스마트 TV 용 고속 IO 기법

정창훈
한양대학교
전자컴퓨터통신공학과
서울 성동구 행당동
acezero@hanyang.ac.kr

황태호
한양대학교
전자컴퓨터통신공학과
서울 성동구 행당동
htaeh@hanyang.ac.kr

김규일
한양대학교
전자컴퓨터통신공학과
서울 성동구 행당동
gaya@hanyang.ac.kr

원유집
한양대학교
전자컴퓨터통신공학과
서울 성동구 행당동
yjwon@hanyang.ac.kr

요약: 스마트 TV 는 XML 형식의 파일을 사용하기 위해 데이터 변환과정을 거쳐 트리 형태의 자료구조를 구성한다. 트리 형태의 데이터는 프로세스 주소공간상의 heap 영역에 존재하므로 프로세스 종료와 함께 소멸된다. 따라서 변환과정을 거쳐 트리 형태의 데이터를 구성 하였더라도, 프로세스 재 실행 시 XML 파일을 사용하기 위해서는 다시 변환과정을 거쳐 트리 형태의 데이터를 구성해야 한다. 본 논문에서는 스마트 TV 환경에서 트리 형태의 데이터에 영속성을 부여함으로써 데이터 변환과정 없이 재사용 가능토록 하는 기법을 제안한다. 고속 IO 기법은 Object 를 제공함으로써 프로세스 주소공간의 일부에 영속성을 부여한다. 트리형태의 데이터를 Object 에 저장함으로써 해당 자료구조가 영속성을 가지며, 프로세스 재실행 후에도 데이터 변환과정 없이 재사용할 수 있다. 본 기법을 웹브라우저에 적용하여 HTML 을 파싱하여 트리를 구성하는 과정을 생략하였으며 실행시간을 최대 61% 단축하였다.

핵심어: 스마트 TV, 오브젝트, 웹브라우저

1. 서 론

스마트 TV 는 CPU, DRAM, 낸드플래시 기반의 스토리지로 구성된다. 스마트 TV 에서 웹브라우저와 같이 XML 형식의 데이터를 사용하는 애플리케이션은 XML 을 사용하기 위해 적절한 형태로 변환해야 한다. 웹브라우저의 경우 XML 을 파싱하여 트리를 구축한다. 트리의 노드들은 웹 페이지의 각 요소들의 그려질 위치, 크기, 색 등의 정보를 담고 있으며, 이 트리를 통해 웹 페이지를 화면에 그려준다. heap 영역의 데이터는 프로세스 종료와 함께 소실되므로 트리 형태의 데이터를 재사용하기 위해서는 이를 낸드플래시에 파일 형식으로 저장해야 한다. 하지만 낸드플래시는 바이트 단위의 입출력을 지원하지 않으므로 in memory data 를 파일로 저장하기 위해 serialization 과정이 요구되며, 저장된 파일을 재사용하기 위해 다시 deserialization 과정을 거쳐 트리 형태로 변환해야 한다. 즉, 현재의 시스템에서는 특정 XML 파일을 트리 형태로 변환했더라도, 프로세스가 종료된 이후에 동일한 XML 파일을 사용하기 위해서는 다시 트리 형태로 변환과정을 요구한다. 트리 형태로 변환하는 작업은 256kb 크기의 자료구조를 기준으로 143msec 의 시간을 필요로 하며[1] 애플리케이션의 구동 속도를 저하시키는 주요 요인이다.

