vật lý liên quan đến việc yêu cầu cũng sẽ được lưu trữ với mức độ. Nếu yêu cầu ghi đè bất kỳ dữ liệu hợp lệ, một hoặc nhiều mức độ đại diện cho dữ liệu hiện có sẽ phải được cập nhật. Những mức độ có thể được đặt bằng cách fi nding tất cả các mức độ trùng nhau một đến của các yêu cầu ghi. Một yêu cầu đọc, ngược lại, được coi là một *điều tra* mức độ nào. Để dịch các địa chỉ hợp lý của các yêu cầu đọc cho những người thể chất, VET sẽ tìm kiếm tất cả các mức độ hiện trùng mức độ yêu cầu đọc và trả về địa chỉ vật lý từ những người được tìm thấy.

Không giống như một trang hoặc chặn chương trình lập bản đồ có độ chi tiết của bản đồ là fi cố định hoặc là các trang hoặc các khối, các cửa hàng VET lập bản đồ thông tin ở mức độ khác nhau của các chi tiết, được xác định duy nhất bởi mỗi yêu cầu ghi riêng. Như sẽ được trình bày dưới đây một cách chi tiết, thông tin bản đồ của một yêu cầu ghi được đại diện bởi một hoặc nhiều *mức độ kinh*

*điển.* Vì kích thước của một mức độ kinh điển chỉ có thể bị giới hạn bởi kích thước

của toàn bộ một không gian địa chỉ logic, VET có thể duy trì thông tin bản đồ một cách ngắn gọn hơn và giảm kích thước của một bảng vẽ bản đồ đáng kể.

# Trie ảo

VET là một Trie các chuỗi nhị phân nhưng chỉ trong ý nghĩa hợp lý (như sẽ được thảo luận trong thời gian ngắn). Do đó, nó là một *ảo* Trie. Mỗi

chuỗi nhị phân gồm số không, những người thân và đặc biệt gọi là bit

*không quan tâm* bit (ký hiệu là '\*' ký tự). Các *không quan tâm* bit có thể chỉ xuất hiện ở phần cuối của một chuỗi. Tất cả các dây có chiều dài tương tự nhưng có thể có một số khác nhau của '\*' bit. Một chuỗi nhị phân với một vài dấu '\*' bit trong thực tế đại diện cho một mức độ nào đó có chiều dài là một sức mạnh của hai. Ví dụ, một chuỗi nhị phân

8-bit 0010 \*\*\*\* có thể được sử dụng để đại diện cho một mức độ nào đó có hợp lý địa chỉ bắt đầu là 00100000 và có chiều dài là 16. Vì không phải tất cả các mức độ có một sức mạnh-của-hai chiều dài, một mức độ nào đó có thể phải được phân chia cho một hoặc nhiều *kinh điển* mức độ trước khi được bổ sung vào VET. Về hình thức, mức độ kinh điển được định nghĩa là sau.

**De fi Định nghĩa 1.** *một mức độ \* \*, \* \* được cho là kinh điển nếu chiều dài \* là một sức mạnh của hai địa chỉ bắt đầu \* là một bội số của \*.*

Một mức độ kinh điển *\* \*, \* \** luôn có thể được biểu diễn bởi một chuỗi nhị phân duy nhất, mà có thể thu được bằng cách thay thế các số không trọng yếu không thể thiểu trong biểu diễn nhị phân của *\** với

đăng nhập 2 *\** '\*' Bit. Ví dụ, ở một mức độ kinh điển *\** 8, 4 *\** có thể được đại diện bởi một chuỗi nhị phân 8-bit (cho đơn giản) như sau:

*\** 000.010 \*\* *\*.*

Trong một Trie ảo, một mức độ kinh điển phục vụ như là một *Chìa khóa* để xác định mỗi node.

Lưu ý rằng địa chỉ bắt đầu vật lý *\** liên kết với một mức độ kinh điển được loại trừ trong De fi Định nghĩa 1. Kể từ *\** không liên quan đến bản đồ mức độ trong tờ giấy, để thuận tiện cho các cuộc thảo luận của chúng tôi nó chỉ là bỏ qua.



*a: <*

>

: Nút nội bộ: nút lá

: cuống lá

: mức độ lưu trữ

*b: <* 0

>

*c: <* 1

>

*d: <* 10

>

*h: <* 11

>

*f: <* 100

>

*i: <* 110

>

*o: <* 111

>

*e* 3

*g* : <1000 > *j* : <1100 > *m: <* 1101 >

*e* 1

*kkk:::* 1*<<*1110100101>

*n: <*1110110010>

*e* 2

Sung. 1. Trie ảo bằng cách chỉ định phạm vi Do *\** 1, *\** 2 và *\** 3

Hình 1 cho thấy một Trie ảo để lưu trữ các mức độ đầu vào trong các hình thức kinh điển như sau:

*\** 1 = { *\** 0 \*\*\*\*\*\*\* *\*, \** 1000 \*\*\*\* *\*}*

*\** 2 = { *\** 11001 \*\*\* *\*, \** 11010 \*\*\* *\*}*

*\** 3 = { *\** 111 \*\*\*\*\* *\*}.*

Nds fi chương trình VET một tập hợp các mức độ kinh điển cho họ, sử dụng các phím nút ở trên xuống thời trang, như trong Hình 1. Ví dụ, đưa ra *\** 1, các nút *một* 'Chìa khóa s ( *\*\*\*\*\*\*\*\*\*\*)* được so sánh với *\** 1 trong Trie ảo. Bởi vì chính không phải là

tương đương với *\** 1, VET xác định *đứa trẻ* của nút *một* có thể bao hàm *\** 1 trong toàn bộ. Trong trường hợp này, không phải của các con, các nút *b* và *c,* hoàn toàn có thể chứa *\** 1; do đó, *\** 1 được chia thành các mức độ nhỏ hơn và vẫn giữ được những kinh điển ứng cử viên. Bằng cách này, tất cả các mức độ kinh điển của *\** 1 có thể được đặt, và

* 1 cuối cùng cũng giữ nút *b* và *g.* Một khi các mức độ đầu vào khác *\** 2 và *\** 3 được lưu trữ, các Trie ảo sẽ giống như trong hình 1.

Trong một Trie ảo, có tồn tại hai loại nút: *lá* và

*nội bộ* những người thân. Họ chia sẻ phím thông thường (hoặc mức độ kinh điển) cấu trúc, nhưng chỉ có một nút lá có thể có một mức độ nhất định. Các nút lá nói trên *b* và *g* cửa hàng *\** 1, nút lá

*k* và *n* có *\** 2, và nút lá *o* giữ *\** 3 trong Hình 1. Một nút nội bộ, trái lại, chỉ

phục vụ như một *người giúp việc*

mà giúp đạt các nút lá. Làm thế nào các nút nội bộ được khai thác sẽ được giải thích trong Phần III-B.

