Regular paper http://dx.doi.org/10.13067/JKIECS.2013.8.10.1529

NAND 플래쉬메모리 기반 색인에 관한 연구

김동현* · 반재훈**

A Survey of the Index Schemes based on Flash Memory

Dong-Hyun Kim* · Chae-Hoon Ban**

요 약

NAND 플래쉬메모리는 적은 크기로 대용량의 데이터를 저장할 수 있고 전력소모량이 적기 때문에 스마트폰, 센서노드 등의 다양한 휴대용기기에서 사용되어 진다. 플래쉬메모리에 저장된 대용량의 데이터를 효율적으로 처리하기 위하여 색인을 사용해야 한다. 그러나 쓰기 연산의 속도가 읽기 연산보다 매우 느리고 덮어쓰기를 지원하지 않기 때문에 기존의 색인을 사용하면 성능이 저하되는 문제가 발생한다. 이 논문에서는 플래쉬메모리의 특성을 이용하여 색인을 설계한 기존의 논문들을 살펴본다. 그리고 플래쉬메모리에서 색인을 설계할 때 고려해야할 성능요소를 제시한다.

ABSTRACT

Since a NAND-flash memory is able to store mass data in a small sized chip and consumes low power, it is exploited on various hand-held devices, such as a smart phone and a sensor node, etc. To process efficiently mass data stored in the flash memory, it is required to use an index. However, since the write operation of the flash memory is slower than the read operation and an overwrite operation is not supported , the usage of existing index schemes degrades the performance of the index. In this paper, we survey the previous researches of index schemes for the flash memory and classify the researches by the methods to solve problems. We also present the performance factor to be considered when we design the index scheme on the flash memory.

키워드

Index, Flash Memory, Main Memory Buffer Scheme, Log Scheme, Index Tree
색인, 플래쉬메모리, 메인메모리버퍼 기법, 로그 기법, 색인트리

Ⅰ. 서 론

플래쉬메모리는 비휘발성 메모리칩을 이용하는 저장장치의 일종으로 전원이 꺼지더라도 저장된 데이터가 유지된다. 또한 메모리칩의 특성 상 저비용, 작은 크기로 고용량의 데이터를 저장할 수 있고 전력 소모량이 적고 데이터 읽는 속도가 매우 빠른 특성을 가지고 있다[1][2]. 따라서 플래쉬메모리는 스마트기기, 센서 노드, USB 메모리 등 여러 다양한 휴대용 기기에서 사용되고 있으며 저장용량의 크기가 계속 증가하는 추세이다.

플래쉬메모리는 메모리칩의 논리회로의 종류에 따라 NOR형과 NAND형으로 구분된다. NAND형은 블록단위로 접근하지만 단위 가격이 저렴해서 데이터

* 동서대학교 컴퓨터정보공학부(pusrover@dongseo.ac.kr)
** 교신저자 (corresponding author) : 고신대학교 인터넷비즈니스학과(chban@kosin.ac.kr)
접수일자 : 2013. 08. 27 심사(수정)일자 : 2013. 09. 23 게재확정일자 : 2013. 10. 21

저장용으로 주로 사용된다. NAND형 플래쉬메모리는 전자기 신호를 이용하여 데이터에 접근하기 때문에 저장 위치에 상관없이 데이터를 읽는 속도는 일정한 장점이 있지만 동일위치에 덮어쓰기가 안 되고 쓰는 속도가 읽는 속도에 비하여 매우 느린 단점이 있다 [1][2][3].

플래쉬메모리 저장장치의 용량이 증가함에 따라 대용량의 데이터를 저장하는 것이 가능해지기 때문에 연산 처리를 위한 색인이 필요하다. 기존의 색인 연구는 크게 디스크기반의 색인과 메모리 기반의 색인 연구로 나누어진다. 디스크기반 색인은 디스크 접근 횟수를 줄이는 것이 중점을 두고 있다. 반면에 메모리기반 색인은 색인 연산을 위한 중앙처리장치의 비용을 줄이는 것에 중점을 두고 있다. 그러나 기존의 색인을 플래시메모리 기반 저장장치에 적용하면 플래쉬메모리의 특성 때문에 색인의 성능이 저하되는 문제가 발생한다.

