최소집합 로깅 : 비휘발성 메모리상의 자료구조 복구기법

이건우○ 정재민 원유집

한양대학교 컴퓨터 소프트웨어학과
kunulee@hanyang.ac.kr, tsecret@hanyang.ac.kr, yjwon@hanyang.ac.kr

Minimal set logging : Data structure recovery mechanism in nonvolatile memory

Keonwoo Lee○ Jaemin Jung Youjip Won

Department of Computer and Software, Hanyang University

요 약

비 휘발성 메모리는 프로그램의 자원들을 프로그램의 수명에 상관없이 메모리 내에 유지하고 있는 영속적인 특징을 가진 차세대 소자이다. 비 휘발성 메모리를 관리하는 대표적 메커니즘인 "HEAPO"에 구현되어 있는 키-값 저장 자료구조는 갑작스러운 전원의 손실이나, 오류로 인해 잦은 데이터 손실을 야기한다. 본 논문에서는 비 휘발성 메모리상에서 발생하는 비정상적인 오류로 초래되는 데이터의 손실을 일관성 있는 상태로 복구하고자 Undo 로깅 기반 손실 복구 기법을 제시하고, 일반 Undo 로깅 연산의 잦은 캐시 플러시를 해결하고자 최소 집합 연산을 로깅하는 최소집합(Minimal set) 갱신 기법을 제시한다. 또한 구현되어 있는 각 자료구조별 동일한 워크로드에 따른 최소집합 갱신의 성능을 측정했다. 아무런 복구기법도 사용하지 않은 자료구조의 워크로드 연산처리 속도에 비해 2.3배 지연되는 것을 확인했다.

1. 서 론

현재 대부분의 시스템은 휘발성 메모리에 근간한 컴퓨터 시스템으로 이루어져 있다. 최근 비휘발성 메모리에 관심이 높아지면서 비 휘발성 메모리에 저장하는 데이터에 대한 일관성 보장 관련 연구가 활발히 진행 되고 있다. 비 휘발성 메모리는 프로그램이 사용하는 자원들을 프로그램이 동작하는 시간에 상관없이 메모리 내에 유지하고 있는 영속적인 특징을 가진 차세대 소자이다. 즉, 비휘발성 메모리는 별도의 전기적 접근이 없이 데이터를 유지할 수 있고, 휘발성 메모리처럼 바이트 단위 접근을 함으로서 현재 DRAM이 직면한 기존의 문제점들을 해결할 수 있는 기회로 급부상 하고 있다[1]. 비 휘발성 메모리를 관리하는 방법은 다양하며 대표적인 비 휘발성 메모리 관리 메커니즘은 HEAPO(Heap-Based Persistent Object Store)[1], NV-HEAPs[2], Mnemosyne[3] 등이 있다. 본 논문에서는 대표적인 영속 힙 저장소 관리 메커니즘인 HEAPO를 사용하여, 비 휘발성 메모리에 키-값을 저장하기 위해 구현된 자료구조인 Linked List, Hash Table, B tree의 삽입, 삭제 연산을 하면서 발생하는 데이터의 비정상적인 오류에 대한 손실 복구 기법으로 각 자료구조별 최소집합(Minimal set) 갱신 기법을 제시하고, 기존의 복구 기법과의 연산처리속도 차이를 비교한다.

2. 설 계

일반적인 시스템에서의 데이터손실 복구 방법으로는 데이터베이스가 사용하는 Undo 로깅 또는 Redo 로깅 등이 있으며, SQLite[4]와 같은 상용 데이터베이스들이 실제로 사용하고 있다. 복구 외에도 대부분의 파일시스템이 사용하는 COW(Copy-On-Write)와 같은 방법으로 데이터의 일관성을 유지하기도 한다.