Một nút trong Trie ảo không có bất kỳ con trỏ vật chất để con của nó. Thay vào đó, một nút được kết hợp với các nút con của nó chỉ có một cách logic và có thể xác định chúng bằng cách gấp trọng yếu nhất *không quan tâm* bit của nút hoặc là '0' hoặc '1'. Ví dụ, các nút *b* và *c* trong hình 1 có thể được xác định là con trái và bên phải của nút *một,* khi họ chia sẻ cùng *không quan tâm* bit ngoại trừ đầu tiên fi một ở mức độ kinh điển của họ.

# Bảng Hash

VET là một Trie ảo, nhưng nó *thể chất* lưu mức độ kinh điển thành một *bảng băm.* Bảng băm là một cấu trúc dữ liệu truyền thống để trả lại giá trị kết hợp với một phím băm qua một hàm băm. Trong một Trie ảo, một mức độ nhất định được lưu giữ trong các hình thức của những người theo giáo luật mà phục vụ như là chìa khóa nút, do đó phù hợp tốt với các cấu trúc bảng băm. Hơn nữa, một tra cứu bảng băm không dựa vào cách nhiều mục nằm trong bảng, như trái ngược với một cấu trúc vật lý Trie. Việc tra cứu bảng băm trung bình, do đó, thường được thực hiện trong vòng *O (* 1),

giả định rằng không thể fi trọng yếu trên OWS fl hoặc va chạm không diễn ra. Những đặc điểm này dẫn quyết định của chúng tôi để thực hiện một Trie ảo bởi một bảng băm.

1. Thuật toán cho TRIE ảo

Một I / O yêu cầu đến từ máy chủ hoặc là đọc hoặc viết hoạt động sẽ được thực hiện trên một đoạn bộ nhớ tro fl. Mỗi hoạt động đi kèm với địa chỉ bắt đầu hợp lý và kích thước của một đoạn. Đối với một hoạt động đọc, nhiệm vụ của VET là để dịch các địa chỉ hợp lý để những vật lý cho các trang bộ nhớ tro fl ở đoạn này. Nếu yêu cầu là một viết, VET được cung cấp với các địa chỉ vật lý, nơi các dữ liệu đang thực sự bằng văn bản. VET là sau đó chịu trách nhiệm cập nhật các thông tin bản đồ địa chỉ. Phần này trình bày các thuật toán quan trọng mà VET sử dụng để tìm kiếm hoặc cập nhật các thông tin bản đồ địa chỉ.

Như mô tả trong Phần II, VET duy trì các thông tin bản đồ trong các hình thức mức độ kinh điển và lưu trữ các mức độ trong một Trie ảo. Một lần nữa, chúng tôi thực hiện các Trie ảo sử dụng một bảng băm. Vì vậy, tất cả các hoạt động như chèn thêm, xóa và tìm kiếm được giải thích trong bối cảnh

của một Trie, nhưng họ đang thực sự được thực hiện bởi chèn băm, xóa và tra cứu.

# Cập nhật cho Viết yêu cầu

Một yêu cầu ghi đến từ máy chủ luôn luôn gây ra các dữ liệu được viết để làm sạch các trang trong bộ nhớ tro fl và thông tin bản đồ địa chỉ để được cập nhật cho phù hợp. Nếu dữ liệu được ghi vào một vị trí có địa chỉ logic là không bị chiếm đóng bởi dữ liệu hợp lệ, sau đó một mảnh mới của thông tin bản đồ sẽ được tạo ra và thêm vào. Nếu dữ liệu được ghi vào một vị trí có địa chỉ logic trùng - hoàn toàn hoặc một phần - những dữ liệu có giá trị hiện có, sau đó các thông tin bản đồ của các dữ liệu hiện có được ghi đè sẽ được loại bỏ hoàn toàn hoặc một phần thay thế bởi đó của dữ liệu đến. Kể từ khi chúng tôi lưu trữ các thông tin bản đồ địa chỉ trong các hình thức mức độ kinh điển trong các chương trình VET, xử lý một yêu cầu ghi sẽ làm gia tăng mức độ mới và loại bỏ các mức độ cũ một phần hoặc hoàn toàn từ một Trie ảo. Phần này công phu như thế nào các hoạt động cập nhật được thực hiện trong các chương trình VET.

* 1. *Chèn một Mức độ:* Mỗi yêu cầu ghi được đối xử như một mức độ nhất định có chứa địa chỉ bắt đầu và kích thước của dữ liệu được ghi. Bước đầu tiên kinh hướng xử lý một yêu cầu ghi là để xác định vị trí bất kỳ mức độ hiện chồng chéo một nhất định. Điều này có thể được thực hiện bởi các thuật toán tìm kiếm mô tả trong Phần III-B. Nhiệm vụ tiếp theo của chương trình VET là để lắp lại mức độ hiện tại được cập nhật bởi sự chồng chéo (nếu có), xóa các mức độ lỗi thời, và fi nally thêm mức độ nhất định.
     1. *LIS (Linear Insertion Scheme):* Nói chung, ở một mức độ nhất định không nhất thiết phải là kinh điển như yêu cầu có thể được đặt bất cứ nơi nào trong không gian địa chỉ. Do đó, nó cần phải được chuyển đổi thành một hoặc nhiều mức độ kinh điển. Trong quá trình định vị mức độ kinh điển của nó, là chương trình VET không chỉ tạo ra tất cả các nút tổ tiên của mình nhưng cho biết thêm mức độ kinh điển riêng của mình để Trie ảo. Chúng tôi kêu gọi kế hoạch này *LIS (* Tuyến tính Insertion Scheme).

Để hiểu làm thế nào LIS làm việc, chúng ta hãy xem lại Hình 1 trình bày

trong Phần II-B. vì *\** 3 = *\** 111 \*\*\*\*\* *\*,* Ví dụ, chương trình VET sử dụng LIS (tuyến tính) chèn các nút nội bộ *một,*

*c,* và *h* tiếp theo là nút lá *o* chỉ đến *\** 3. Nếu bất kỳ của các nút đã tồn tại trong Trie ảo, nó chỉ đơn giản là bỏ đi. Nếu nút lá *o* trước đây có mức độ hiện hành cũ của nó, nó sẽ được thay thế bằng *\** 3.

* 1. *Xóa một Mức độ:* Nếu một mức độ nào đó được tạo ra bởi một yêu cầu ghi đè lên mức độ hiện tại Trie ảo, sau đó các thông tin bản đồ được lưu trữ trong phạm vi hiện tại sẽ không còn giá trị hoặc là hoàn toàn hoặc một phần. Nếu nó cần phải được vô hiệu hoàn toàn, mức độ tương ứng của nó sẽ được loại bỏ khỏi Trie ảo. Nếu nó cần phải được chấm dứt hiệu lực một phần, mức độ tương ứng sẽ được chia để chỉ những phần hiệu lực có thể được gỡ bỏ từ Trie ảo.