이 논문에서는 플래쉬메모리에 적합한 기존의 색인 연구들에 대하여 알아보고 색인을 설계할 때 살펴볼 요소를 제시한다. 이를 위하여 플래쉬메모리의 단점을 해소하기 위한 기법에 따라 현재까지의 연구들을 분류하고 연구 내용 및 장단점을 기술한다.

논문의 구성은 다음과 같다. 2장에서는 NAND 플래쉬메모리의 특성에 대하여 기술하고 3장에서는 메인메모리 버퍼를 사용한 색인 기법들을 제시한다. 4장에서는 메모리버퍼를 사용하지 않는 기법과 색인 설계 시 살펴보아야 할 성능요소에 대하여 제시하고 마지막으로 5장에서 결론을 기술한다.

II. NAND 플래쉬메모리

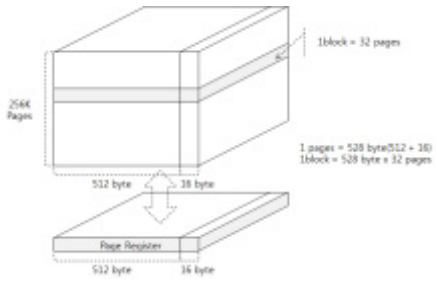


그림 1. NAND 플래쉬메모리 구조
Fig. 1 Structure of NAND flash memory

NAND 플래쉬메모리는 그림 1에서 보듯이 여러 개의 페이지와 블록으로 구성된다. 하나의 페이지는 512바이트와 16바이트의 여유영역으로 구성되며 하나의 블록은 32개의 페이지로 구성된다.

NAND 플래쉬메모리는 구성 셀의 논리회로로 인하여 자기디스크 기반의 저장장치에 비하여 다음과 같은 특성을 가지게 된다. 첫 번째는 기계적인 대기시간이 없다. 자기디스크는 기계적 장치인 헤드와 암을 이용하여 데이터에 접근하기 때문에 데이터가 저장된 위치에 따라 접근 속도가 다르다. 그러나 플래쉬메모리는 전기 신호를 사용하여 셀의 데이터를 접근하기 때문에 모든 페이지의 데이터 접근을 위한 속도가 균일하다. 두 번째는 읽기와 쓰기의 속도가 다르다. 통상적으로 읽기 연산의 속도가 쓰기 연산의 속도보다 두 배이상 빠르다.

세 번째는 동일 위치에 덮어쓰기(overwrite)를 지원하지 않는다. 동일 위치에 데이터를 덮어쓰기를 할 경우에 새로운 데이터는 자유 영역에 위치한 다른 페이지에 저장한다. 그리고 기존의 페이지는 무효화시킨다. 이러한 기법을 외부영역변경(out-place update)이라 한다. 만약 자유 영역의 페이지가 부족하게 되면 전체 메모리 페이지를 스캔하면서 무효화된 페이지를 검색한 후에 무효화 페이지를 초기화시키는 지움(erase)연산을 수행한다. 지움 연산은 페이지 단위가 아닌 블록 단위로 수행한다. 외부영역변경 기법에 의하여 데이터의 논리적인 위치와 물리적인 위치가 달라질 수 있으므로 논리적 위치를 물리적 위치로 변환하기 위한 변환테이블을 사용한다. 마지막으로 각 블록의 지움 연산 횟수는 제한이 있다. 통상적으로 1,000,000번의 지움 연산을 수행할 수 있으며 그 이후에는 해당 블록을 사용할 수 없다. 따라서 특정 위치의 블록에 지움 연산이 집중될 경우에 전체 저장장치의 수명이 줄어들 수 있다.