일관성 유지를 위한 여러 가지 방법들 중, 데이터베이스의 대표적인 복구 기법인 Undo 로깅은 시스템의 갑작스러운 전원의 손실이나, 오류가 발생했을 때, 장애 발생 이전의 상태로 복구시켜주는 기법으로 '로그'라는 갱신 저장 공간을 사용하여 이전의 상태를 기록한다. 논리적으로 한 순간에 발생하는 단위를 트랜잭션이라고 부르며, Undo 로깅은 트랜잭션 동안 발생하는 데이터의 쓰기 연산을 로그에 기록한다. 만약 연산 중 비정상적인 오류가 발생하는 경우에는 장애 이전의 상태로 돌려준다. [그림 1]은 영속 힙 저장소 관리 메커니즘인 "HEAPO"에서 데이터의 일관성을 보장하기 위해 구현된 로그 설계도이다. (4+i*2)번째 인덱스에는 갱신하려는 곳이 저장된 주소 값, (5+i*2)번째 인덱스에는 갱신하려는 곳의 원래 값을 갱신하여 비정상 적인 오류가 발생했을 때 로그를 참조하여 이전의 상태로 되돌릴 수 있는 메커니즘을 제공한다.

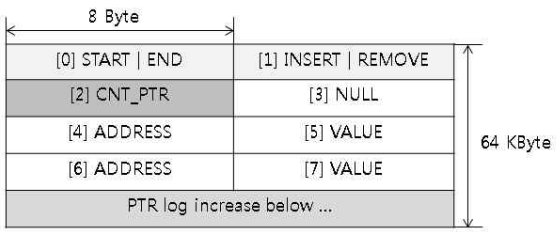

그림 1. 데이터의 일관성을 위한 로그 구조

휘발성 메모리의 응용프로그램들은 메모리의 쓰기 연산에 대한 갱신을 하지 않아도 심각한 문제를 초래하지 않는다. 프로세스가 수명이 다되어, 프로세스가 종료되는 경우 모든 메모리의 반환이 일어나기 때문이다. 즉 메모리의 누수나 데이터가 깨지는 현상을 발생시키지 않는다. 하지만 비 휘발성 메모리의 응용프로그램은 프로세스의 수명이 종료되어도 데이터가 비 휘발성 메모리 내에 남아 있기 때문에 일관성 있는 데이터를 비휘발성 메모리에 관리하기 위해서는 메모리 쓰기 연산에 대한 이력을 갱신하는 작업이 필요하다. 또한 비 휘발성 메모리 내의 데이터 일관성을 유지하기 위해서 대부분의 비 휘발성 관리 메커니즘은 주기적으로 메모리 연산 후에 캐시 플러시를 호출하여 캐시에 있는 데이터들을 메인 메모리로 내려준다. 시스템적 측면에서 잦은 캐시 플러시는 응용프로그램의 성능에 크게 악영향을 끼칠 수 있다. [그림 2]는 영속 힙 저장소 관리 메커니즘인 “HEAPO“에 구현되어 있는 Linked List 와 Hash Table 그리고 B tree의 삽입 연산에 대한 캐시 플러시 유무에 따른 처리시간을 측정한 그래프이다.

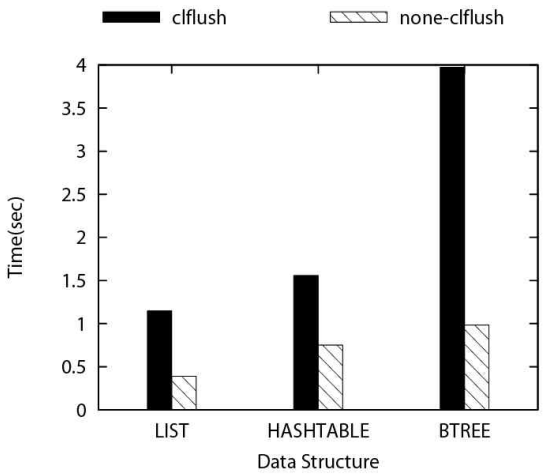

그림 2. 실행시간 : 1,000,000 번 삽입연산

1,000,000번의 삽입연산에 대한 Linked List, Hash Table, B Tree 각각에 대한 캐시 플러시 연산 진행 성능은 캐시 플러시를 진행하지 않은 연산보다 약 2.9배, 2.0배, 4.0배 느린 것을 확인 할 수 있다. 기존의 Undo 로깅을 진행할 경우, 모든 연산에 대한 이력을 ‘로그’에 갱신하는 연산까지 포함해서 캐시 플러시를 진행하기 때문에 성능의 차이는 더욱 심해진다.

