Un mécanisme de cache pour les E/S séquentielles en mémoires flash

Pierre Olivier, Jalil Boukhobza

RenPar'20 / **SympA'14** / CFSE 8
Saint-Malo, France, du 10 au 13 mai 2011

Un mécanisme de cache pour les E/S séquentielles en mémoires flash

Pierre Olivier*, Jalil Boukhobza+

Université Européenne de Bretagne Occidentale, Université de Brest
+UMR 3192 Lab-STICC, *Département informatique,
20 avenue Le Gorgeu - CS 93837, 29238 Brest cedex 3
pierre.olivier@etudiant.univ-brest.fr, jalil.boukhobza@univ-brest.fr

---

Résumé

En raison d'une demande exponentielle du marché, le coût des systèmes de stockage à base de mémoire NAND flash chute de manière importante. Ces mémoires deviennent le principal média de stockage dans le domaine de l'informatique mobile, et tendent à être moins confinées à ce domaine. Néanmoins, cette technologie n'est pas assez mature pour permettre une utilisation à grande échelle pour les systèmes de stockage en entreprise. Cela est dû aux faibles performances des opérations d'écriture, causées par la structure interne de ces mémoires. La contrainte majeure de cette technologie est le nombre limité d'effacements, provoquant l'usure rapide de la mémoire. Pour pallier ce problème, les solutions actuelles implémentent des politiques de répartition de l'usure (*wear levelling*) pour augmenter la durée de vie des mémoires. Ces politiques, intégrées au sein de la *FTL* (*Flash Translation Layer*), contribuent grandement à la baisse de performance en écriture. Dans cet article, nous proposons d'augmenter les performances en réduisant le problème de l'usure en absorbant les effacements de blocs grâce à un système de double cache, remplaçant la plupart des services de la FTL. Les évaluations expérimentales menées s'appuient sur des traces réelles et synthétiques. Notre système de cache améliore considérablement les performances par rapport aux FTLs actuelles, réduisant de plus de 65 % le temps de réponse moyen et le nombre d'opérations d'effacement pour certaines traces intensives en écriture.

Mots-clés : Mémoires flash NAND, cache, FTL, performances, stockage, charge E/S, wear levelling.

---

1. Introduction

Les mémoires flash NAND sont de plus en plus utilisées comme système de stockage principal. On peut les trouver dans les lecteurs mp3, les smartphones, les ordinateurs portables, et dans un grand nombre d'appareils électroniques. Ces mémoires sont basées sur des puces à semi-conducteurs leur donnant des caractéristiques très intéressantes. Elles sont petites, légères, résistantes aux chocs, et très peu consommatrices d'énergie. Pour toutes ces raisons, elles sont considérées comme une technologie prometteuse pour les systèmes de stockage de masse [8] [9] [10]. Le stockage à base de mémoire flash est toutefois très coûteux : 5$/GO début 2009 [9] par rapport aux disques durs : 0.3$/GO en 2008 [8]. Cependant, les coûts chutent drastiquement en raison de la maîtrise des processus de fabrication et d'une demande exponentielle du marché. Cette chute continue du prix au bit, et la compatibilité avec les systèmes de stockage traditionnels encourage les entreprises à migrer vers des solutions de systèmes de stockage de masse à base de mémoires flash. Ce faisant, elles font face à un problème de performances [1] : certains disques durs surpassent encore les mémoires flash sur certaines traces intensives en écriture, séquentielles comme aléatoires. Cet article ne traite pas le problème des faibles performances en écriture aléatoire, même si l'on propose quelques perspectives intéressantes. Les techniques présentées

améliorent les performances des mémoires flash NAND en écriture séquentielle.
Les faibles performances des mémoires flash sont dues au nombre limité d'opérations d'effacement que l'on peut exécuter sur un bloc donné. La FTL est une couche matérielle/logicielle implémentée au sein du périphérique flash. L'une de ses fonctionnalitées principales est la traduction d'adresses logiques vers des adresses physiques. A travers cette traduction est accompli le *wear levelling*, la technique généralement adoptée pour pallier le problème de la limitation du nombre d'opérations d'effacement. Le *wear levelling* consiste à répartir les opérations d'écriture sur l'intégralité de la surface de la mémoire, pour augmenter la durée de vie moyenne des blocs. Ainsi, pour modifier des données, les blocs les contenants doivent être entièrement copiés vers un autre emplacement. L'utilisation d'une telle technique implique de prendre en considération la validité des données, car différentes versions des mêmes données peuvent se trouver dans différents blocs. Il faut également implémenter des mécanismes de ramasse-miettes (*garbage collection*) pour récupérer de l'espace libre au besoin.
Le *wear levelling* dans la FTL s'appuie sur de la SRAM, intégrée à la mémoire flash, où les tables de traduction sont stockées. L'utilisation d'une telle mémoire étant très coûteuse, les concepteurs de FTLs essayent d'en minimiser la taille. Il s'agit de trouver un compromis entre, d'une part, l'accroissement des performances en utilisant des algorithmes de traduction à base de pages, consommant beaucoup de SRAM mais réduisant le nombre d'opérations d'effacement, et d'autre part, la réduction de l'espace de SRAM en réduisant les métadonnées et donc en augmentant la granularité de la traduction ce qui accroît le nombre d'effacements. L'ensemble des solutions actuelles se situent entre ces deux extrêmes.
Dans cet article, nous proposons de remplacer les techniques de *wear levelling* existantes par un nouveau système de cache de petite taille, flexible et configurable nommé *C-lash*. C-lash prend en compte la structure et les contraintes de la mémoire flash, et se loge dans une SRAM. Ce système de cache est utilisé principalement comme un tampon, retardant les opérations d'écriture. Les contributions de cette étude sont les suivantes : 1) une nouvelle technique de cache remplaçant les solutions de *wear levelling*. Ce système montre une importante hausse de performance pour des traces réelles, de plus de 60 % séquentielles. 2) l'implémentation d'un simulateur utilisé pour réaliser l'évaluation de performance basé sur FlashSim [12] [1], implémenté en étendant DiskSim 3.0 [13]. Nous avons ajouté le support du cache pour FlashSim, qui intégre déjà la mémoire flash, le contrôleur, les pilotes et les interconnexions. 3) la validation avec un grand nombre de charges d'E/S réelles comme des traces OLTP et de moteurs de recherche, ainsi que différentes traces synthétiques.
Cet article est organisé comme suit : On décrit les mémoires flash dans la section état de l'art, et certains travaux précédemment réalisés sont étudiés dans la troisième section. Dans la quatrième section, les politiques et l'architecture de C-lash sont présentées. La section suivante décrit la méthode d'évaluation de performance, et la sixième section présente les résultats des simulations. En dernière partie on conclut et on donne certaines perspectives.