III. 메인메모리 버퍼를 사용한 플래쉬메모리 색인

3.1 플래쉬 기반 색인 분류

자기디스크는 데이터를 접근하기 위한 단계 중 디스크 I/O를 위한 비용이 매우 크기 때문에 디스크 기

반 색인은 주로 디스크 접근 횟수를 줄임으로써 색인의 성능을 개선하였다. 반면에 메인메모리는 상대적으로 중앙처리장치의 연산 비용이 매우 크기 때문에 메인메모리 기반 색인은 색인 연산을 위한 연산 횟수를 줄임으로써 색인의 성능을 개선하였다.

그러나 NAND 플래쉬메모리 기반의 저장장치에서 기존의 색인을 구축하면 플래쉬메모리 특성 때문에 색인의 성능이 저하되는 문제가 발생한다. 특히 덮어쓰기가 지원되지 않는 특성과 쓰기 연산 속도가 읽기 연산 속도보다 두 배이상 느린 특성은 잦은 삽입/삭제 연산으로 인하여 빈번한 쓰기 연산이 발생할 경우 치명적인 성능 저하를 유발할 수 있다.

이를 해결하기 위하여 플래쉬메모리 기반 색인들은 주로 쓰기 연산과 덮어쓰기 연산의 횟수를 줄이기 위한 기법들을 제시하였다. 첫 번째 방법은 메인메모리의 버퍼를 사용하여 변경된 색인의 내용을 임시로 저장하는 방법이다. 두 번째는 메인메모리의 버퍼를 사용하지 않고 병합기법 또는 압축 기법을 활용하여 여러 번의 쓰기 연산을 한 번의 일괄작업으로 처리하는 방법이다. 표 1은 메인메모리 버퍼의 사용 여부에 따른 플래쉬 기반 색인 연구들을 분류한 것이다.

표 1 플래쉬 기반 색인 분류
Table 1. Classification of indexes based on flash memory

Scheme	Indexes
Buffering	BFTL, BOF, R-Tree(F), MR-Tree(F), Self-Tuning B+-Tree, LA-Tree
Non-Buffering	F-Tree, FD-Tree, FAST

3.2 버퍼 사용 색인 기법

Chin H. W. et al.[1]은 B-tree를 이용한 BFTL (B-tree Flash Translation Layer) 기법을 제안하였으며 플래쉬메모리에서 디스크 기반 색인을 개선한 첫 연구이다. BFTL은 기존 B트리의 구조 및 분할/합병 알고리즘을 사용하면서 노드의 엔트리 정보들을 저장하기 위하여 색인유닛을 제시한다. 색인유닛은 노드 ID와 키값외에 부모노드와 왼쪽, 오른쪽 색인유닛을 가리키기 위한 포인터들로 구성된다.

새로운 키값의 삽입 또는 삭제 연산이 발생하면 키값이 삽입될 노드를 검색한 후에 삽입 또는 삭제될 키값이 포함된 색인 유닛을 구성한다. 그리고 구성된 색인유닛은 플래쉬메모리에 바로 쓰지 않고 유보버퍼(reservation buffer)에 임시로 저장한다. 계속된 색인유닛의 삽입으로 인하여 유보버퍼에 여유 공간이 없으면 완료정책(commit policy)에 따라 비로소 플래쉬메모리에 쓰기 연산을 수행한다.

유보버퍼내의 색인유닛들이 완료정책에 의하여 플래쉬메모리에 저장되기 때문에 동일 노드의 색인 유닛들이 플래쉬메모리의 서로 다른 페이지에 저장될 수 있다. 따라서 동일노드의 색인유닛들이 저장된 물리적 페이지를 연결하기 위하여 노드변환테이블을 사용한다. 노드변환테이블은 각 색인노드별로 노드를 구성하는 색인유닛이 저장된 물리적 페이지를 리스트형태로 관리한다.