Hình 2 exempli fi es việc huỷ bỏ hiệu một phần. Giả sử rằng một mức độ nhất định *\** 4 = *\** 1100100 \* *\** đến tại Trie. Bởi vì

* 4 sự chồng chéo *\** 2 hiện tại Trie, *\** 2 được phân rã thành các mức độ sau đây:

*\** 4: *{\** 1100100 \* *\*}*

*\** 5: *{\** 1100101 \* *\*, \** 110.011 \*\* *\*, \** 11010 \*\*\* *\*}.*

**Thuật toán 1:** LIS

**đầu vào :** *Chìa khóa (* ở mức hiện tại), *\* -* Một mức độ nhất định

**đầu ra:** *KHÔNG AI*

**1** *nút ←* Thực hiện một tra cứu với *Chìa khóa ;*

**2 nếu** *Chìa khóa là một mức độ kinh điển của \** **sau đó**

**3 nếu** *nút == NULL* **sau đó** Tạo một nút lá với *\*.*

**4 khác**

**5 nếu** *nút == nội* **sau đó** Xóa subtrie bắt nguồn từ *nút.*

**6** có *nút* giữ *\*,* tháo *nút* Là mức độ cũ nếu có.

**7 kết thúc**

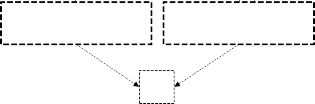
**số 8 khác**

**9 nếu** *nút == NULL* **sau đó** Tạo một nút nội bộ.

**10 else if** *nút == lá* **sau đó** *nút* thiết bị chuyển mạch *nội bộ.*

**11** Đệ quy fi nd các phần còn lại của mức độ kinh điển của *\** sử dụng con trái hoặc bên phải của *Chìa khóa.*

Sung. 3. Subtrie xóa



*h: <* 11

>

*e* 6

*i: <* 110

>

*o: <* 111

>

*e* 3

*j: <* 1100 >

*m: <* 1101

>

*k: <* 11001>

*n: <* 11010>

*e* 2

*trở thành một nút lá*

**12 kết thúc**

được thực hiện mà không đi qua các Trie ảo. Thay vào đó, nó sẽ được thực hiện ef hơn fi ciently bằng cách thực hiện một *tìm kiếm nhị phân*

chống lại các nút trên một con đường (hoặc *cấp độ)* của Trie ảo có chứa các nút mục tiêu.



*j: <* 11110000 >

*m: <* 11101101 >

*k:* 11001

*e* 2

*p: <* 110.010>

*q: <* 1100100> *r: <* 1100*rr::*1*<<*011>100101>

*e* 4

*e* 5

*n: <* 11010>

*t: <* 110.011>

Ví dụ, nếu địa chỉ logic chứa trong một yêu cầu đọc cho là 204 ( hoặc là 11001100 trong biểu diễn nhị phân), sau đó tìm kiếm bắt đầu từ điểm giữa của đường dẫn

gốc-to-bottom bằng cách lấy chỉ nửa fi đầu tiên của chuỗi nhị phân như một chìa khóa tìm kiếm. Nói cách khác, nửa thứ hai của chuỗi nhị phân được thay thế bằng '\*' ( *không quan tâm)* bit, và fi khóa cần tìm kiếm đầu tiên được hình thành như sau.

*\** 1100 \*\*\*\* *\**

Sung. 2. Ví dụ về Chấm dứt hiệu lực một phần

Bằng việc huỷ bỏ hiệu một phần, mức độ đến *\** 4 ghi đè lên một trong những mức độ kinh điển của *\** 2, và do đó, các chương trình VET tạo nút nội bộ *p* và nút lá *q* vì *\** 4. Ngoài ra, hiện nay mức độ mới *\** 5 bao gồm ba mức độ kinh điển khác, VET tạo ra các nút lá *r* và *t,* và chỉ cần giữ nút lá *n,* kết nối nó với *\** 5. Cuối cùng, *\** 2 được đuổi ra khỏi nhà với nút lá *k,* mà sử dụng để giữ lại một trong những mức độ kinh điển ( *\** 11001 \*\*\* *\*)* của *\** 2.

Lưu ý rằng một mức độ nào đó có thể yêu cầu hủy bỏ hiệu lực đến một hoặc nhiều những cái hiện tại cùng một thời điểm. Như minh họa trong hình 3, mức độ hiện tại *\** 2 = { *\** 11001 \*\*\* *\*, \** 11010 \*\*\* *\*}*

và *\** 3 = *\** 111 \*\*\*\*\* *\** bị vô hiệu được đưa ra một mức độ nào đó đến *\** 6 = *\** 11

\*\*\*\*\*\* *\*.* Hoạt động này có thể được thực hiện theo cách định vị và xóa tất cả các nút nội bộ và lá dùng để lưu trữ *\** 2 và *\** 3. Nó cũng có thể được tối ưu hóa bằng cách loại bỏ toàn bộ một subtrie bắt nguồn từ, nút nội bộ trước đây phổ biến *h* vì *\** 2 và *\** 3, theo sau bằng cách thêm *\** 6 đến nút *h*

bây giờ trở thành một lá (như đã thấy ở dòng 5-6 trong Algorithm 1).

# Tìm kiếm đọc yêu cầu

Khi một yêu cầu đọc đến từ máy chủ, địa chỉ logic của nó đã được dịch sang một vật lý. Điều này được thực hiện bằng cách tìm kiếm các Trie ảo cho một mức độ nào đó có chứa các địa chỉ logic. Kể từ khi một lần nữa các Trie lưu trữ các thông tin bản đồ là *ảo*

và thực hiện bởi một bảng băm, các hoạt động tra cứu có thể

Nếu một tra cứu với khóa tìm kiếm trên thành công và trận đấu được tìm thấy trong một nút lá, sau đó các thủ tục tìm kiếm sẽ chấm dứt. Từ thực tế là địa chỉ logic 11001100

được chứa trong một mức độ kinh điển *\** 1100 \*\*\*\* *\*,* các logicalto-vật lý địa chỉ dịch của 11001100 có thể thu được ngay lập tức từ các nút lá.

Nếu tra cứu thành công nhưng trận đấu được tìm thấy trong một nút nội bộ, sau đó các thủ tục tìm kiếm sẽ tiếp tục ở nửa dưới của con đường. Điều này là do các nút mục tiêu vẫn có thể được tìm thấy trong các subtrie bắt nguồn bởi các nút nội bộ mà trận đấu được tìm thấy. Để tiếp tục các thủ tục tìm kiếm ở nửa dưới của con đường, một khóa tìm kiếm mới được hình thành bằng cách khôi phục một nửa fi đầu tiên của các bit đeo mặt nạ ra bởi '\*' bit trong bước trước. Trong trường hợp này của ví dụ, phím tìm kiếm tiếp theo sẽ là