비 휘발성 메모리의 캐시 플러시 오버헤드를 줄이기 위해서, 본 논문에서는 영속 힙 저장소 메커니즘에 저장되어있는 각 자료구조의 삽입, 삭제 연산에 대하여 최소집합만을 로깅하는 최소집합(Minimal Set) 로깅 기법을 제시한다. 최소집합(Minimal Set) 로깅 기법이란 트랜잭션의 연산 중 모든 메모리 쓰기에 대한 이력을 로그에 갱신하는 일반 Undo 로깅과는 달리 공유하는 자원에 대한 일관성에 위험이 되는 핵심적 연산에 대한 집합만을 로깅하는 메커니즘을 말한다.

[그림 3]는 각 자료구조별 삽입 연산에 대한 과정이다. 그 중 Linked List의 삽입연산은 세 과정으로 이루어지며, 새로운 노드의 할당, 할당 된 노드의 다음 노드 포인팅, 리스트의 head 포인터 이동으로 구성된다. 비 휘발성 메모리에서 이러한 세 과정에 대한 모든 이력 갱신 작업은 잦은 캐시 플러시 연산을 의미하기 때문에 리스트 자체의 일관성을 위반할 위험이 있는 연산인 리스트의 head 포인팅만을 로그에 갱신한다. Hash Table의 삽입연산도 Linked List와 마찬가지로 세 가지 과정을 수행한다. 새로 노드를 할당받는 작업, 할당 받은 노드의 Hash 값을 구해, 할당 받은 노드의 다음 노드에 관련한 포인터를 Hash Table 내의 Hash에 해당하는 인덱스에 연결시켜 주는 작업, 마지막으로는 Hash 값에 해당하는 Hash Table 인덱스의 head 자체를 새로 할당받은 노드로 이동해주는 작업으로 구성되어 있다. 이러한 연산 중 Hash Table 자체의 일관성에 문제가 될 수 있는 연산은 Hash Table의 Head 포인터를 할당 받은 노드로 변경 시키는 작업으로 다른 작업은 연산 중간에 비정상적인 오류가 발생하더라도 Hash Table 자체의 일관성을 깨뜨리지 않는다. 그러므로 마지막 연산에 대한 이력을 갱신함으로써 캐시플러시의 수행 빈도를 감소시킬 수 있다. 마지막으로 B Tree의 삽입 연산의 경우, B Tree 자체의 일관성을 훼손할 수 있는 연산들이 많다. 예를 들면 키-값을 저장하는 B Tree의 삽입연산의 경우, 이미 연결되어 있는 B Tree 노드에 연산 과정을 진행하기 때문에 최소집합(Minimal Set)을 추출하기 힘들다. 따라서 노드 내부의 키-값을 저장하는 연산에 대한 최소집합(Minimal Set) 추출이 아닌, 노드의 할당 과정에 대해서 최소집합(Minimal Set) 추출을 고려한다. [그림 3]의 B Tree의 삽입 연산 중에서 B Tree자체의 높이를 증가시키는 연산을 진행 할 경우 새로운 노드를 할당받고 기존의 루트노드에 해당하는 주소 값과, 키의 일부를 새로이 할당받은 노드에 할당하는 과정은 B Tree자체의 일관성을 훼손하지 않는 작업이므로 최소집합(Minimal Set)으로 설정하지 않는다. 그리고 B Tree Head가 새로 할당받은 노드를 가리키는 작업이나 B Tree자체의 높이를 증가 시키는 작업은 B Tree자체의 일관성을 훼손시킬 수 있는 위험이 있으므로 두 과정을 최소 집합(Minimal Set)으로 설정한다.

3. 평 가

최소 집합(Minimal Set) 로깅 기법이 적용된 비 휘발성 메모리의 자료구조들의 연산과 아무런 로깅 기법도 적용하지 않은 기법을 비교해 성능을 분석했다. [그림 4]은 영속 힙 저장소 메커니즘 상에서 동작하는 세 가지 자료구조 Linked List, Hash Table, B Tree에 대한 1,000,000번 삽입 연산에 대한 ops/sec을 측정한 그래프이다.