2. Etat de l'art et travaux précédemment réalisés

Les mémoires flash sont des mémoires non volatiles de type EEPROM. Elles sont de deux types : 1) la flash NOR et 2) la flash NAND. La flash NOR supporte un accès aléatoire par bit, possède une faible densité de données et un coût supérieur comparée à la flash NAND. La flash NOR est appropriée pour stocker du code [14]. Les mémoires NAND sont adressées par blocs et offrent une densité supérieure à un coût moins important. Elles offrent de bonnes performances pour d'importantes opérations de lecture/écriture. Ces propriétés les rendent plus adéquates pour le stockage de données [14]. L'étude présentée dans cet article traite des mémoires flash NAND.
La structure d'une mémoire flash NAND est la suivante : elle est composée d'une ou plusieurs puces ; chaque puce est divisée en plusieurs plans. Un plan est composé d'un nombre fixé de blocs, chacun d'entre eux contenant un nombre (multiple de 32) fixé de pages. Les anciennes versions de mémoires flash (*small block*) sont composées de blocs, chacun contenant 32 pages de 512 Octets. Les versions récentes (*large block*) ont des blocs de 64 pages de 2 KO. Une page est divisée en un espace de données utilisateur, et en une petite zone de métadonnées (64 Octets pour une page de 2 KO) [1] [3].
Trois opérations sont supportées : lecture, écriture, et effacement. La lecture et l'écriture se font sur les pages, tandis que les effacements se font sur des blocs. Dans un bloc donné, les pages doivent être écrites séquentiellement. La mémoire NAND flash ne supporte pas la modification de données sur place. Pour

modifier une page, cette dernière doit être effacée et réécrite, ou complètement copiée vers une autre page pour éviter des latences supplémentaires. Dans ce cas, l'entrée correspondante dans la table de traduction est mise à jour. L'opération d'effacement est effectuée de manière asynchrone. S'il n'y pas de pages libres disponibles, le ramasse-miettes est exécuté pour recycler les pages invalidées.
Une des contraintes fondamentales des mémoires flash est le nombre limité de cycles d'écriture/effacement (10^4 pour les mémoires MLC, 10^5 pour les SLC) [11]. Lorsque le nombre maximum de cycles d'effacements est atteint, une cellule mémoire ne peut plus contenir de données. Des cellules de rechange existent sur la puce pour pallier ce problème d'usure. A cause de la localité des données (temporelle et/ou spatiale dans de nombreuses traces) certains de ces blocs mémoires peuvent être beaucoup plus utilisés que les autres et tendent à s'user plus rapidement. Cet inconvénient pousse la communauté des chercheurs à réfléchir au problème du *wear levelling* pour répartir les opérations d'écriture sur tous les blocs, même si ces techniques réduisent drastiquement les performances. La plupart des systèmes de stockage à base de mémoire NAND intégrent une couche FTL. La FTL émule un périphérique de type bloc, en maintenant les tables de traduction d'adresse des blocs logiques (LBA, *Logical Block Adress*) vers les blocs physiques (PBA, *Physical Block Adress*).

2.1. Algorithmes de FTL de base

Les algorithmes de FTL peuvent être classifiés en 3 catégories principales selon la granularité de la traduction.

Traduction par page. La traduction par page (*page mapping*) est un système dans lequel chaque page logique est indépendamment mappée vers une page physique. Quand une requête d'écriture adressant une page (qui contient déjà des données) est reçue, la page physique est invalidée, et la page logique est mappée vers une autre page libre, évitant ainsi une opération d'effacement. Le *page mapping* montre de bonnes performances en termes de temps de réponse et de nombre d'effacements. L'inconvénient principal du *page mapping* est la taille de la table de traduction : elle est égale au nombre de pages mappées, qui peut être considérable.

Traduction par bloc. Les systèmes à base de traduction par bloc (*block mapping*) prennent en compte la granularité d'un bloc plutôt que celle d'une page. La table de traduction contient les informations de traduction entre blocs logiques et physiques, et l'*offset* d'une page à l'intérieur d'un bloc reste toujours le même. Comparé au *page mapping*, la taille de la table est radicalement réduite : elle est égale au nombre de blocs du périphérique flash. Avec le *block mapping*, les problèmes apparaissent lorsqu'une requête de modification arrive, impliquant une coûteuse opération de copie de l'intégralité de l'ancien bloc vers un bloc libre, ce qui explique les mauvaises performances du *block mapping*.

Traduction hybride. Avec cette technique, la mémoire flash est divisée en deux parties [5] : les blocs de données et les blocs de *log*. Les premiers sont mappés par bloc, l'*offset* physique de chaque page écrite doit toujours être égal à l'*offset* logique. Les seconds sont mappés par page, ainsi l'*offset* physique peut être différent de l'*offset* logique, ce qui est plus flexible. Comme la méthode de *page mapping* requiert un grand espace de SRAM pour les métadonnées, le nombre de blocs de *log* est limité. Les blocs de *log* maintiennent les données les plus fréquemment/récemment utilisées alors que les données les moins accédées sont reportées sur les blocs de données. Quand le nombre de blocs de *log* devient faible, ces blocs sont fusionnés et transformés en blocs de données (mapping par bloc). Cette opération réduit considérablement les performances d'un système à base de traduction hybride.

2.2. Algorithmes de FTL avancés

Dans cet article, nous proposons un système de cache en remplacement des algorithmes de *wear levelling* et de ramasse-miettes. Nous comparons ce système avec les systèmes de FTLs récents suivants : Fully Associative Sector Translation (FAST) et Demand-Based Flash Translation Layer (DFTL).
FAST [5] est une FTL hybride, utilisant une table de traduction basée sur des blocs, et le partitionnement blocs de *log*/blocs de données présenté précédemment. FAST sépare les blocs de *log* en deux espaces : un espace d'écriture séquentielle, et un espace d'écriture aléatoire. lorsqu'une modification de donnée survient, la nouvelle version de la page est écrite dans l'un des blocs de *log*, selon la séquentialité des

requêtes. La taille de l'espace des blocs de *log* est d'environ 0,07 % de la taille totale de la mémoire.
DFTL [1] est un système basé sur le *page mapping*. Il résout la contrainte de la taille de la table en plaçant une partie en SRAM. Le reste des informations de traduction est stocké sur la mémoire flash. Pour améliorer le temps de réponse, les adresses des pages les plus accédées sont stockées en SRAM : le système prend en considération la localité temporelle. DFTL divise l'espace flash en deux parties : l'espace de traduction (0,02 % de l'espace total) qui contient les métadonnées, et l'espace de données.
De nombreux autres systèmes de FTL on été développés : une FTL convertible (page/bloc, CFTL) [16], STAFF (State Transition Fast Based FTL) [15], Mitsubishi [19], SSL [20], NFTL [17], d'autres algorithmes hybrides comme BAST [5], BFTL (B-Tree FTL) [18], etc.

