UFLRU: 비휘발성 메모리와 플래시 메모리의 계층 구조를 위한 영속 객체 스와핑 알고리즘

UFLRU: A Persistent Object Swapping Algorithm for The Hierarchical Structure of Non-Volatile Memory and Flash Memory

요 약

프로세스가 생성한 영속객체를 비휘발성 메모리에 저장함으로써, 비휘발성 메모리의 낮은 입출력 지연을 이용하여 영속 객체를 생성 및 관리할 수 있다. 그러나 비휘발성 메모리는 낸드 플래시 메모리 또는 하드디스크에 비하여 적은 용량을 갖는 제약이 있다. 본 논문에서는 낸드 플래시 메모리를 비휘발성 메모리의 스왑 영역으로 사용함으로써 프로세스가 생성할 수 있는 영속 객체의 총 용량을 확장한다. 스왑 영역으로 옮길 희생 영속객체를 선택하는 UFLRU (Unmapped-object First LRU)알고리즘을 개발하였으며, CFLRU 알고리즘 대비 약 15%의 성능 향상을 확인하였다.

1. 서 론

비휘발성 메모리 (Non-Volatile Memory)는 바이트 단위의 입출력 동작, DRAM과 유사한 수준의 입출력 지연, 그리고 데이터의 영속적인 저장과 같은 장점을 가지는 차세대 저장장치이다. 비휘발성 메모리로는 Spin-torque transfer RAM (STT-RAM), Phase change memory (PCM), Resistive RAM (ReRAM) 등이 있다.

영속 객체 힙 저장소 (Heap-Based Persistency Object Store) [1]는 프로세스의 가상 주소공간 중 일정 영역을 영속객체를 위한 주소영역으로 사용하며, 이 주소공간을 영속객체 힙이라 한다. 커널은 프로세스에게 영속객체 생성 및 관리를 위한 인터페이스를 제공하고, 프로세스는 이러한 인터페이스를 이용하여 영속객체 힙에 영속객체를 생성한다. 영속객체 힙에 생성된 영속객체는 시스템의 비휘발성 메모리에 저장되어, 시스템 종료 시에도 데이터의 영속성을 보장받는다.

영속객체 힙 저장소의 각 논리주소는 비휘발성 메모리의 물리주소와 1:1로 매핑되어 있다. 따라서 비휘발성 메모리의 물리주소가 모두 할당되면 추가적인 영속 객체를 생성할 수 없게 된다. 즉, 영속 객체 힙 저장소의 총 영속 객체의 크기는 비휘발성 메모리의 크기에 한정되는 문제점이 발생한다. 본 논문에서는 이를 해결하기 위해, 비휘발성 메모리와 플래시 메모리의 계층 구조를 구성하고, 비휘발성 메모리에 용량이 부족할 경우 영속 객체를 낸드 플래시로 방출하는 스왑 인/아웃 기법을 적용한다. 해당 기법을 통해 프로세스가 생성할 수 있는 영속 객체의 총 용량을 확장한다. 또한 스왑 영역으로 옮길 희생 영속 객체를 선택하는 UFLRU (Unmapped-object First LRU)알고리즘을 개발하여, CFLRU 알고리즘 대비 약 15%의 성능 향상을 확인하였다.

2. 배 경 : 리눅스의 스와핑 알고리즘

리눅스는 Second Chance 알고리즘을 사용하여 DRAM에서 디스크의 스왑 영역으로 스왑 아웃 시킬 페이지를 선택한다. DRAM의 사용중인 페이지는 2개의 링크드 리스트 자료구조로 관리되며, 각각 최근에 사용된 페이지를 저장하는 액티브 리스트와, 스왑 아웃의 대상 페이지들이 관리되는 인액티브 리스트이다. 최근에 사용하였거나 자주 접근된 페이지일수록 액티브 리스트에, 사용되지 않았거나 오래된 페이지들은 인액티브 리스트에 위치시킨다. DRAM에 할당 공간이 부족할 경우, 인액티브 리스트의 Tail부터 검사하여 스왑 대상 페이지를 선정하고 스왑 영역으로 방출한다.

3. 비휘발성 메모리 계층 구조를 위한 스와핑 알고리즘

비휘발성 메모리에 더 이상 새로운 객체를 저장할 영역이 없을 경우, 비휘발성 메모리의 영속 객체를 플래시 메모리로 방출하여 빈 공간을 생성하도록 한다. 본 논문에서 제안하는 UFLRU (Unmapped-object First LRU)알고리즘은 인액티브 리스트의 페이지 정렬 기준을 페이지의 매핑 여부에 따라 결정하는 방식이다. 이 때, 매핑 객체 (Mapped Object)는 프로세스의 영속객체 힙

영역에 주소공간을 할당 받은 객체를 말하며, 비매핑 객체 (Unmapped Object)는 물리주소공간에는 데이터가 존재하나, 이를 접근하는 프로세스가 없는 객체를 말한다. 비매핑 객체는 현재 프로세스가 사용 중이지 않은 객체이므로, 매핑되기 전까지는 프로세스에 의한 접근이 발생하지 않는다. 따라서 스왑 아웃 되어도 프로세스에 의한 재 접근 가능성이 낮아, 스왑 아웃 대상으로 매우 적합하다. 따라서 우리는 인액티브 리스트를 해당 페이지의 매핑여부에 따라 정렬하고, 매핑되어있지 않은 페이지를 우선적으로 회수하는 UFLRU (Unmapped-object First LRU) 알고리즘을 개발하였다.