본 논문에서는 스마트 TV 에서 동적으로 할당 받는 메모리 데이터에 선택적으로 영속성을 부여하는 기법을 제시한다. 본 논문에서 영속성이란 해당 프로세스가 종료된 후 재실행되더라도 프로세스 주소공간상의 데이터가 보존되어 재사용 가능함을 뜻한다. 이 기법을 적용하여 스마트 TV 에서 웹브라우저와 같이 XML 형식의 데이터를 사용하는 애플리케이션에서 트리 형태의 데이터에 영속성을 부여하고 재사용 가능토록 한다. 프로세스 종료 후 재실행 하더라도 저장된 트리 형태의 데이터를 재사용할 수 있으므로 XML 파일을 트리 형태로 변환하는 과정을 생략할 수 있고, 이를 통해 애플리케이션의 성능을 향상시킨다. 메모리 데이터에 영속성을 지원하기 위해, 영속적 heap 의 개념으로 Object 를 제시한다. Object 는 프로세스 주소공간에 존재하며 mmap() 시스템콜을 통해 파일과 사상되어 있다. 이를 Object File 이라 한다. Object 의 데이터는 페이지캐시를 거쳐 Object File 에 저장되므로 비휘발성을 지닌다. Object 는 바이트 단위의 메모리 할당/해제 인터페이스를 제공하며, 스마트 TV 의 애플리케이션은 이를 통해 메모리를 동적으로 할당 받아 트리 형태의 데이터를 구성한다. 프로세스 종료 후 object file 을 mmap() 시스템콜을 통해 프로세스 주소공간에 사상함으로써 Object 를 재사용 할 수 있다. 또한 트리의 루트노드를 기억함으로써 Object 내에 존재하는 트리형태의 데이터를 재사용할 수 있으며, XML 데이터의 deserializaion 과정을 생략 할 수 있다.

남은 장의 구성은 다음과 같다. 2 장에서는 관련연구에 대해 설명하고, 3 장에서는 고속 IO 기법을 제시한다. 4 장에서는 고속 IO 기법의 성능에 대해 설명하며 5 장에서는 결론에 대해 설명한다.

2. 관련연구

Grasshopper[2]는 single address space 운영체제로서 container 를 통해 데이터의 영속성을 제공해 준다. 하지만 포인터 유효성 문제를 해결하기 위해 포인터 스위즐링[3] 과정을 거쳐야 한다. 따라서 application 에 적용하기 위해서는 복잡한 코드 수정이 필요하다는 단점이 있다.

SoftPM[4]은 container 를 통해 메모리 데이터의 영속성을 보장한다. 루트노드를 지정하면, 이와 연결된 노드들을 따라가며 container 에 속하지 않은 모든 노드를 container 로 옮겨 영속성을 부여한다. 따라서 SoftPM 은 자동으로 루트노드와 연결된 메모리 데이터에 영속성을 부여한다. SoftPM 에 애플리케이션의 소스코드 수정이 적다. SoftPM 은 스토리지에 저장된 컨테이너를 복원할 때 페이지 맵핑 주소가 고정되어 있지 않기 때문에 포인터 스위즐링 과정을 필요로 한다. 반면 Smart TV 고속 IO 기법은 고정된 주소공간에 Object 가 맵핑되기 때문에 포인터 스위즐링 오버헤드를 가지지 않는다는 장점을 가진다.

본 논문에서는 메모리/디스크기반의 시스템에서 in memory data 에 영속성을 부여하는 기법을 제시한다. 또한 포인터 스위즐링 과정을 제거함으로써 이 기법을 애플리케이션에 적용함에 있어 간단한 코드수정만을 필요로 하도록 한다.

3. 스마트 TV 고속 IO 기법

본 논문의 기법을 적용하여 스마트 TV 의 트리 형태의 데이터를 변환과정 없이 관리/재사용 할 수 있도록 한다.