2.3. Systèmes de cache et FTLs pour mémoires flash

Même si les techniques de FTL conçues sont de plus en plus efficaces, les performances des opérations d'écriture sont toujours très faibles. Des systèmes de mémoire tampon ont étés conçus pour pallier ce problème, en réorganisant les flux de requêtes non séquentielles avant de les envoyer vers la FTL. Pour ce faire, ces systèmes utilisent une certaine quantité de mémoire RAM placée en amont de la FTL.
Clean First Least Recently Used (CFLRU) [6] est une politique LRU utilisant une liste de pages en cache. CFLRU partitionne le cache en deux espaces : une région contenant les pages les plus accédées, et l'autre les moins accédées. Dans cette dernière, les pages évincées en priorité sont celles qui contiennent les même données que sur la flash, autrement dit elles sont simplement libérées du cache. Cela réduit considérablement les coûts de remplacement en cache.
Flash Aware Buffer (FAB) [2] place une liste de pages dans le cache. Si les données adressées par une requête sont présentes dans le cache, elles y sont lues / écrites. Sinon, les pages adressées en lecture sont ramenées dans le cache, et celles adressées en écriture sont écrites dans un emplacement nouvellement créé. La taille du cache utilisé avec FAB est comprise entre 1 et 32 MO.
Block Padding Least Recently Used (BPLRU) [4] présente un tampon utilisé seulement pour les requêtes d'écriture, les requêtes de lectures étant directement envoyées vers la FTL. BPLRU utilise une liste de blocs avec une politique d'éviction LRU. Un bloc récemment écrit de manière séquentielle est déplacé vers la queue de la liste LRU. La taille du cache varie entre 8 et 16 MO.
Block-Page Adaptative Cache (BPAC) [7] partitionne le cache en deux parties : une liste LRU de pages, utilisée pour stocker les données avec une haute localité temporelle, et une liste de blocs divisée elle même en deux ensembles organisés par ancienneté et par taille. Un ensemble est utilisé pour les données avec une haute localité spatiale, et l'autre stocke les données avec une faible localité spatiale. La taille du cache de BPAC varie entre 8 et 128 MO.
Cold and Largest Cluster policy (CLC) [3] travaille sur des ensembles (*clusters*) de pages de tailles variables, répartis dans une liste LRU de *clusters* de tailles variables, et des listes de *clusters* de tailles fixes (une liste par taille). Lorsqu'un *cluster* est généré / accédé, il est placé en tête de la liste LRU. Les *clusters* à évincer sont les clusters les plus grands et les moins accédés, choisis dans les listes de tailles fixes.

L'ensemble des systèmes de cache cités plus haut sont implémentés en appui sur des FTLs existantes (coût important) et ont des tailles conséquentes comparées à celui proposé dans cette étude.

3. Architecture et politiques du système C-lash

3.1. Architecture du double cache C-lash

C-lash est un cache partitionné en deux espaces distincts : un espace de pages (*p-space*) et un espace de blocs (*b-space*). Le p-space est un ensemble de pages qui peuvent appartenir à différents blocs de la mémoire flash, tandis que le b-space est constitué de blocs qui peuvent être directement mappés sur la flash, comme on peut le voir sur la figure 1. Ce système de double cache permet de représenter les deux granularités des opérations exécutées sur la mémoire flash : lecture et écriture de pages, et effacement de blocs. Les pages dans le p-space et les blocs dans le b-space ont respectivement les mêmes tailles que les pages et blocs flash sous-jacents.
C-lash est également hiérarchique, il possède deux niveaux de politiques d'éviction : l'une évinçant des pages du p-space vers le b-space, (G sur la figure 1), et l'autre évinçant les blocs du b-space vers la flash.

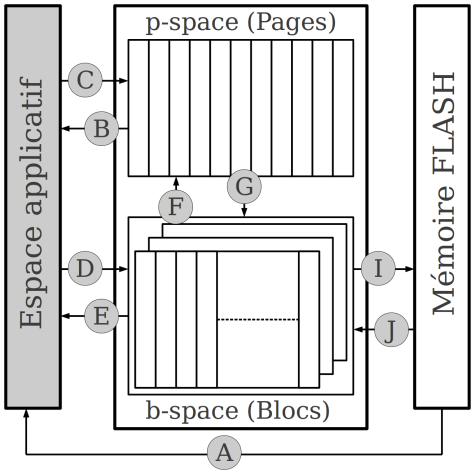

FIGURE 1 – Structure du système C-lash

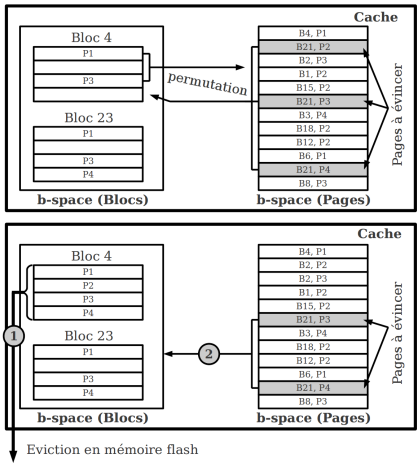

FIGURE 2 – Exemples de différents scénarios de politiques d'éviction du p-space et du b-space

Cet algorithme garantit que ce sont toujours des blocs, et non des pages, qui sont écrits sur la flash.
Les régions p-space et b-space contiennent toujours soit des pages/blocs valides, soit des pages/blocs libres respectivement. Par conséquent, lors de l'arrivée d'une requête de lecture, la donnée demandée est cherchée dans les deux espaces. Si elle s'y trouve, la donnée est lue dans le cache (B ou E sur la figure 1). Dans le cas contraire, la donnée est lue depuis la mémoire flash. En aucun cas un *cache miss* en lecture ne génère de copie de données de la flash vers le cache. Lors de l'arrivée d'une opération d'écriture, si les pages adressées sont présentes dans le p-space ou le b-space, elles sont écrasées (respectivement C et D sur la figure 1) sans impact sur la mémoire flash, ce qui évite une opération coûteuse d'effacement puis d'écriture (et de fusion si la donnée est modifiée). Si les données à écrire ne sont pas présentes dans le cache, elle ne peuvent être écrites que dans le cache de premier niveau, c'est à dire le p-space (C sur la figure 1). S'il y a assez de pages libres, les données y sont écrites. Dans le cas contraire, certaines pages sont choisies pour être évincées du p-space vers le b-space (G sur la figure 1), et les nouvelles données sont écrites dans les emplacements nouvellement libérés.