*\** 110.011 \*\* *\**

Một tra cứu có thể thất bại nếu không có subtrie có chứa node đích. Tuy nhiên, điều đó không nhất thiết có nghĩa rằng có tồn tại không có mức độ kinh điển có chứa địa chỉ logic. Nó chỉ có nghĩa là tìm kiếm nhị phân đã đi quá xa. Nếu một mức độ nào đó được tìm thấy ở một mức độ cao hơn (so với subtrie không tồn tại), sau đó các dịch địa chỉ logic-to-vật lý vẫn có thể thu được bằng cách sử dụng các thông tin được lưu trữ bởi các mức độ. Do đó, việc tìm kiếm phải tiếp tục bằng cách thu hẹp phạm vi lên trên con đường (hoặc bằng cách thêm dấu '\*' bit trong khóa tìm kiếm). Tiếp tục ví dụ chạy, phím tìm kiếm tiếp theo sẽ là

*\** 11001 \*\*\* *\**

Tóm lại, các hướng tìm tới được xác định bởi sự tồn tại của một nút nội bộ lưu trữ khóa tìm kiếm hiện tại. Hình 4 mô tả quá trình tìm kiếm dọc theo con đường của Trie ảo.

tra cứu trên *Chìa khóa* ở cấp *h ở giữa*



chúng ta có thể thấy rằng một số tổ tiên cho một mức độ kinh điển không được sử dụng cho việc tìm kiếm nhị phân trên các nút mục tiêu. Ví dụ, nút lá *k* trong hình 1 có tổ tiên sau của nó:

nút nội bộ *một \*\*\*\*\*\*\*\** nút nội bộ *c* 1 \*\*\*\*\*\*\* nút nội bộ *h* 11 \*\*\*\*\*\*

nút nội bộ *tôi* 110 \*\*\*\*\*

nút nội bộ *j* 1100 \*\*\*\*

*hh lê==n* 22

" *lên* "Đường dẫn trong trường hợp tra

Nút *j* là đầu tiên fi (và duy nhất) một để trở nên truy cập trong quá trình tìm kiếm nhị phân với các nút lá mục tiêu *k.* Nói cách khác, các nút nội bộ *một, c, h* và *tôi* sẽ không

*h mid =* 4

*Chìa khóa*

cứu *Thất bại* p *f*

" *xuống* " con đường

được sử dụng trong phần còn lại của việc tìm kiếm. Do đó, nó là khôn ngoan để chèn các nút nội bộ, ngoại trừ nút *j* cho vị trí nút lá *k.* Bằng cách thêm vào chỉ một nút nội

*h xuống =* 6

trong trường hợp tra cứu *sự thành công*

bộ không thể thiếu (s), chúng tôi có thể không chỉ dành ít thời gian trên mà còn tiết kiệm dung lượng bộ nhớ cho chèn của một mức độ nào đó. chương trình tối ưu hóa này được gọi là *BIS (* Binary Insertion Scheme).

Sung. 4. tìm kiếm nhị phân trên Levels VET



*b: <* 0

>

*h: <* 11

>

*o: <* 111 >

*e* 3

*gg* : <1000

>

*jj* : <1100 >

*m: <* 1101

>

*e* 1

*kkk:::* 1*<<*1110101001>

*nnn:::* 1*<<*1110110010>

*e* 2

Lưu ý rằng kịch bản được đưa ra trong ví dụ trên là hơi oversimpli fi ed. Nếu một yêu cầu đọc là lớn, sau đó mức độ của nó có thể không được bao phủ hoàn toàn bởi một mức độ kinh điển duy nhất được tìm thấy trong một nút lá. Trong trường hợp này, việc tìm kiếm phải tiếp tục bằng cách hình thành một khóa tìm kiếm mới cho các phần chưa được giải quyết của địa chỉ logic (không bao gồm mức độ che phủ một phần bởi mức độ kinh điển).

**Thuật toán 2:** mức Tìm kiếm

**đầu vào :** *\* -* Một mức độ nhất định

**đầu ra:** *\* -* Một danh sách các mức độ hiện chồng chéo *\**

**1** Đẩy rất fi khóa cần tìm kiếm đầu tiên vào *cây rơm.*

**2 trong khi** *stack là không có sản phẩm nào* **làm**

**3** *nút ←* Thực hiện một tra cứu với một phím popped.

**4 nếu** *nút / = VÔ GIÁ TRỊ* **sau đó** / \* Tra cứu thành công \* /

**5 nếu** *nút == lá* **sau đó** /\* Cuống lá \*/

**6** Đặt mức độ hiện giữ *nút* trong *\*,*

cập nhật địa chỉ mới bắt đầu sử dụng các mức độ được tìm thấy.

**7** Bắt đầu lại tìm kiếm với một chìa khóa mới nếu có thể.

**số 8 khác** / \* Nút nội bộ \* /

**9** Tiếp tục tìm kiếm ở nửa dưới của con đường với một chìa khóa mới nếu có thể.

**10 kết thúc**

**11 khác** / \* Suy Lookup \* /

**12** Tiếp tục tìm kiếm ở nửa trên của con đường với một chìa khóa mới nếu có thể.

**13 kết thúc**

**14 kết thúc**

Thuật toán 2 trau chuốt các thủ tục tìm kiếm mô tả ở trên. Lưu ý rằng ngăn xếp để lưu trữ các phím tìm kiếm được khai thác. Một lần nữa, sự chỉ đạo của các tìm kiếm được xác định bởi kết quả của một tra cứu với một phím popped.

# Tối ưu hóa bằng sơ đồ chèn nhị phân

Một quan sát cẩn thận bởi việc tìm kiếm nhị phân nói trên có thể dẫn đến những cải tiến về thủ tục chèn các LIS về cách sử dụng bộ nhớ và thời gian xử lý. Cụ thể,

Sung. 5. Trie ảo qua BIS bởi Extents Do *\** 1, *\** 2 và *\** 3

Hình 5 đại diện cho Trie ảo do BIS. Hầu hết các nút nội bộ không được nhìn thấy, như trái ngược với hình 1. Đó là bởi vì các phương pháp tiếp cận BIS xuất phát các nút nội bộ chỉ nằm cùng với các đường dẫn tìm kiếm nhị phân để định vị từng mức độ kinh điển, và thêm chúng vào trong Trie ảo.