Chin H.W et al.[2]은 플래쉬메모리에서 R-Tree를 구축하기 위한 기법을 제안하였다. R-Tree에 대한 삽입, 삭제 그리고 재균형 연산은 많은 노드들에 대하여 동일한 저장 위치에 노드 데이터를 쓰는 연산들을 필요로 한다. 디스크 기반의 저장 장치에서는 덮어쓰기가 지원되기 때문에 이러한 연산들이 효율적으로 수행된다. 그러나 플래쉬메모리에서 동일한 연산이 적용되어질 경우 필히 지움 연산을 먼저 수행해야하기 때문에 플래쉬메모리의 가용 공간이 급격히 감소하고 성능이 저하되는 문제가 발생한다.

이를 개선하기 위하여 BFTL 기법[1]과 유사한 유보버퍼를 사용한다. 데이터의 삽입 또는 삭제 연산이 발생하면 이를 위한 객체를 생성하고 유보 버퍼에 저장한다. 유보버퍼의 객체는 크게 메타 데이터와 데이터로 구성되며 메타 데이터는 데이터 객체에 대한 최소경계사각형, 연산의 종류 그리고 데이터에 대한 포인터로 구성된다. 유보버퍼가 가득 차면 각각의 객체는 색인의 노드 엔트리에 해당하는 색인 유닛으로 변경된 다음에 플래쉬메모리의 페이지에 저장된다. 이 경우에 동일 노드의 엔트리 정보들이 플래쉬메모리의 동일 페이지에 함께 저장되지 않기 때문에 동일 노드의 엔트리 정보와 실제 물리적 페이지 정보를 연결하기 위하여 노드변환테이블을 사용한다.

Jung H.N et al.[3]은 BFTL 기법을 개선한 BOF(B tree On Flash memory) 기법을 제안하였다. BFTL 기법은 색인유닛이 여러 페이지에 나누어 저장되어있

기 때문에 노드에 접근할 때마다 노드변환테이블을 이용하여 노드를 구성해야 하는 부담이 발생한다. 이를 개선하기 위하여 BOF는 동일한 색인유닛을 사용하지만 동일노드에 속하는 유보버퍼의 색인 유닛들은 플래쉬메모리의 동일 페이지에 저장한다. 따라서 한 페이지에 하나의 색인 노드 정보만을 저장하기 때문에 한 번의 읽기 연산을 통하여 한 노드의 정보를 읽을 수 있고 노드변환 테이블을 사용할 필요가 없다.

Hyun S. L. et al.[4]은 트리 색인의 일종인 MR-tree를 적용하기 위한 기법을 제시하였다. MR-tree는 R-tree의 기본 구조를 사용하면서도 R-tree와는 달리 서브 트리간의 높이가 다른 불균형 트리 구조이다. [4]에서는 플래쉬메모리의 쓰기 연산 횟수를 줄이기 위하여 색인의 수정, 즉 노드의 삽입/삭제 연산을 모아서 일괄 처리하는 기법을 제시한다. 노드의 삽입/삭제 연산의 내용을 메인메모리 버퍼에 모은 다음에 플래쉬메모리의 페이지 크기가 되었을 때 비로소 플래쉬메모리에 저장한다.

Suman N. et al.[5]은 적용되는 환경에 따라 유동적으로 두 가지 종류의 노드로 전환 가능한 Self-Tuning B+-tree를 제안하였다. [5]에서는 트리의 노드에 대하여 D와 L의 모드를 설정한다. 노드가 D 모드인 경우 읽기/쓰기 연산을 B+-tree(D)와 동일하게 수행한다. 반면에 L모드인 경우 읽기/쓰기 연산을 B+-tree(L)와 동일하게 수행한다. L모드를 위하여 로그버퍼(Log Buffer)와 노드변환테이블을 사용한다. 로그버퍼는 L모드의 노드 엔트리가 변경되면 변경된 데이터를 저장한다. 로그버퍼의 크기는 플래쉬메모리의 페이지의 크기이며 로그버퍼가 가득 차면 플래쉬메모리에 쓰기 연산을 수행한다. 그리고 노드변환테이블을 사용하여 노드 엔트리의 페이지 위치를 관리한다.