3.2. Politiques d'éviction de C-lash

3.2.1. Politique d'éviction du p-space

Comme énoncé plus haut, le p-space contient des pages écrites appartenant à différents blocs de la mémoire flash. Lors de l'arrivée d'une requête d'écriture adressant une page qui n'est présente ni dans le p-space, ni dans le b-space, une nouvelle page doit être allouée. Si une page libre est disponible, elle est écrite et la page correspondante sur la flash est invalidée. S'il n'y a pas de place libre dans le p-space, le système doit choisir une ou plusieurs pages à évincer vers le b-space (et non pas vers la mémoire flash). Le choix des pages à évincer est fait en deux étapes. Tout d'abord, le système cherche dans le p-space le plus grand ensemble de pages appartenant au même bloc. Ensuite, deux cas se présentent : 1) Un bloc libre est disponible dans le b-space, dans ce cas l'ensemble de pages victimes est copié vers ce bloc. 2) Aucun bloc n'est libre dans le b-space, dans ce cas le nombre de pages de l'ensemble victime est comparé au nombre de pages valides dans chaque bloc du b-space : s'il se trouve un bloc contenant moins de pages valides que l'ensemble victime, une opération de permutation est exécutée dans la SRAM (flèches F et G sur la figure 1). Cela signifie que les pages dans le bloc victime du b-space sont déplacées vers le p-space, alors que l'ensemble de pages victimes du p-space est déplacés par le bloc ainsi libéré dans le b-space. Cela n'a aucun impact sur la mémoire flash. La seconde possibilité survient lorsque tous les blocs dans le b-space contiennent plus de pages valides que l'ensemble de pages victimes à évincer du p-space. Dans ce cas, la politique d'éviction du b-space est exécutée pour évincer un bloc du b-space vers la mémoire flash.
Dans la figure 2 on peut voir un exemple d'éviction du p-space ; les pages choisies sont celles correspondantes au bloc B21 contenant le plus grand nombre de pages. Les deux blocs du b-space ne contiennent

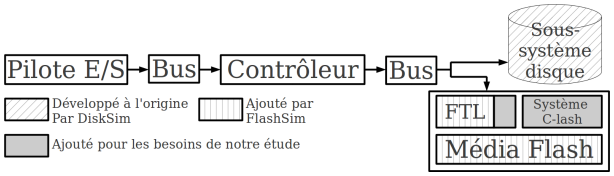

FIGURE 3 – Architecture du système de stockage simulé

chacun que deux pages valides, ce qui est inférieur aux 3 pages à évincer. Ainsi, le système C-lash effectue une permutation entre le bloc avec le plus petit nombre de pages et l'ensemble de 3 pages. Ainsi, le système libère une page sans évincer de données sur la mémoire flash.
La politique d'éviction du b-space est exécutée lorsque le système doit écrire un ensemble de pages victimes dans le b-space, et que les blocs du b-space sont pleins. Dans ce cas, C-lash écrit un bloc entier du b-space en mémoire flash, dans le but de libérer un bloc dans le cache. Un algorithme LRU est utilisé pour cette éviction. Grâce à cet algorithme, le système prend en considération la localité temporelle présentée par de nombreuses charges d'E/S.
Lorsqu'un bloc est évincé, tout le bloc correspondant est effacé de la mémoire flash, puis remplacé par celui venant du cache. Dans le cas ou le bloc flash contient toujours des pages valides, une opération de fusion (J sur la figure 1) doit être effectuée. Cette opération consiste à lire les pages toujours valides de la flash et non présentes dans le cache, et à les copier dans le cache, avant d'évincer le bloc entier. Cette opération de lecture peut être menée lorsque les pages sont déplacées du p-space vers le b-space, nous l'appelons *fusion anticipée*, ou juste avant l'éviction d'un bloc vers la mémoire flash, nous l'appelons *fusion tardive*. Les fusions tardives sont plus avantageuses sous une trace intensive en lecture et présentant de la localité temporelle et/ou spatiale car nous avons plus de chance de bénéficier des données en cache. Si la trace est intensive en écriture, la fusion anticipée ne présente pas d'avantages, on préfère alors retarder au maximum les opérations de fusion en utilisant la politique de fusion tardive. Ce faisant, on assure deux optimisations principales : 1) On effectue moins de lectures car entre le moment ou les pages sont évincées du p-space, et celui ou elles sont évincées sur la flash, de nombreuses pages peuvent être écrites et donc invalidées en mémoire flash (inutile de les lire lors de la fusion). 2) Comme il est possible pour un bloc dans le b-space d'être déplacé dans le p-space durant l'éviction du p-space, faire l'opération de fusion trop tôt peut être contraignant. Cela peut causer des lectures supplémentaires. On restreint le champ de l'étude actuelle à l'utilisation d'une politique de fusion tardive.
Un exemple d'éviction de bloc est présenté sur la figure 2. Dans l'illustration du bas, on décrit une éviction du p-space menant à une éviction du b-space et une écriture sur la mémoire flash. Dans la phase d'éviction du p-space, les pages choisies comme victimes sont celles appartenant au bloc 21. Les blocs dans le b-space contiennent plus de pages valides que le nombre de pages à évincer du p-space. Ainsi, le système doit tout d'abord évincer un bloc vers la flash. Après cela, le système copie les deux pages du B21 vers le bloc nouvellement libéré dans le b-space. Dans cet exemple spécifique, il n'y a pas d'opération de fusion car toutes les pages du bloc à évincer du b-space sont valides. Dans le cas contraire, une opération de lecture est exécutée pour fusionner les pages flash valides avec les pages valides du bloc en cache, avant de terminer l'éviction.

4. Evaluation de performances

Les performances de C-lash sont comparés, dans cette section, avec celles d'FTL récentes.

4.1. Environnement de simulation

FlashSim [12] est un simulateur à événement discret basé sur DiskSim [13], le plus utilisé des simulateurs de systèmes de stockage à base de disques durs. DiskSim simule l'ensemble du système de stockage, du pilote de périphérique jusqu'aux mouvements détaillés du disque, intégrant de nombreuses stratégies de contrôleur, files d'attentes, caches, bus, ainsi que la description détaillée du disque en termes de latences et de débit. Il inclut également un générateur de trace synthétique très détaillé, utilisé dans

une partie de nos simulations. DiskSim n'inclut pas de support de mémoires flash en natif. FlashSim intègre certains modules spécifiques à la simulation de sous-systèmes de mémoires flash. Il rend possible la simulation d'infrastructures de périphériques flash basiques pour l'implémentation des opérations spécifiques : lecture, écriture, effacement, etc. Des mécanismes de traduction d'adresses logiques vers des adresses physiques sont également implémentés avec des politiques de ramasse-miettes. FlashSim implémente de nombreuses politiques de FTL : FAST, DFTL, et une FTL idéale (une traduction par page).
Nous avons modifié et augmenté les fonctionnalités de FlashSim pour permettre la simulation d'un sous système de double cache, comme montré sur la figure 3. Le cache utilisé est configurable et de nombreuses politiques d'éviction ont été développées pour le b-space (FIFO, LRU, LFU).

4.2. Métriques de performances

Nous nous basons sur deux métriques principales : le temps de réponse moyen par requête, et le nombre d'opérations d'effacement. Le temps de réponse est mesuré au niveau du pilote d'E/S (voir figure 3), incluant toutes les latences intermédiaires : caches, contrôleurs, files d'attentes, etc. Nous avons essayé de minimiser l'impact des éléments intermédiaires, pour se concentrer sur le comportement du système flash. La seconde métrique est le nombre d'opérations d'effacement exécutées. Cette mesure montre l'usure de la mémoire. Nous démontrerons que le système C-lash améliore non seulement le temps de réponse, mais qu'il réduit l'usure aussi en absorbant les effacements dans le cache.