**Thuật toán 3:** BIS

**đầu vào :** *Chìa khóa (* ở mức hiện tại), *\* -* Một mức độ nhất định

**đầu ra:** KHÔNG AI

**1 nếu** *Chìa khóa là một mức độ kinh điển của \** **sau đó**

**2** Xác định vị trí và xóa một mức độ lỗi thời nếu có.

**3** Chèn các nút nội bộ và lá, xem xét các đường dẫn tìm kiếm nhị phân trên *Chìa khóa.*

**4** Xóa subtrie bắt nguồn từ nút lá có *Chìa khóa.*

**5 khác** Đừng dòng 9-11 trong Algorithm 1.

Thuật toán 3 trong thời gian ngắn mô tả các quá trình của BIS. Hầu hết các bộ phận tương tự như thuật toán Thuật toán 1. BIS, tuy nhiên, là hơi khác nhau, vì nó sẽ xác định vị trí và loại bỏ một nút lá với một mức độ không hợp lệ mà có thể nằm trên một nút nội bộ mới được đưa vào, như minh họa ở dòng 2 . Ví dụ, trong hình 2 nút *k* với *\** 2 phải bị đuổi trước khi chèn các nút nội bộ *p.* hoạt động thêm này là không cần thiết bởi vì LIS nút *k* sẽ bị xóa đã được cấp có thời gian nút nội bộ *p* được chèn vào.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Kiểu | Tên | Quyết định. | Địa chỉ. Không gian | Số viết | Số lần đọc |
| Avg. Viết Kích | Avg. đọc Kích |
| Thực | tài chính ứng d | ụng OLTP [9] | 644GB | 8.2M | 2.5M |
| 3.7KB | 2.5KB |
| nhà MS Exch | ange Server [10], [11] | 542GB | 16 triệu | 0.4m |
| 4KB | 12.0KB |
| wdev | MS Exchange Server [10], [11] | 51GB | 1.1M | 0.23M (12.6KB) |
| 8.2KB | 12.6KB |
| WSF | Web sur hoạt động fi ng trên PC | 32GB | 0.3M | 93K |
| 22.5KB | 19.8KB |
| Sợi tổng hợp | ói mữa | phun [ 12] khối lượng công việc phát | 16GB *~* 1TB 50K |  | 0 |
| 83.8KB *~* 376.8KB N / A |  |

BẢNG tôi

R EAL / S YNTHETIC W ORKLOADS U SED CHO P ERFORMANCE E ĐỊNH GIÁ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Kiểu | tài chính | nhà | wdev | WSF |
| Bộ nhớ sử dụng (MB) | BIS | 15.07 | 23.25 | 1,15 | 4.69 |
| LIS | 16.89 | 28.50 | 1,63 | 6.03 |

Viết đọc

đã đọc

Avg. Đã qua thời gian (chúng tôi)

BIS

16.84

2.17

4,43

0.99

10,29

0.95

9,64

1,44

LIS

24,63

2.47

8,25

1,07

9,44

1.00

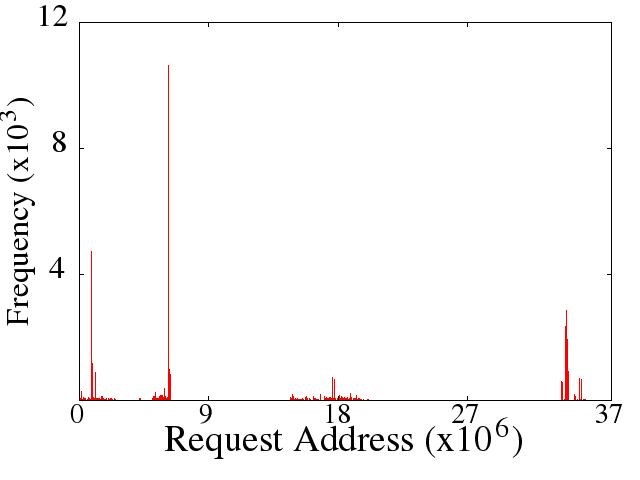
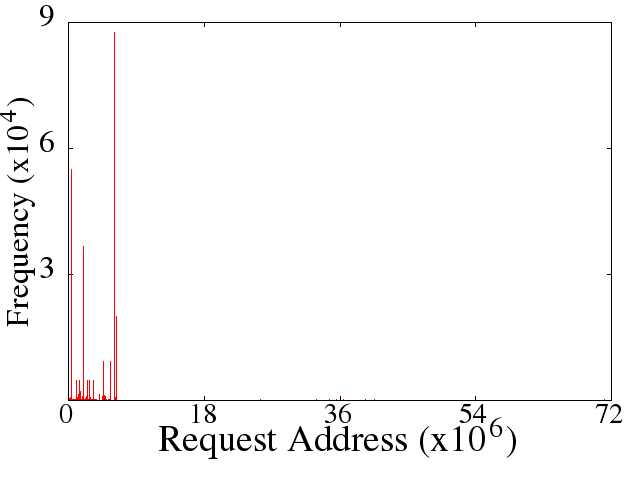
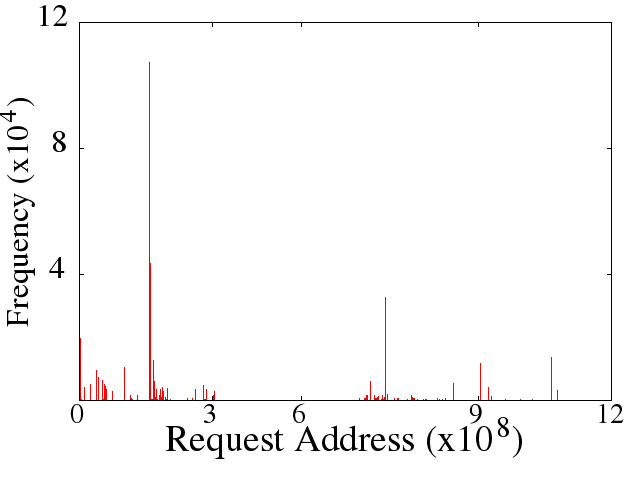
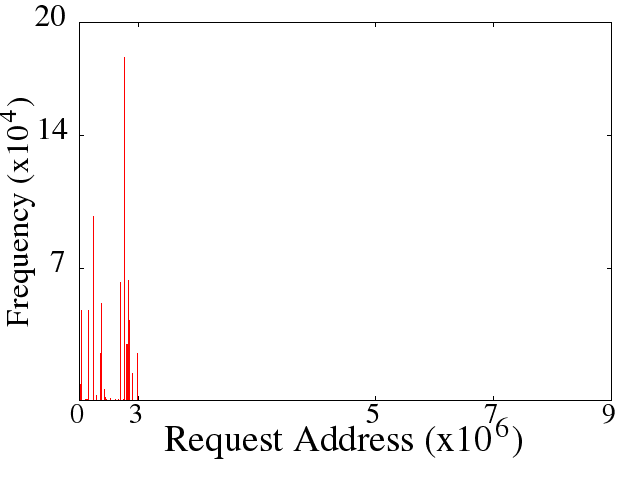
12.97