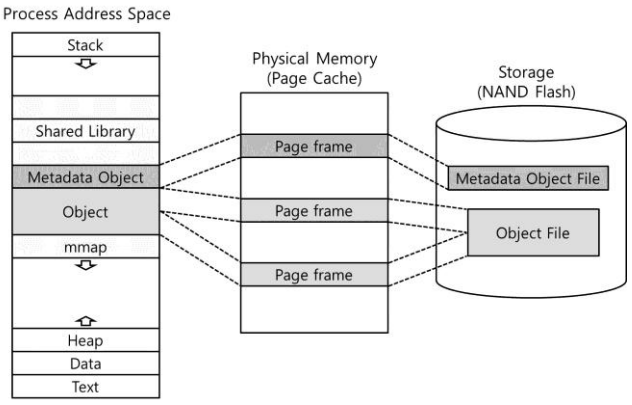

그림 1. 기본개념

본 논문은 In memory data 에 영속성을 부여하기 위한 저장공간 Object 를 제안한다. Object 는 프로세스 주소공간에 존재하며 mmap() 시스템콜 호출을 통해 파일시스템상의 Object File 과 사상되어있다.

낸드플래시가 바이트 단위의 메모리 입출력을 지원하지 못하기 때문에 낸드플래시가 직접적으로 프로세스 주소공간에 사상 될 수 없는 한계점을 갖는다. 따라서 Object file 은 파일시스템의 파일로 존재하며 낸드플래시에 저장되지만, 프로세스 주소공간상의 object 는 실제적으로 Object file 데이터가 복사된 페이지캐시를 사상한다. Object 에 대한 메모리 입출력은 운영체제의 페이지캐시를 거쳐 파일 시스템의 정책에 따라 낸드플래시에 플러시 된다.

3.1 Object 확장

노드 할당 중 Object 내의 할당 가능한 메모리가 부족한 경우 Object 의 크기를 증가시켜야 한다. 추가적으로 할당 받은 주소공간을 region 이라 한다. Object 의 인접한 주소공간이 이미 사용중인 경우 불연속적인 주소에 region 이 할당된다.

3.2 고정주소 사상기법

Object file 이 프로세스의 주소공간에 사상될 때, 그 위치가 고정되어 있지 않다면 Object 내부 노드 간의 연결을 나타내는 포인터 값이 유효하지 않게 된다. Object 맵핑 정보를 관리하기 위한 메타데이터는 다음과 같이 구성되어 있다.

3.2.1 메타데이터 관리

모든 object 들을 관리하기 위한 메타데이터 p_superblock, 각 object 의 메타데이터인 p_ns_entry, 각 region 에 대한 메타데이터인 p_vm_area 가 존재한다.

p_superblock 은 object 에 대한 메타데이터인 p_ns_entry 를 해시테이블로 관리하도록 구현하여 object 의 빠른 삽입, 삭제, 탐색을 가능하도록 하였다. 또한 p_ns_entry 를 list 로 관리하여 해시테이블의 모든 엔트리를 탐색하지 않고, 모든 p_ns_entry 를 순회할 수 있도록 하여, 순차탐색에 유리하도록 구현하였다.

p_ns_entry 는 해당 object 에 대한 이름필드를 가지고 있다. 또한 object 를 구성하는 region 들의 메타데이터인 p_vm_area 를 list 로 관리하며, 처음과 마지막 region 에 대한 포인터를 유지한다. 이는 Object 의 증가/감소가 파일의 끝 부분에서만 이루어지는 것을 고려한 것이다. 즉, p_vm_area 의 삽입/삭제는 끝 부분에서만 이루어지므로 트리와 같은 복잡한 자료구조의 필요 없이 리스트로 충분하다.

p_vm_area 는 해당 region 이 사상될 프로세스 주소공간상의 위치와 크기, object file 의 offset 에 대한 정보를 저장한다. 또한 해당 object 의 다른 region 에 접근하기 위해 prev, next 필드를 유지한다.