Devesh A. et al.[6]은 노드의 쓰기 작업을 일괄 처리하기 위하여 메인메모리 버퍼를 사용하는 LA-tree(Lazy Adaptive Tree)를 제시하였다. 트리의 노드에 대하여 삽입 또는 삭제 연산이 발생하면 해당 노드에 메인메모리 버퍼를 할당하고 변경 내용을 할당된 버퍼에 저장한다. 트리에 대하여 삽입/삭제 연산이 발생하면 새로운 키값은 루트 노드의 버퍼에 삽입된다. 그리고 ADAPT 알고리즘을 사용하여 버퍼를 비울 것인지 결정한다. 만약 버퍼를 비우기로 하면 버퍼의 모든 엔트리를 하위레벨의 버퍼에 삽입한다. 이러한 버퍼 비우기는 단말노드까지 재귀적으로 수행된다. 만약 단말 노드의 버퍼를 비우기로 하면 단말노드의 버퍼를 정렬한 다음에 왼쪽 선두 단말노드부터 시작하여 버퍼와 단말노드를 병합한다.

IV. 버퍼를 사용하지 않는 플래쉬메모리 색인

4.1 메인메모리 버퍼 비사용 색인 기법

Siwoo B. et al.[7]은 단방향재쓰기(one-way rewrite) 기법을 활용한 F-tree(Flash memory based Tree)를 제안하였다. 플래쉬메모리의 지움 연산은 블록의 각 비트를 1로 초기화하는 연산이며 쓰기 연산은 비트의 1을 0으로 바꾸는 연산이다. 단방향재쓰기 기법은 쓰기 연산의 특성을 이용하여 쓰기 연산이 수행된 페이지이지만 지움 연산을 수행하지 않고 특정 비트의 1의 값을 0으로 재쓰기 하는 기법이다.

재쓰기 기법을 활용하여 B+-tree 기반의 F-tree는 각 엔트리의 정보를 압축한다. B+-Tree에서 가장 효율적인 성능을 보여주는 노드의 사용률이 66%이기 때문에 LZO-3 알고리즘을 이용하여 노드 엔트리 자료를 압축하면 각 노드에 대하여 52%의 비어있는 공간을 확보할 수 있다. 만약 노드에 첫 쓰기 연산이 발생하면 노드 자료를 압축하여 빈 공간을 확보한다. 그리고 동일노드에 대하여 두 번째 쓰기 연산 즉, 덮어쓰기 연산이 발생하면 비어있는 공간에 재쓰기 연산을 수행하여 지움 연산의 횟수를 줄인다.

Yinan L. et al.[8]은 병합 기법을 사용하는 FD-tree를 제안하였다. FD-tree는 여러 레벨인 L_0부터 L_i로 나누어지는 런으로 구성된다. 하나의 런은 키값을 이용하여 정렬된 데이터를 보유하며 L_i의 런들은 플래쉬메모리의 연속된 페이지에 저장된다. 각 런은 엔트리로 구성되며 각 엔트리는 키값과 레코드 식별자의 조합인 색인엔트리와 키값과 하위레벨의 런에 대한 포인터의 조합인 경계(fence)로 분류된다. 그리고 최상위 레벨인 L_0는 헤드트리라고 하며 여러 개의 B+-tree로 구성된다.

Mohamed S. et al.[11]은 새로운 색인 구조를 제시하지 않고 기존의 B-tree, R-tree를 플래쉬메모리에서 사용하기 위한 프레임워크인 FAST(Flash Aware

Spatial Tree index structure)를 제안하였다. 메인메모리 버퍼는 시스템 오류가 발생할 경우 색인의 연속성을 보장할 수 없기 때문에 FAST는 트리변경테이블과 로그파일을 사용하였다. 트리변경테이블은 메인메모리에 위치한 해쉬 테이블로서 플래쉬메모리에 반영되지 않는 트리의 변경 내용을 저장한다. 로그 파일은 플래쉬메모리에 복구 목적으로 예약된 공간으로 노드 트리에 대한 삽입, 삭제, 변경 연산들에 대한 로그 정보를 저장한다.