1,53

Viết đọc Viết đọc Viết

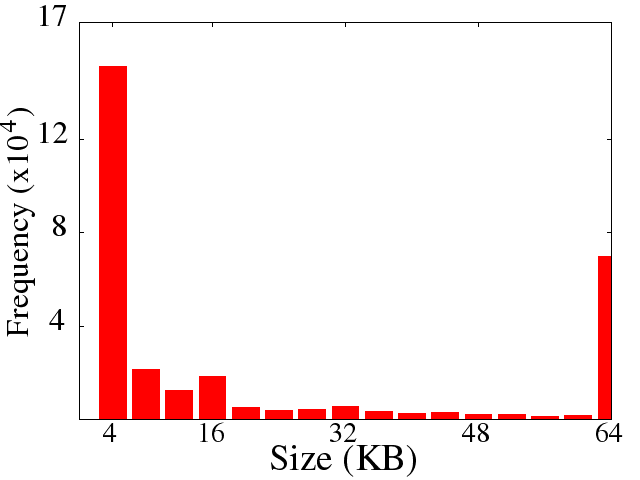
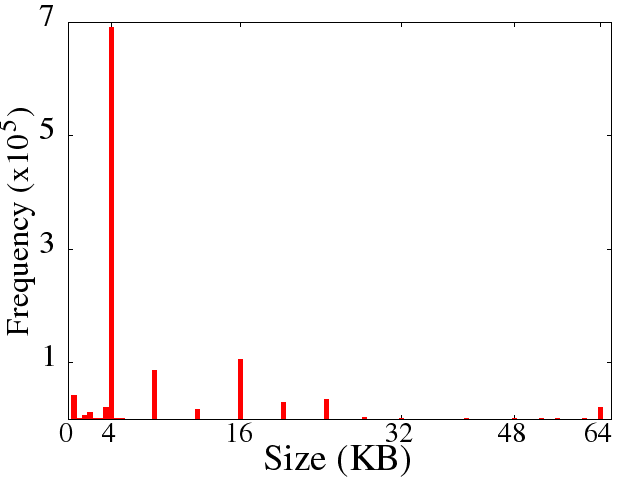
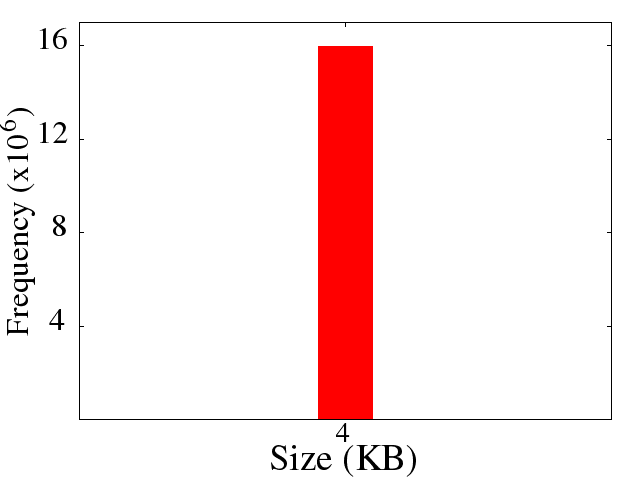
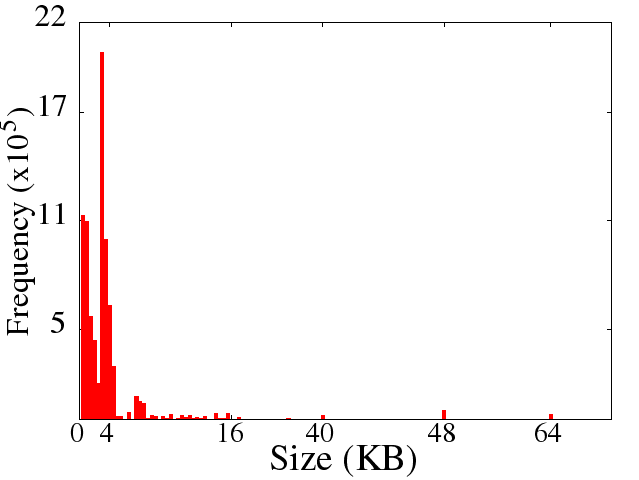
BẢNG II

M Emory F OOTPRINT VÀ Một VERAGE E lầm lổi T IMES



(A) tài chính (B) nhà (C) wdev (D) WSF

Sung. 6. Biểu đồ trên Viết Request Địa chỉ



(A) tài chính (B) nhà (C) wdev (D) WSF

Sung. 7. Biểu đồ trên Write Yêu cầu Kích

1. THỬ NGHIỆM

Trong phần này, chúng tôi mô tả các thiết lập môi trường của chúng tôi, và trình bày kết quả mô phỏng của chúng tôi.

# Cài đặt môi trường

Chúng tôi thực hiện tất cả các thuật toán được đề xuất trong ngôn ngữ C, và tiến hành các thí nghiệm trên một cấp thấp máy 32-bit với một bộ xử lý 3.00GHz 4 CPU Pentium và bộ nhớ 2GB, chạy các hệ thống Linux với một hạt nhân 2.6.32-32-generic phiên bản.

Các mô tả của các khối lượng công việc được sử dụng để đánh giá hiệu suất được đưa ra trong bảng I. Thế giới thực dấu vết -

tài chính, nhà cửa, wdev - thu được từ tôi công / O

dấu vết kho [9], [11], và WSF là khối lượng công việc thực tế được chiết xuất từ các hoạt động web. Các ói mữa dấu vết thu được bởi một máy phát điện khối lượng công việc [12] bao gồm chỉ các yêu cầu ghi. Như trong phần IV-B3, dấu vết này được đặc biệt sử dụng cho việc phân tích bộ nhớ trên cao với một không gian địa chỉ ngày càng tăng. Chúng tôi cũng cung cấp các biểu đồ về địa chỉ yêu cầu ghi và kích thước trong khối lượng công việc thực tế, như minh họa trong Hình 6 và 7.

# Đánh giá hiệu suất B.

Phần này trình bày kết quả đánh giá hiệu suất của chúng tôi.

* 1. *Phân tích toàn thể:* Chúng tôi phân tích hiệu suất tổng thể của chương trình VET trên các dấu vết thế giới thực, liên quan đến việc sử dụng bộ nhớ và mức độ kinh điển với.

2,5 3

700

600

PMT

HMT VET

500



Tạo bởi một Write Req. Chồng chéo một Write Req. Trở lại Để một Read Req.

Canonical định phạm vi trung bình

1,5 2

400

300

Bộ nhớ sử dụng (MB)

200

0.5 1

100

0

nhà tài trợ wdev WSF

0

nhà tài trợ wdev WSF

Sung. 8. Canonical Mức kê trên Với I / O yêu cầu

* + 1. *Mức tiêu thụ bộ nhớ:* Như đã thấy trong Bảng II, số lượng bộ nhớ cần thiết để điều trị mỗi dấu vết là tương đối rất nhỏ, xem xét các không gian địa chỉ của các dấu vết. (Chắc chắn, đã có một số bản đồ trên không.) Speci fi biệt, chương trình VET chỉ được sử dụng nhiều nhất về bộ nhớ 17MB để chế biến các tài chính dấu vết, trong đó có không gian địa chỉ lớn nhất, không phân biệt bất cứ phương pháp chèn (LIS hoặc BIS) đã được sử dụng. Rõ ràng, nhà tiêu thụ các hợp đồng lớn nhất của bộ nhớ trong số đó.