4.3. Traces simulées

Nous avons utilisé différents ensembles de traces réelles et synthétiques pour étudier l'impact du système C-lash, et le comparer aux différentes FTLs simulées. Le choix de la comparaison à des FTLs et non pas à des systèmes à base de FTL et de cache est dû au fait que C-lash soit conçu pour remplacer les FTL actuelles d'une part, et d'autre part, à cause de la différence énorme en termes de taille entre notre systèmes (512KO) et les systèmes à base de cache (64MO en moyenne). En plus des applications intensives en écriture, nous présentons également des résultats basés sur des traces en lecture intensive.
Pour ce qui est des tests de performances sur traces réelles, nous avons utilisés des traces d'E/S d'applications OLTP s'exécutant dans 'une institution financière disponibles grâce au *Storage Performance Council* (SPC) [22]. Nous avons également simulé des traces de moteurs de recherche [21]. Ces traces sont extraites de sous systèmes de stockage de 5 à 20 disques. Nous avons isolé les requêtes E/S pour chaque disque, et appliqué chaque trace sur le SSD simulé. Nous n'avons pas pris en considération l'ensemble des disques, plus spécifiquement ceux qui généraient un fort taux d'écritures aléatoires, ce qui est hors du champ d'étude de cet article. Les traces simulées sont récapitulées dans la table 1.
Les traces synthétiques utilisées ont été produites par le simulateur synthétique de DiskSim. Ce dernier

Taux de séquentialité		**Taux de localité spatiale**	
Par défaut	Autres	Par défaut	Autres
80 %	50 %, 65 %	40 %	30 %, 10%

Taux d'écriture		**Taille de flash simulée**	
Par défaut	Autres	Par défaut	Autres
80 %	50 %, 65 %	1 GO	0,5 GO, 2 GO

Traces	**Taux d'écriture**	**Taux de séquentialité**	**Taille moyenne de requête (KO)**
Financial 1	99%	83%	4
Financial 1	81%	81%	4
Financial 1	18%	13%	11,5
Financial 2	95%	30%	44
Financial 2	24%	1%	2,5
Financial 2	0%	-	7,5
Websearch 1	0%	-	14,5
Websearch 2	1%	-	15
Websearch 3	1%	-	15,5

TABLE 1 – Paramètres des traces synthétiques et réelles testées

permet de générer des charges d'E/S basées sur un taux de requêtes séquentielles, de lecture/écriture, de localité spatiale, de criticité temporelle, etc [13].

4.4. Tests effectués

Nous avons simulé une mémoire flash NAND de taille variant entre 100 MO et 2 GO. La taille des pages est de 2 KO et celle des blocs de 128 KO (+4 KO de métadonnées). L'opération de lecture d'une page

prend 130,9 µs, l'opération d'écriture prend 405,9 µs et l'opération d'effacement d'un bloc prend 2 ms. Cette configuration est basée sur une mémoire flash NAND Micron [23].
Dans les ensembles de tests effectués, nous comparons les performances d'une configuration de C-lash avec les différentes FTLs simulées : FAST, DFTL, et la FTL idéale à *page mapping*. Comme souligné précédemment, la FTL à correspondance de page consomme une très grande quantité de mémoire SRAM mais donne des performances idéales. La configuration de C-lash est définie à 2 blocs (128 KO chacun) et 128 pages, ce qui constitue une taille totale de 512 KO. Deux blocs dans le b-space est un minimum permettant d'absorber la localité temporelle.

5. Résultats et analyses

Comme indiqué dans la table 1, nous présentons deux ensembles de résultats d'évaluation de performances : un premier comparant les différentes FTLs avec C-lash sous des traces synthétiques, et un second utilisant des traces réelles d'entreprise.

Simulation de traces synthétiques. Dans cette partie nous décrivons le comportement de C-lash comparé aux autres FTLs en faisant varier différents paramètres.

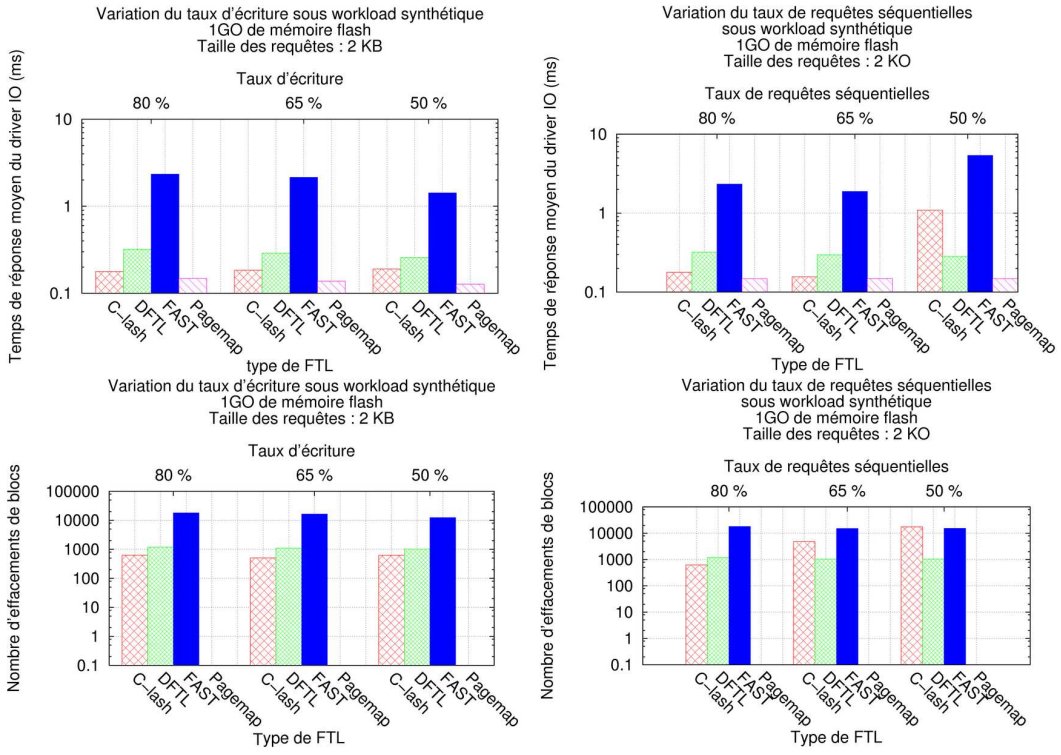

FIGURE 4 – Temps de réponse (en millisecondes) et nombre d'effacements pour la configuration de traces synthétiques décrite dans la table 1 : variation des taux d'écriture et de séquentialité des requêtes