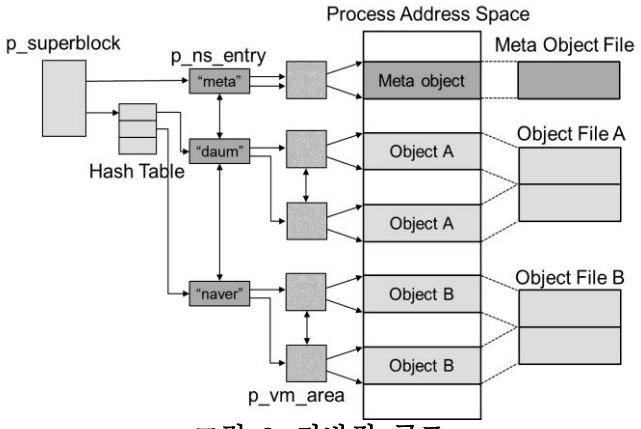

그림 2. 전반적 구조

3.2.2 메타데이터 위치

네임스페이스 관련 메타데이터를 저장하기 위한 Object 가 존재한다. 이 Object 를 metadata Object 라 하고, 이와 사상된 파일을 metadata Object file 이라고 한다. 이들 메타데이터는 metadata Object 에서의 node 할당/해제를 통해 관리된다.

4. 실험

이 장은 고속 IO 기법을 비교적 간단한 웹브라우저에 적용하여 성능 측정을 하고, 기법을 적용하지 않은 경우와 비교하였다. 실험은 AMD Phenom X4 925 Processor, 4GB DDR3 DRAM 환경에서 진행되었다. 스토리지로는 500GB 하드디스크 7200RPM 와 60GB SSD (OCZ VERTEX2 SATA 2)을 사용하여 각각의 경우에 대해 성능측정을 진행하였다. 웹브라우저는 Dillo 2.2 를 사용하였다.

고속 IO 기법을 적용하여 웹브라우저의 캐시 메커니즘에 변화를 주었다. 기존의 웹브라우저는 웹 페이지에 접근할 경우 HTML 을 웹에서 받아와 디스크에 캐시하고, HTML 을 파싱하여 트리를 구성한다. 트리를 통해 해당 웹 페이지를 그려주며, 브러우저 종료시 트리형태의 자료구조는 사라지게 된다. 캐시된 웹 페이지에 재 접근시 디스크에서 HTML 을 읽어와 파싱과정을 거쳐 트리를 생성하고, 이를 화면에 그려주는 작업을 한다. 스마트 TV 고속 IO 기법을 적용한 웹브라우저의 경우 파싱과정을 거쳐 생성한 트리형태의 자료구조를 캐시한다. 웹브라우저 종료 후 캐시된 웹 페이지에 접근할 경우 트리형태의 자료구조를 재사용 하여 웹페이지를 그려줌으로써 파싱과정과 트리 구축 과정을 생략할 수 있다.

성능 평가는 웹브라우저를 실행하여 해당 웹 페이지를 화면에 그려주기까지의 시간을 측정함으로써 했다. 구체적으로 Disk IO, Parsing, Drawing 각 단계에 걸리는 시간을 측정하였다. 각 웹페이지에 대해 기존 웹브라우저와, Fast IO 기법을 적용한 웹브라우저 두 가지에 대해 실험을 하였고 실험한 스토리지에 따라 램디스크, SSD, 하드 디스크로 구분하였다.

스토리지로 하드 디스크를 사용했을 때의 경우 모든 웹 페이지에 대하여 Parsing Time 을 제거하는 효과를 보였으나, Disk IO 시간이 크게 증가하였다. 이는 트리 형태의 자료구조가 HTML 의 파일 size 에 비해 10 배이상 증가함으로써 생기는 오버헤드이다. 하드디스크는 느린 입출력 성능을 보임으로 Disk IO 로 인한 오버헤드가 Parsing 과정을 생략하는 효과보다 크며, 하드디스크를 스토리지로 사용하는 환경에서는 고속 IO 기법의 적용이 적절하지 않다는 것을 확인할 수 있다.