Đó là bởi vì nhà khối lượng công việc cho thấy phạm vi rộng nhất của các địa chỉ yêu cầu, như minh họa trong hình 6, và nó đã có thêm rất nhiều ghi yêu cầu hơn bất kỳ khối lượng công việc khác, như được chỉ ra trong Bảng I. Điều này ngụ ý rằng một thỏa thuận lớn về thông tin bản đồ cần phải được tạo ra trên toàn bộ không gian địa chỉ, và tương ứng với dung lượng bộ nhớ là cần thiết cho nhà. Ngược lại, các yêu cầu ghi trong wdev là khả năng hiển thị một dải địa chỉ tương đối hẹp so với các dấu vết khác. Ngoài ra, các địa chỉ yêu cầu cùng một xu hướng được truy cập nhiều lần, do đó phát sinh không có thông tin bản đồ phụ bản. Vì lý do này, wdev theo dõi bộ nhớ ít cần thiết hơn những người khác. (Đề án VET là de fi nitely thuận lợi để loại khối lượng công việc.) Các tài chính khối lượng công việc dường như có như hẹp một dải địa chỉ như của wdev. Tuy nhiên, nhiều người trong số các địa chỉ không rõ ràng trong hình 6 (a) là xa rải rác rộng rãi hơn trong và ngay cả

bên ngoài phạm vi địa chỉ nhìn thấy được. Tất nhiên, số lượng các yêu cầu

là đến nay lớn hơn so với wdev cũng. Các WSF dấu vết dường như đã có một loạt địa chỉ rộng hơn tài chính

một, nhưng trên thực tế, nó được sử dụng chỉ một vài địa chỉ trong toàn bộ phạm vi, và do đó, bộ nhớ tiêu thụ bởi các chương trình VET là ít hơn nhiều hơn tài chính.

* + 1. *Các phạm vi quy chuẩn:* Hình 8 minh họa các mức độ kinh điển trung bình đối với một yêu cầu I / O. Đối với mỗi khối lượng công việc, các tiên fi và thanh thứ hai đánh dấu mức độ kinh điển trung bình đã được tạo ra bởi và chồng chéo ở một mức độ nhất định, tương ứng. Người thứ ba chỉ ra mức độ kinh điển trung bình đáp ứng một mức độ yêu cầu.

Như đã trình bày, các tài chính và nhà khối lượng công việc yêu cầu các con số trung bình lớn nhất và ít nhất trong mức độ kinh điển liên quan đến một yêu cầu I / O, tương ứng. Trong các dấu vết khác,

Sung. 9. Bộ nhớ So sánh trên cao

ít hơn nhiều mức độ kinh điển trung bình mỗi yêu cầu đọc được trả lời cho wdev hơn cho WSF. Trong trường hợp viết, trong khi đó, wdev dấu vết cho thấy mức độ hơi trung bình nhiều chồng chéo nhưng khoảng một nửa trong số những người được tạo ra mỗi ghi yêu cầu hơn WSF dấu vết.

Các quan sát trên là khá phù hợp với kết quả thời gian trôi qua trung bình thể hiện trong Bảng II. Các mức độ kinh điển ít tham gia, nhanh hơn mức trung bình trôi qua lần trên một I / O yêu cầu nhất định.

Lưu ý rằng tất cả các dấu vết trên trung bình tạo ra tối đa là ba mức độ tắc cho một mức độ đầu vào, như thể hiện trong hình 8. Điều này chứng tỏ rằng các lý thuyết ràng buộc (2 đăng nhập *| \* |)* về số lượng trung bình của mức độ kinh điển là quá bi quan trong thực tế. Các bằng chứng sẽ được cung cấp trong phiên bản mở rộng của tờ giấy.

* 1. *VET vs trang /Lạp bản đồ Hybrid :* Như thể hiện trong hình 9, các chương trình VET sử dụng ít bộ nhớ hơn so với một bảng vẽ bản đồ trang (PMT) [2] hoặc một bảng vẽ bản đồ lai (HMT) [4], [5], [6], [7]. Speci fi biệt, khi điều trị khối lượng công việc, nó tiêu thụ chỉ khoảng 2,3%, 4,3%, 2,3%, và 27% của tổng số bộ nhớ yêu cầu của PMT theo thứ tự. Ngay cả so với HMT, các chương trình VET sử dụng chỉ khoảng

5,3%, 9,8%, 5,2% và 64% của tổng số bộ nhớ của HMT theo thứ tự. Điều này là do thực tế là bộ nhớ yêu cầu của VET là không phụ thuộc vào kích thước không gian địa chỉ nhưng chủ yếu được xác định bởi *đặc điểm khối lượng công việc.* Các đặc tính có thể được de fi định nghĩa theo cách nhỏ hay lớn, yêu cầu ghi là, và làm thế nào phân tán hoặc thu hẹp địa chỉ của họ.

Ví dụ, kích thước ghi yêu cầu trung bình của tài chính hoặc là

nhà không phải là quá nhỏ mà lúc tốt nhất hai nút nội bộ được tạo ra. Hơn nữa, cả hai khối lượng công việc có không gian địa chỉ lớn hơn 0.5TB, nhưng viết của họ chạm vào chỉ một vài phần của từng không gian, mà không tạo ra thông tin bản đồ nhiều trong VET. Do đó, một số lượng lớn các bộ nhớ có thể được cứu.

Các wdev dấu vết đã có một không gian địa chỉ nhỏ, nhưng một lần nữa nhiều lần ghi, mà kích thước cũng được tối ưu hóa để VET, có xu hướng sử dụng cùng một địa

chỉ truy cập trước đây, do đó gây không tiêu thụ bộ nhớ bổ sung. Trong trường hợp WSF, nó có không gian địa chỉ nhỏ nhất, và phần thậm chí còn ít hơn của không gian

được còn nguyên vẹn không giống như các dấu vết khác. Do đó, việc giảm bộ nhớ trên WSF không quá lớn như của những người khác.

* 1. *Khả năng mở rộng thử nghiệm:* Trong thí nghiệm này, chúng tôi dân sự

ói mữa khối lượng công việc bằng cách thúc đẩy phát tán đi [12] công cụ; nghĩa là, với một không gian địa chỉ tăng từ 16GB đến 1TB, 50K có yêu cầu, có kích thước là một bội số của 8KB và xác định giữa 8KB và 1MB, hình thành từng khối lượng công việc.

1000

100

Sử dụng bộ nhớ - Đăng Scale

10

1

200 400 600 800 1000

Flash Memory Device suất (GB) PMT HMT VET

Sung. 10 sử dụng bộ nhớ trên không gian địa chỉ phát triển

Hình 10 triển lãm bộ nhớ tiêu thụ của tất cả các chương trình. Bởi vì một sự khác biệt rất lớn quan sát, sử dụng được vẽ trong quy mô đăng nhập. Khi không gian đã lớn hơn, các chương trình thông thường bị tràn bộ nhớ rất lớn, so với các chương trình VET rằng thay vẫn fl tại. Đặc biệt, không gian địa chỉ 1TB nới rộng khoảng cách giữa các PMT và VET lên đến hai bậc độ lớn.