La figure 4 montre les variations du temps de réponse et du nombre d'effacements par rapport au taux d'écriture et de séquentialité de la charge d'E/S. Les illustrations sur la gauche démontrent que plus on augmente le taux de requêtes d'écriture, plus le système C-lash surpasse les autres FTLs et se rapproche des performances idéales de la FTL à *page mapping*. Pour un taux d'écritures de 80 %, C-lash améliore le temps de réponse de DFTL de plus de 43 %, et réduit le nombre d'effacements de plus de 47 %. Pour ces simulations, le système surpasse dans tous les cas DFTL et FAST, et s'approche de la FTL à *page mapping*. On peut aussi observer que FAST donne de très faibles performances sur des traces séquentielles.
Le second paramètre de la trace que l'on fait varier est le taux de séquentialité, comme on peut le voir

sur la droite de la figure 4. Au niveau du temps de réponse moyen, C-lash surpasse DFTL et FAST pour des taux de séquentialité supérieurs à 65 % (jusqu'à 48 % d'amélioration en termes de temps de réponse, et 54 % en termes d'opérations d'effacement), mais entraîne de mauvaises performances lorsque le taux de séquentialité est inférieur à 60 % pour des requêtes de petites tailles. On peut, de plus, observer que pour un taux de séquentialité de 65 %, C-lash donne un meilleur temps de réponse tout en générant plus d'opérations d'effacements que DFTL. La décroissance de la localité spatiale génère plus d'opérations d'évictions, ce qui augmente le nombre d'opérations d'effacement, mais C-lash bénéficie toujours d'un faible temps de réponse car les écritures sont reportées dans le cache.

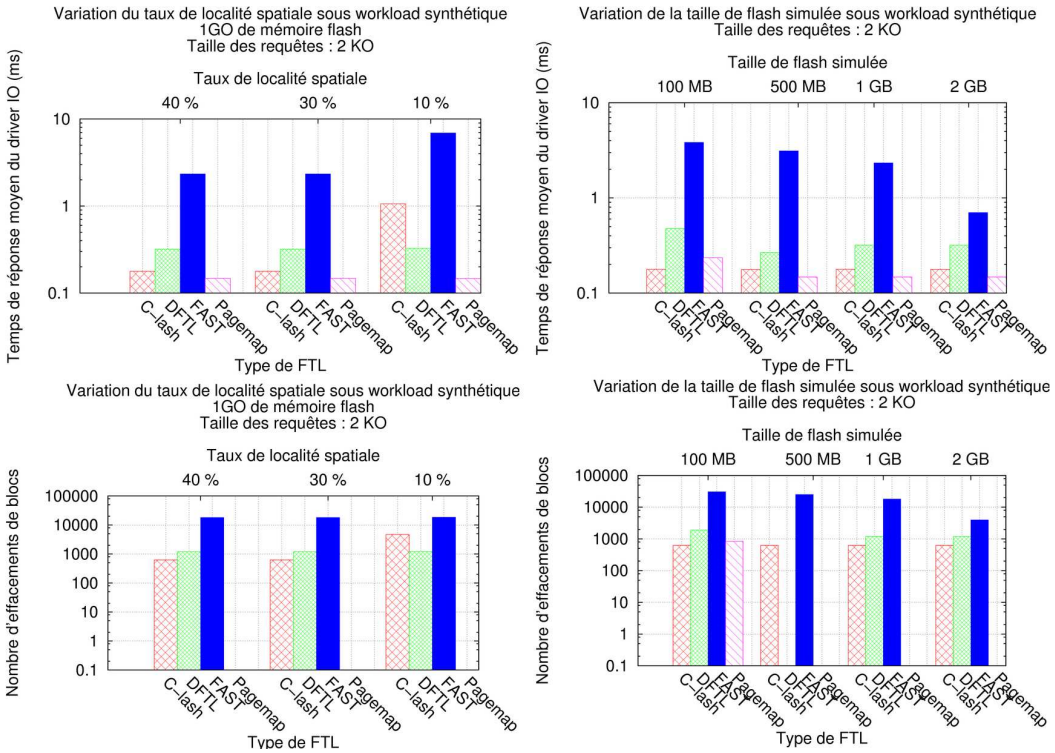

FIGURE 5 – Temps de réponse (en millisecondes) et nombre d'effacements pour la configuration de traces synthétiques décrite dans la table 1 : variation de la localité spatiale et de la taille de flash simulée

La figure 5 montre l'évolution des deux métriques de performances lorsque l'on fait varier la localité spatiale et la taille de mémoire flash simulée. On peut observer que pour des localités spatiales supérieures à 30 %, C-lash surpasse les autres FTLs (plus de 40 % d'amélioration), alors que l'on obtient de faibles performances pour un taux de 10 %. La localité spatiale est un paramètre différent de la séquentialité du point de vue du simulateur : la séquentialité représente des requêtes avec des adresses strictement contigües, tandis que la localité spatiale représente des requêtes avec des adresses voisines.
Les illustrations sur la droite de la figure 5 décrivent les variations de performances par rapport à la taille de mémoire flash simulée. On peut noter que le temps de réponse moyen du système C-lash est toujours meilleur que celui des autres FTLs, avec une augmentation de performances comprise entre 35 % et 63 %. C-lash génère toujours le même nombre d'effacements, indépendamment de la taille de la mémoire flash, tandis que les autres FTLs sont moins stables. Par exemple, DFTL ne génère aucune opération d'effacements pour 500 MO de flash, nous n'avons pas réussi à expliquer un tel résultat.
En termes de charges d'E/S synthétiques, C-lash donne de meilleures performances par rapport aux autres FTLs pour des traces présentant un taux de séquentialité supérieur à 60 %, et d'au moins 20 % de localité spatiale. On peut obtenir une importante hausse de performances, jusqu'à 65 % en termes de temps de réponse et de nombre d'effacements, comparé à la seconde meilleure FTL qui est DFTL.

Traces réelles d'applications OLTP et de moteurs de recherches. Dans cette partie on décrit les résultats

des simulations menées avec les traces réelles présentées dans la table 1. On montre les résultats obtenus sur trois disques représentatifs (présentant un bon taux de séquentialité) de chacune des traces *financial* 1 et *financial* 2, et d'un des disques de chacune des traces *websearch*.
La figure 6 montre que pour chacun des disques testés, les performances de C-lash sont de loin meilleures par rapport aux autres FTLs. Elles surpassent même celles de la FTL idéale à *page mapping* pour les deux premiers disques. En termes de temps de réponse moyen, C-lash améliore le temps de réponse de FAST de plus de 95 %, celui de DFTL de plus de 53 %, 51 % et 13 % respectivement pour les trois disques simulés. Du point de vue des opérations d'effacement, C-lash génère moins d'effacements que toutes les autres FTLs pour le premier disque, et réduit ce nombre de plus de 57 % pour DFTL au niveau du second disque. Pour ce qui est du troisième, en dépit du fait que C-lash donne un meilleur temps de réponse, il produit plus d'opérations d'effacements. Ces résultats confirment que C-lash surpasse les FTLs testées pour des charges d'E/S hautement séquentielles (disque 0 : 83 % et disque 19 : 81 %).