4.2 플래쉬메모리 색인 개발 시 고려해야 할 성능 요소

색인연산을 위한 비용은 크게 두 가지로 나누어진다. 첫 번째는 색인 구조에 따라 삽입/삭제 연산 처리를 위한 중앙처리장치의 연산 비용이다. 두 번째는 저장장치에 저장된 자료를 접근하기 위한 접근비용이다. 플래쉬메모리 기반의 색인도 특성에 따라 많은 비용이 소요되는 부분을 개선해야 전체 성능을 향상할 수 있다. 저장장치로써의 특성은 다음과 같다. 첫째, 모든 셀에 동일한 속도로 접근할 수 있다. 둘째, 쓰기 연산 속도가 읽기 연산 속도가 두 배이상 느리다. 셋째, 덮어쓰기 연산이 지원되지 않는다. 동일 위치에 자료를 저장하기 위해서는 필히 지움 연산을 먼저 해야 하나 지움 연산은 블록 단위이기 때문에 연산의 속도가 매우 느리다. 넷째, 지움 연산의 횟수가 고정되어 있다.

트리 색인의 구조 상 노드에 대한 읽기와 쓰기 연산의 성능에 따라 색인의 성능이 결정된다. 그러므로 기존의 색인 연구는 쓰기 연산의 비용을 개선하기 위한 기법들을 제시하였다. 쓰기 연산의 횟수를 줄이기 위하여 메인메모리의 버퍼를 사용하거나 또는 병합기법 이나 로그기법을 사용하였다.

따라서 플래쉬메모리 저장장치 기반의 색인을 설계할 때 고려해야 할 성능 요소는 다음과 같다. 첫째, 쓰기 연산의 횟수이다. 저장장치로의 쓰기 연산 횟수는 매우 중요한 성능 요소이다. 둘째, 덮어쓰기 연산의 횟수이다. 덮어쓰기 연산의 속도가 다른 연산에 비하여 두 자리 수 이상 느리므로 매우 중요한 성능 요소가 될 수 있다. 셋째, 읽기 연산을 위한 노드 크기이다. 향후 설계하고자 하는 색인이 데이터 유도 방식인지 또는 공간 분할 방식인지의 종류와는 무관하게 성능 개선을 위해서는 세 가지 요소에 대한 고려가 필요하다.

V. 결 론

플래쉬메모리는 휴대하기 좋고 크기에 비하여 많은 용량의 자료를 안정적으로 저장할 수 있기 때문에 스마트폰, MP3 등의 휴대용 기기에서 주로 사용되어지고 있으며 저장용량이 커짐에 따라 자기디스크 저장장치를 대체할 것으로 예상된다. 따라서 플래쉬메모리 저장장치에서 대용량 데이터가 저장될 경우에 데이터에 대한 연산을 신속히 처리하기 위해서 색인을 사용해야 한다. 그러나 디스크나 메인메모리와는 다른 특성을 가지고 있기 때문에 기존의 색인을 사용할 없는 문제가 있다.

이 논문에서는 플래쉬메모리 기반의 색인에 대한 기존의 연구들을 살펴보았다. 기존의 연구들은 주로 트리 기반 색인에 대하여 수행되었으며 쓰기 연산의 횟수를 줄이기 위한 기법들을 제시하였다. 이는 쓰기 연산의 속도가 읽기 연산에 비하여 매우 느리기 때문에 고비용이 소요되어 전체 색인의 성능을 저하시키기 때문이다. 이를 개선하기 위하여 주로 메인메모리 버퍼를 사용하여 쓰기 연산의 횟수를 줄였으며 일부 연구에서는 메인메모리 버퍼를 사용하지 않고 압축기법과 병합 기법을 사용하여 횟수를 줄였다.

향후 연구로는 대부분의 기존 연구가 트리 기반의 색인에 대하여 수행되었으므로 데이터 공간 분할 기반의 색인에 대한 연구가 필요하다.