1. CÔNG VIỆC CÓ LIÊN QUAN

Waldvogel *et al. [* 13] đề xuất một thuật toán có được sản phẩm tốt nhất phù hợp với IP trước fi x thông qua một tìm kiếm nhị phân trên một bảng định tuyến bằng trước dài fi x tổ chức. Trong thực tế, các fi trước x và các bảng định tuyến tương ứng với các khái niệm về một mức độ kinh điển và một Trie ảo được đề cập trong bài báo. Tác phẩm của ông được trình bày trong phạm vi của mạng, do đó, âm thanh tương tự như các chương trình VET.

Tuy nhiên, sự khác biệt chính giữa VET và làm việc Waldvogel có thể được tóm tắt như sau. Thứ nhất, mức độ lưu trữ trong một Trie ảo phải vẫn không chồng chéo (tách rời) ở tất cả các lần, do đó các thông tin bản đồ mới nhất nên được giữ bằng cách các out-of-ngày một hiệu lực. Trong công việc của mình, trái lại, XES fi trước có một phổ biến có thể cùng tồn tại trong bảng định tuyến.

Thứ hai, công việc của mình cần phải sử dụng *quay lui*

để fi thứ tốt nhất, kết hợp chính xác trước fi x trừ dài nhất được tìm thấy. Trong đề án VET, sự quay lui không diễn ra, như việc tìm kiếm một địa chỉ logic nhất định được chấm dứt ngay lập tức nếu bất kỳ mức độ nào đó có chứa địa chỉ không thể được đặt. Thứ ba, một fi trước tùy ý x tự giữ *kinh điển*

bất động sản, trong khi một mức độ nhất định không phải là kinh điển. Cuối cùng, bảng định tuyến trong công việc của mình được giả định là *tĩnh;*

cụ thể là, nó không thay đổi nhiều theo thời gian. Trong khi đó, VET là khả năng

xử lý *năng động* cập nhật các mức độ.

Trang bản đồ [2] trưng bày các chi phí lập bản đồ ít nhất, *O (* 1).

Đề án VET cũng bảo vệ thời gian tìm kiếm chỉ trong vòng

*O (* đăng nhập đăng nhập *| \* |),* nhưng nó vẫn còn tụt lại phía sau trang bản đồ [2] đối với thời gian lập bản đồ với.

DFTL [14] đã được đề xuất để tấn công các vấn đề ghi ngẫu nhiên trong các

thiết bị bộ nhớ fl ash bằng cách tận dụng địa phương thời gian. Tuy nhiên, nó vẫn còn dính vào đồ trang, vì vậy nó không thể thoát khỏi gánh nặng của chi phí bộ nhớ cùng với một không gian địa chỉ ngày càng tăng.

khối tinh khiết lập bản đồ [3] thường kiếm được giảm không gian nhất. Nó, tuy nhiên, không thể thưởng thức sử dụng fl linh hoạt của các trang tro fl vật lý, bởi vì vấn đề của số trang logic bù đắp mà phải fi cố định trong một khối. Kết quả là, thu gom rác thải trên không là rất cao, và nó không phải là hoàn toàn được sử dụng trong thực tế.

Lai lập bản đồ [4], [5], [6], [7] để vượt qua những hạn chế như vậy cũng đã được đề xuất. Tuy nhiên, nó không phải là quá fl linh hoạt như lập bản đồ trang. Nó thậm chí không được giảm nhiều bộ nhớ hơn VET, như trong thí nghiệm của chúng tôi.

1. PHẦN KẾT LUẬN

Trong bài báo này, chúng tôi đề xuất *BÁC SĨ THÚ Y,* một chương trình lập bản đồ mức độ mới cho các thiết bị bộ nhớ tro fl. So với các đề án lập bản đồ truyền thống, VET tăng giảm không gian đáng kể bởi thậm chí lên đến một bậc trên dấu vết thực sự, và nó cũng có quy mô rất tốt với sự gia tăng không gian bởi khối lượng công việc tổng hợp. giảm trí nhớ đáng kể như vậy đã đạt được với chi phí lập bản đồ không đáng kể.

1. LỜI CẢM ƠN

Công trình này được hỗ trợ bởi Quỹ Khoa học quốc gia thuộc số NSF Grant IIS-0.848.503.

R EFERENCES

[1] Tập đoàn Intel, "Hiểu biết Flash Dịch Layer (FTL)

Speci fi cation, "App. Lưu ý AP-684, 1998. [2] Birrell, A. et al., "Một thiết kế cho đĩa High-Performance Flash," *ACM*

*SIGOPS Operating Systems Review,* vol. 41, không có. . 2, tr 88-93, tháng tư năm 2007. [3] Ban, A., "Flash File System", tháng 4 năm 1995, uS sáng chế 5404485. [4] -, "Flash File System Tối ưu hóa cho Trang-Chế độ Flash Technolo -

Gies, "August 1999, uS sáng chế 5937425. [5] Lee, SW. et al., "Một Log Buffer-Dựa

Flash Dịch Lớp Sử dụng

Đã Associative ngành dịch " *ACM Trans. trên hệ thống máy tính nhúng,* vol. 6, không có. 3, trang 1-27, 2007. [6] Jung, D. et al, "Superblock FTL:.. Một Superblock-Dựa Flash dịch

Layer với một Địa chỉ lai Scheme dịch " *ACM Trans. trên hệ thống máy tính nhúng,* vol. 9, không có. 4, trang 1-41, 2010. [7] Lee, S. et al, "CUỐI:.. Địa phương-Aware ngành dịch cho NAND

Flash Memory-Based hệ thống lưu trữ, " *ACM SIGOPS Operating Systems Review,* vol. 42, tr. 36-42, 2008. [8] Agrawal, N. et al., "Thiết kế Cân bằng cho SSD hiệu suất," trong *pro-*

*ceeding của USENIX ATC 2008.* USENIX, 2008, tr. 57-70. [9] UMass vết Repository, "OLTP ứng dụng I / O Trace," năm 2011,

[http://traces.cs.umass.edu.](http://traces.cs.umass.edu/)

. [10] Narayanan, D. et al, "Viết Tắt nạp: Thực hành Quản lý năng lượng cho lưu trữ doanh nghiệp ", trong *Kỷ yếu 6 USENIX Conf. trên FAST.* USENIX, 2008, tr. 253-267.

[11] Hiệp hội Công nghiệp Phần mềm Cisco IOS, "Block I / O vết Repository,"

2011, [http://iotta.snia.org/tracetypes/3.](http://iotta.snia.org/tracetypes/3)

[12] Patterson, A., "phun: An I / O Đo lường hiệu suất và Load

Thế hệ Tool, "2011, [http://spew.berlios.de/.](http://spew.berlios.de/) [13] Waldvogel, M. et al., "Khả năng mở rộng tốc độ cao IP Routing tra cứu," *ACM*

*SIGCOMM Máy tính Truyền Review,* vol. 27, không có. 4, tr. 25-36,

Năm 1997.

[14] Gupta, A. et al, "DFTL:. Một Flash dịch lớp sử dụng Demand-

Dựa Caching chọn lọc Mappings trang-Level Address ", trong *Kỷ Conf 14 Intl '. trên ASPLOS.* ACM, 2009, tr. 229-240.