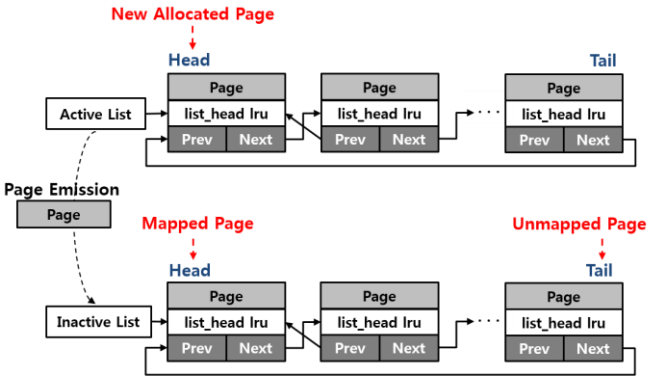

그림 1. UFLRU 페이지 관리 방법

그림 1은 UFLRU를 사용하여 액티브 리스트와 인액티브 리스트를 정렬하는 모습을 보여준다. 새로 할당된 페이지는 액티브 리스트에 먼저 삽입되며, 접근이 빈번하게 발생하지 않은 페이지들은 인액티브 리스트로 방출된다. 방출된 페이지가 인액티브 리스트에 삽입될 때에는, 해당 페이지가 비매핑 상태일 때는 리스트의 Tail로, 그외의 경우에는 Head로 삽입되어, 추후 스왑 아웃 동작 시, 비매핑 상태의 페이지가 먼저 회수될 수 있도록 한다.

4. 성능 평가

UFLRU의 성능을 비교 평가하기 위해 리눅스 커널 3.15에 default로 적용되어 있는 Second Chance (SC)알고리즘, First-In First-Out (FIFO), Clean-First LRU (CFLRU) [2]를 구현하여, 동일한 워크로드에서 각 알고리즘의 성능을 측정하였다. 1Gbyte DRAM과 512Mbyte 크기의 가상 비휘발성 메모리를 DRAM으로 구성하였으며, 2Mbyte 크기 영속객체 400개를 생성하여 스왑 동작이 발생하도록 하였다. 영속 객체 생성 후, 임의의 객체에 접근하여 객체의 모든 데이터를 업데이트하는 동작을 총 500회 수행하였다. 이 때 각 객체들을 빈번하게 접근되는 Hot 객체와 간헐적으로 접근되는 Cold 객체로 구분하여, Hot 객체가 전체 객체 접근의 80%를 차지하도록 하였다. 즉, 총 500회의 객체 접근 중 400회가 Hot 객체에 발생하고, 나머지 100회 접근이 Cold 객체에 발생한다

그림 2는 Hot 객체의 비율을 10%에서 40%로 증가시켰을 때, 워크로드의 총 실행시간을 보여주며, 그림 3은 각 실험에서 스왑 영역으로 스왑 아웃 된 페이지의 수를 보여준다. Hot 객체의 비율이 많아질수록, 스왑 아웃 동작이 자주 발생함을 확인할 수 있다. 그에 따라 테스트 수행시간 또한 Hot 객체 비율이 커질수록 약 3초에서 9초로 3배 증가하였다. 특히 Hot 객체 비율이 50%인 워크로드에서는 UFLRU의 실행 시간이 CFLRU 대비 15% 감소함을 확인하였다.

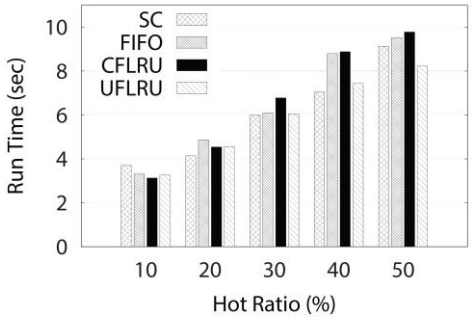

그림 2. 희생 페이지 선정 알고리즘에 따른 실행시간

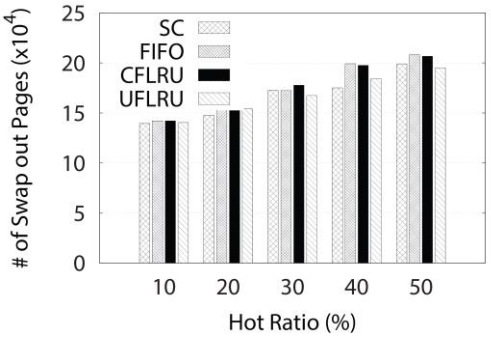

그림 3. 희생 페이지 선정 알고리즘에 따른 스왑아웃 페이지 수 비교

5. 결론

본 논문에서는 비휘발성 메모리에 여유 공간이 부족할 때 객체 매핑 여부를 고려하여 스왑 대상 페이지를 선정 및 플래시 메모리로 방출하는 UFLRU 알고리즘을 개발하였다. 제안한 알고리즘은 성능 평가 결과 CFLRU 알고리즘 대비 15%의 성능 개선을 보였다.

참 고 문 헌

[1] Taeho Hwang, Jaemin Jung, and Youjip Won, “HEAPO: Heap-based Persistent Object Store”, ACM Transactions on Storage, Vol. 11, Issue 1, Dec. 2014

[2] Seon-yeong Park, Dawoon Jung, Jeong-uk Kang, Jin-soo Kimg, and Joonwon Lee, “CFLRU: A Replacement Algorithm for Flash Memory”, In Proc. of CASES, ACM, 2016