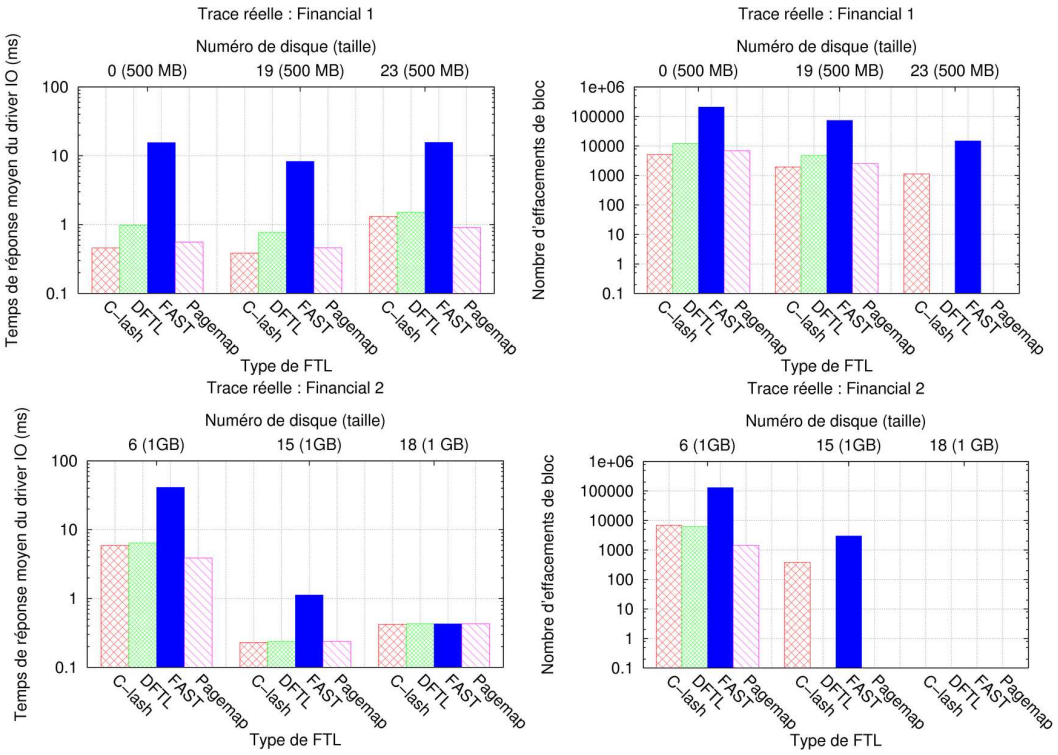

FIGURE 6 – Temps de réponse (en millisecondes) et nombre d'effacements pour les configurations Financial 1 et 2 décrites dans la table 1.

L'amélioration de performance donnée par C-lash sur la trace financial 2, montrée sur la figure 6, est moins importante que la précédente, même si le système surpasse les performances de toutes les autres FTLs testées sur les deux derniers disques. On observe une amélioration de moins de 10 % en temps de réponse, comparé à DFTL. La raison principale de cette faible augmentation est que le taux de séquentialité des opérations d'écriture de *financial* 2 est faible.
La figure 7 illustre les variations de performance sur une trace réelle intensive en lecture provenant d'un moteur de recherche en ligne. Même si C-lash est conçu pour optimiser spécifiquement les performances en écriture séquentielle, il est aussi performant avec des traces dominantes en lecture grâce à l'effet de cache. Ainsi, lorsque des données préalablement écrites sont adressées en lecture, cette opération est réalisée dans le cache, ce qui réduit le temps de réponse moyen. Comparé à DFTL, C-lash améliore les performances de plus de 27 % pour les trois traces simulées. En effet, DFTL subit des coûts supplémentaires, dus à la traduction d'adresse. C-lash donne également des résultats légèrement meilleurs que FAST et la FTL à *page mapping*.

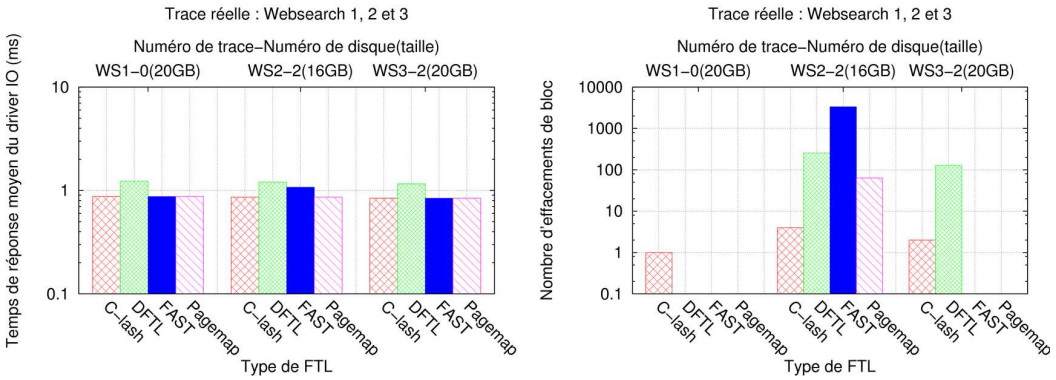

FIGURE 7 – Temps de réponse (en millisecondes) et nombre d'effacements pour la configuration Websearch décrite dans la table 1.

5.1. Réflexions sur le coût et les performances

Certains aspects liés au coût et aux performances doivent être pris en compte :

- Dans la partie évaluation de performances, le simulateur ne considère pas toutes les latences de la traduction d'adresses et du ramasse-miettes pour les différentes FTLs. Ce paramètre peut impacter négativement leurs performances, comparé à la simple traduction directe utilisée par C-lash.
- L'inconvénient principal dans le fait de réduire le nombre d'opérations d'effacement grâce à un cache est la perte de données induite par une coupure d'énergie imprévue. Ce genre de problème est très important à prendre en compte. Différentes solutions matérielles peuvent être proposées. Une petite pile peut retarder l'arrêt effectif jusqu'à l'éviction complète des données dans le cache [4]. En concevant C-lash nous avons pensé à ce problème et avons implémenté l'algorithme LRU dans le b-space (contenant un petit nombre de blocs) et pas dans le p-space, ce qui pourrait provoquer plus d'opérations d'éviction de données sur la mémoire flash. Cela assure une éviction des données plus régulière, et ainsi moins de données perdues.
- Le coût de l'utilisation d'une solution C-lash, par rapport à l'utilisation des solutions de caches + FTL existantes, est très attractif ; C-lash utilise moins de mémoire sans FTL sous-jacente. Nous pouvons encore réduire la taille de C-lash car l'intégralité des pages des blocs dans le p-space n'est pas toujours utilisée. Une solution possible est la virtualisation de l'adressage des pages dans le b-space pour s'assurer l'obtention d'un bon ratio d'usage mémoire. Cela est particulièrement vrai sous des charges d'E/S peu séquentielles, mais se fait au dépend de la complexité de la traduction.