[None]은 아무런 로깅 기법도 적용하지 않음으로서 성능이 좋은 장점이 있지만, 비정상적인 종료가 발생했을 시에는 데이터의 깨짐 현상이 발생하고, 복구가 불가능하다. 반면 최소집합(Minimal Set)을 추출하여 Undo 로깅 방법을 사용한 방법인 [UNDO]는 데이터가 깨져도 복구가 가능하며 일관성 있는 자료구조를 유지할 수 있다. 하지만 약간의 성능하락이 존재한다. 로그에 이력을 갱신하는 연산에 대한 캐시플러시 때문이며, Linked List

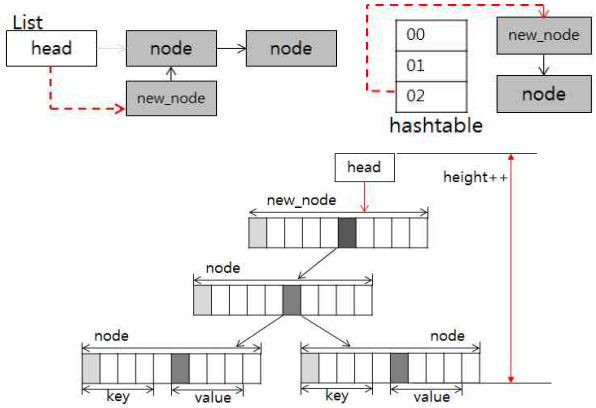

그림 3. 각 자료구조별 삽입 연산 과정

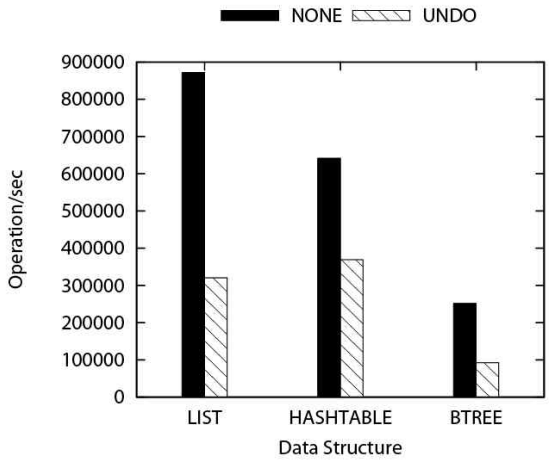

그림 4. 자료구조별 최소집합 로깅 유무 성능 비교

가 약 2.7배 , Hash가 약 1.7배, B tree가 약 2.7배 정도 성능 하락이 있다.

4. 결 과

본 논문에서는 비휘발성 메모리를 관리하는 대표적 메커니즘인 “HEAPO” 내부에 구현되어 있는 키-값 자료구조를 사용하여, 데이터의 손실에 따른 복구메커니즘에 대하여 제시했다. 또한 이러한 로깅 메커니즘이 로깅하지 않는 기존의 메커니즘에 비해 얼마만큼의 성능 하락이 있는지, 또한 어떠한 이점이 있는지 살펴보았다.

하지만 최소집합(Minimal Set)을 통하여 Undo Logging을 진행하는 것은 가비지가 생성될 확률이 있다. 예를 들어 노드의 할당 과정 후 비정상 적인 오류가 발생하면 최소집합(Minimal Set)에 포함되지 않는 연산에 대한 복구는 진행하지 않기 때문에 이러한 요소들은 비휘발성 메모리에 남아 가비지가 될 수 있다. 이를 개선하기 위한 자료구조별 가비지 컬렉션을 개발함으로써 문제를 해결함과 동시에 성능 향상을 보장할 수 있을 것이다.

5. 사 사

본 연구는 지식경제부 및 한국산업기술평가관리원의 산업원천 기술개발사업(정보통신)의 일환으로 수행하였음, [No.10041608, 차세대 메모리 기반의 스마트 디바이스용 임베디드 시스템 소프트웨어]

참고 문헌

[1] Taeho Hwang, Jaemin Jung and Youjip Won, “HEAPO: Heap-Based Persistent Object Store“ in ACM Transactions on Storage(TOS), Volume 11 Issue 1, February 2015.

[2] J. Coburn, A. M. Caulfield, A. Akel, L. M. Grupp, R. K. Gupta, R. Jhala, and S. Swanson. NV-Heaps: making persistent objects fast and safe with next-generation, non-volatile memories. In International Conference on Architectural Support for Programming Languages and Operating Systems, 2011.

[3] Haris Volos , Andres Jaan Tack , Michael M. Swift, Mnemosyne: lightweight persistent memory, Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems, March 05-11, 2011

[4] The SQLite
http://www.sqlite.org/