감사의 글

본 연구는 2012년도 동서대학교 학술연구조성비 지원으로 수행되었음.

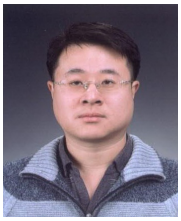
참고 문헌

[1] Chil-Hsien Wu, Li-Pin Chang, Tei-Wei Kuo, “An Efficient B-Tree Layer for Flash Memory Storage System”, Real-Time and Embedded Computing Systems and Applications, LNCS 2968, Springer, pp. 409-430, 2003.

[2] Chin-Hsien Wu, Li-Pin Chang, Tei-Wei Kuo, "An Efficient R-Tree Implementation over Flash-Memory Storage Systems", ACM Intl. Symp. on Advances in Geographic Info-

rmation Systems, ACM, pp. 17-24, 2003.
[3] Junghyun Nam, Dong-Joo Park, "The Efficient Design and Implementation of the B-Tree on Flash Memory", Proc. of the 25th KISS Fall, pp. 55-57, 2005.
[4] Hyun Seung Lee, Ha Yoon Song, Kyung-Chang Kim, "Performance of Index trees on Flash Memory", Intl. MultiConf. on Computer Science and Information Technology, pp. 725-734, 2007.
[5] Suman Nath, Aman Kansal, "FlashDB: Dynamic Self-tuning Database for NAND Flash", Intl. Conf. on Information Processing in Sensor Networks, pp. 410-419, 2007.
[6] Devesh Agrawal, Deepak Genesan, Tamesh Sitaraman, Yanlei Diao, Shasho Singh, "Lazy-Adaptive Tree: An Optimized Index Structure for Flash Devices", Proc. of the VLDB Endowment, pp. 361-372, 2009.
[7] Siwoo Byun, Moonhaeng Huh, Hoyoung Hwang, "An Index rewriting schemes using compression for flash memory database system", Journal of Information Science, pp. 1-18, 2007.
[8] Yinan Li, Bingsheng He, Qiong Luo, Ke Yi, "Tree Indexing on Flash Disk", Proc. of ICDE, pp. 1303-1306, 2009.
[9] Mohamed Sarwat, Mohamed F. Mokbel, Xun Zhou, Suman Nath, "FAST : A Generic Framework for Flash-Aware Spatial Trees", Intl. Conf. on Advances in Spatial and Temporal Databases, pp. 149-167, 2011.
[10] Yong-Yoong Chai, "An Analog Content Addressable Memory implemented with a Winner-Tak-All Strategy", The Journal of The Korea Institute of Electronic Communication Sciences, Vol. 8, No. 1, pp. 105-111, 2013.
[11] Yong-Yoong Chai, "An Analog Memory Fabricated with Single-poly Nwell Process Technology", The Journal of The Korea Institute of Electronic Communication Sciences, Vol. 7, No. 5, pp. 1061-1066, 2012.
[12] Chi-Yeon Kim, "A Range Query Method using index in Large-scale Database Systems", The Journal of The Korea Institute of Electronic Communication Sciences, Vol. 7, No. 5, pp. 1095-1101, 2012.

저자 소개



김동현(Dong-Hyun Kim)

1995년 부산대학교 컴퓨터공학과 졸업(공학사)
1997년 부산대학교 대학원 컴퓨터공학과 졸업(공학석사)
2003년 부산대학교 대학원 컴퓨터공학과 졸업(공학박사)
2004년~현재 동서대학교 컴퓨터정보공학부 교수
※ 관심분야 : 데이터베이스, 공간 데이터베이스, GIS, 센서데이터베이스

반재훈(Chae-Hoon Ban)

1997년 부산대학교 컴퓨터공학과 졸업(공학사)
1999년 부산대학교 대학원 컴퓨터공학과 졸업(공학석사)
2006년 부산대학교 대학원 컴퓨터공학과 졸업(공학박사)
2008년 고신대학교 인터넷비즈니스학과 교수
※ 관심분야 : 인터넷응용, RFID, 모바일