6. Conclusion et perspectives

Dans cette étude, nous avons proposé un changement de paradigme en introduisant un système de cache pour la gestion de mémoire flash, dont le rôle est de remplacer les parties wear levelling, ramasse-miettes, et traduction d'adresses de la FTL. Nous avons démontré que C-lash augmente de manière importante les performances pour une large variété de traces synthétiques et réelles, en termes de temps de réponse et de nombre d'effacements.
C-lash prend en considération les caractéristiques des mémoires flash, les coûts des différentes opérations (lecture, écriture et effacement) en termes de temps d'accès et de durée de vie, en évinçant sur le média seulement les blocs contenant un maximum de pages valides. Cela réduit grandement le nombre d'opérations d'effacement, et améliore les performances. C-lash prend également en compte les localités temporelles et spatiales des données, via la politique d'éviction LRU du b-space.
Les performances de C-lash ont été testées sur un large ensemble de charges d'E/S, générées synthétiquement ou extraites des dépôts de benchmarks reconnus. Les expériences menées ont prouvé la pertinence et l'efficacité de C-lash pour toutes les traces présentant un bon taux de séquentialité (plus de 60 %). En effet, nous avons comparé le système avec un algorithme de FTL efficace nommé DFTL, et amélioré les performances sur les traces testées de plus de 65 % dans de nombreux cas. C-lash a également, quelque-

fois, montré meilleures performances que la FTL idéale à *page mapping*, qui consomme plus d'un ordre de grandeur en termes de mémoire SRAM.
Le système C-lash montre de faibles performances sur des charges d'E/S faiblement séquentielles avec des requêtes de petites tailles, si toutefois la taille du cache n'est pas assez importante. Cela est principalement dû à la mauvaise utilisation du b-space en cas d'accès aléatoire. Nous proposons comme perspective de travaux futurs l'étude de la possibilité de virtualiser les accès aux pages du b-space dans le but de mieux exploiter la totalité de l'espace. Une autre solution à étudier est la possibilités pour le double cache d'être reconfiguré dynamiquement en fonction de la séquentialité des traces appliquées.
Nous comptons étendre l'environnement de simulation à des systèmes multi-SSD, pour explorer plus en détail les interactions entre les différents disques de larges systèmes de stockages d'entreprise. Nous escomptons également élargir notre bibliothèque de traces avec différentes traces multimédias.

Bibliographie

1. A. Gupta, Y. Kim, B. Urgaonkar, "DFTL : A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings", ACM Architectural Support for Programming Languages and Operating Systems, Washington, 2009.
2. H. Jo, J. Kang, S. Park, J. Kim, J. Lee, "FAB : Flash-Aware Buffer Management Policy for Portable Media Players", IEEE Transactions on Consumer Electronics, 52, pp. 485- 493, 2006.
3. S. Kang, S. Park, H. Jung, H. Shim, J. Cha, "Performance Trade-Offs in Using NVRAM Write Buffer for Flash Memory-Based Storage Devices", In IEEE Transactions on Computers, 58, no.6, 2009.
4. H. Kim, S. Ahn, "BPLRU : A Buffer Management Scheme for Improving Random Writes in Flash Storage", the 6th USENIX Conference on File and Storage Technologies, CA, 2008.
5. S. Lee, D. Park, T. Chung, D. Lee, S. Park, H. Song, "A Log Buffer-based Flash Translation Layer Using Fully-associative Sector Translation", ACM Trans. Embed. Comput. Syst. 6, 3. 2007.
6. S. Park, D. Jung, J. Kang, J. Kim, J. Lee, "CFLRU : a Replacement Algorithm for Flash Memory". In Proceedings of the 2006 international Conference on Compilers, Architecture and Synthesis for Embedded Systems, Seoul, 2006.
7. G. Wu, B. Eckart, X. He, "BPAC : An Adaptive Write Buffer Management Scheme for Flash-Based Solid State Drives", In Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies, Incline village, 2010.
8. D. Myers, On the Use of NAND Flash Memory in High-Performance Relational Databases, Master of Science technical report, MIT, 2008.
9. A. M. Caulfield, L. M. Grupp, S. Swanson, "Gordon : Using Flash Memory to Build Fast, Power-efficient Clusters for Data-intensive Applications", ACM Architectural Support for Programming Languages and Operating Systems, Washington, 2009.
10. R. Stoica, M. Athanassoulis, R. Johnson, "Evaluating and Repairing Write Performance on Flash Devices", Fifth International Workshop on Data Management on New Hardware, Providence, 2009.
11. F. Chen, D.A. Koufaty, X. Zhang, "Understanding Intrinsic Characteristics and System Implications of Flash Memory based Solid State Drives", ACM SIGMETRICS/Performance, Seattle, 2009.
12. Y. Kim, B. Tauras, A. Gupta, D.M. Nistor, B. Urgaonkar, FlashSim : A Simulator for NAND Flash-based Solid-State Drives, Tech. Report CSE-09-008, Pensylvania, 2009.
13. G. R. Ganger, B. Worthington, Y. N. Patt, The Disksim Simulation Environment Version 3.0 Reference Manual, Tech. Report CMU-CS-03-102, Pittsburgh, 2003.
14. G. Forni, C. Ong, C. Rice, K. McKee, R. J. Bauer, Flash Memory Applications, In Nonvolatile Memory Technologies with emphasis on Flash, edited by Brewer, J.E. and Gill, M., IEEE Press Series on Microelectronic Systems, USA, 2007.
15. T. S. Chung, H. S. Park, "STAFF : A Flash Driver Algorithm Minimizing Block Erasures". Journal of Systems Architectures, 53(12), 2007.
16. D. Park, B. Debnath, D. Du, CFTL : A Convertible Flash Translation Layer with Consideration of Data Access Pattern, Tech Report, University of Minnesota, 2009.
17. M-Systems, Flash Memory Translation Layer for NAND Flash (NFTL), M-Systems, 1998.
18. C. H. Wu, L. P. Chang, T. W. Kuo, "An Efficient B-Tree Layer for Flash-Memory Storage Systems", ACM Trans. On Embedded Computing, Vol 6, Issue 3, 2007.
19. T. Shinohara, Flash Memory Card with Block Memory Address, United States Patent, No.5,905,993, 1999.
20. B. S. Kim, G. Y. Lee, Method of Driving Remapping in Flash Memory and Flash Memory Architecture Suitable Therfor, United States Patent, No. 6,381,176, 2002.
21. OLTP Traces from UMass Trace Repository `http://traces.cs.umass.edu/index.php/Storage/Storage` (Accessed oct. 2010).
22. Storage Performance Council, `http://www.storageperformance.org` (Accédé oct. 2010).
23. Micron, Small Block vs. Large Block NAND Flash Devices, Micron Technical Report TN-29-07, `http://download.micron.com/pdf/technotes/nand/tn2907.pdf`, 2007 (Accédé oct. 2010).