SSD 를 스토리지로 사용한 경우 HDD 환경에서의 실험과 마찬가지로 Parsing Time 이 생략되었으며, Disk IO Time 이 증가했다. 하지만 SDD 의 특성상 입출력 속도가 빠르므로 증가한 Disk IO 오버헤드보다 Parsing Time 생략으로 인해 성능이 향상이 더 크다. 램디스크의 경우 Parsing Time 이 생략되었으며 SSD 에 비해 Disk IO 오버헤드가 더 줄어든 것을 확인하였다.

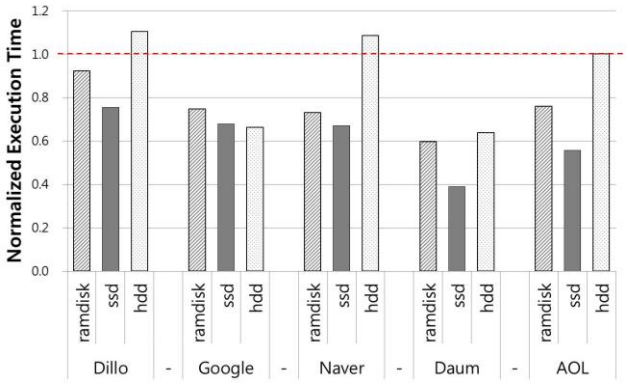

그림 3. Normalized Execution Time

그림 3 은 Fast IO 기법을 적용한 웹브라우저의 실행시간을 기존 웹브라우저를 실행하는데 걸린 시간으로 normalize 한 그래프이다. HDD 의 경우 Disk IO 오버헤드로 인해 성능이 저하된 웹 페이지도 있지만, SSD 의 경우 모든 웹 페이지에서 성능향상을 보였으며, 최대 61 퍼센트의 성능향상을 보였다.

5. 결 론

본 논문은 스마트 TV 환경에서의 고속 IO 기법을 제안하였다. 고속 IO 기법을 통해 동적으로 할당 받는 메모리 데이터에 대해 선택적으로 영속성을 부여할 수 있다. 이를 적용하여 XML 형식의 데이터를 사용하는 애플리케이션이 트리형태의 데이터를 캐시하도록 하여, 프로세스 종료 후 재실행하더라도 deserialization 과정 없이 트리형태의 데이터를 사용

할 수 있도록 하였다. 하지만 실험 결과에서 알 수 있듯이 변환된 자료구조를 저장하는 과정에서 Disk IO 오버헤드가 증가한다. 느린 입출력 성능을 지닌 하드디스크의 경우 deserializaion 의 생략을 통해 얻는 성능향상보다 Disk IO 오버헤드 증가로 인한 성능저하가 더 크므로, 고속 IO 기법을 적용하는 것이 적절하지 않다. 반면 SSD 의 경우 빠른 입출력 특성을 지니므로 Disk IO 오버헤드가 비교적 적으므로 고속 IO 기법을 적용하였을 경우 애플리케이션의 성능향상을 가져온다. 스마트 TV 는 비교적 빠른 입출력 속도를 보장하는 NAND 기반의 스토리지를 사용하므로, 고속 IO 기법을 통해 성능향상을 기대할 수 있다.

6. Acknowledgement

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

참고문헌

[1] H. Volos, and M. Swift, Mnemosyne: Lightweight persistent memory, In Proc. of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Newport, California, USA, Mar. 2011.

[2] A. Dearle, R. Di Bona, J. Farrow, F. Henskens, A. Lindstrom, J. Rosenberg and F. Vaughan, Grasshopper: An orthogonally persistent operating system, Computing Systems, vol. 7, pp. 289--312, Jul. 1994.

[3] J. Eliot, B. Moss, Working with persistent objects: To swizzle or not to swizzle, Software Engineering, IEEE Transactions on, vol. 18, pp. 657--673, Aug. 1992.

[4] J. Guerra, L. Marmol, D. Campello, C. Crespo, R. Rangaswami, and J. Wei, Software persistent memory, In Proc. of the USENIX ATC '12, Boston, MA, June. 2012