스토리지 클래스 메모리를 활용한 시스템의 신뢰성 향상

(Enhancing Dependability of Systems by Exploiting Storage Class Memory)

김 효 진 † 노 삼 혁 ††
(Hyojeen Kim) (Sam H. Noh)

요 약 본 논문에서는 차세대 비휘발성램 기술인 스토리지 클래스 메모리(SCM)와 DRAM을 병렬적으로 메인 메모리로서 도입하고, SCM+DRAM 메인 메모리 시스템을 시스템 신뢰성 측면에서 활용한다. 본 시스템에서는 부팅 없는 즉각적인 시스템 온/오프, 프로세스의 동적인 영속성 또는 비영속성의 선택, 그리고 이를 통하여 전원과 소프트웨어 장애로부터의 빠른 복구를 제공한다. 본 논문에서 제안하는 시스템의 장점은 체크포인팅에서의 문제들, 즉 심각한 오버헤드와 복구 지연을 야기하지 않으며, 특히 응용 프로그램에 대한 완전한 투명성을 제공하기 때문에 보편적인 응용 프로그램에 영속성을 제공할 수 있어 실제 환경에 적용되기가 쉽다. 우리는 이를 검증하기 위해 상용 운영체제인 리눅스 커널 2.6.21을 기반으로 시스템을 구현하였고, 실험을 통해 영속성이 지정된 프로세스가 시스템의 오프-온 후 데이터 손실 없이 즉각적으로 실행을 지속하는 것을 알 수 있었으며, 이를 통하여 우리는 본 시스템에서 가용성과 신뢰성이 향상될 수 있음을 확인하였다.

키워드 : 스토리지 클래스 메모리, 차세대 비휘발성램, 고신뢰성, 프로세스의 영속성, 장애 복구, 즉각적인 시스템 온/오프

Abstract In this paper, we adopt Storage Class Memory, which is next-generation non-volatile RAM technology, as part of main memory parallel to DRAM, and exploit the SCM+DRAM main memory system from the dependability perspective. Our system provides instant system on/off without bootstrapping, dynamic selection of process persistence or non-persistence, and fast recovery from power and/or software failure. The advantages of our system are that it does not cause the problems of checkpointing, i.e., heavy overhead and recovery delay. Furthermore, as the system enables full application transparency, our system is easily applicable to real-world environments. As proof of the concept, we implemented a system based on a commodity Linux kernel 2.6.21 operating system. We verify that the persistence enabled processes continue to execute instantly at system off-on without any state and/or data loss. Therefore, we conclude that our system can improve availability and reliability.

Key words : Storage Class Memory, next-generation non-volatile RAM, dependability, process persistence, failure recovery, instant system on/off

· 이 논문은 2007년도 정부(교육과학기술부)의 재원으로 한국과학재단의 국가지정연구실사업으로 수행된 연구임(No. R0A-2007-000-20071-0)

† 학생회원 : 홍익대학교 컴퓨터공학과
kimhj@necsst.ce.hongik.ac.kr
†† 종신회원 : 홍익대학교 정보컴퓨터공학과 교수
samhnoh@hongik.ac.kr

논문접수 : 2009년 9월 25일
심사완료 : 2009년 10월 13일

정보과학회논문지 : 시스템 및 이론 제37권 제1호(2010.2)

1. 서 론

최근 활발히 연구되고 있는 스토리지 클래스 메모리(Storage Class Memory; SCM)는 배터리의 지원 없이 셀 자체적으로 비휘발성을 띄며 DRAM에 비견되는 접근 속도로 랜덤 접근이 가능한 메모리 기술이다[1]. 이 메모리 기술은 삼성, 인텔, 도시바 등 대부분의 반도체 회사에서 PRAM(또는 PCRAM, PCM), MRAM, FeRAM 등의 형태로 연구/개발되고 있으며 일부는 상용화되어 있다. SCM은 집적도와 전력 소비 효율의 한계에 부딪히고 있는 DRAM과 느린 속도의 디스크를 대체하며

가까운 미래에 일상적 컴퓨팅 구성 요소로서 도입될 것으로 전망되고 있는데[1-3], Freitas 등은 그 시점을 2012년경으로 예상하고 있다[3]. 최근에는 집적도가 매우 좋아 SCM 가운데 가장 유망한 기술로 평가받고 있는 PRAM으로 DRAM을 대체하는 연구들이 여러 연구 그룹에서 수행된 바 있다[4-7].

지난 수십 년간, 접근 속도가 빠른 휘발성램인 DRAM과 접근 속도가 느린 비휘발성의 저장장치인 디스크는 컴퓨터 시스템의 구성요소로서 당연하게 여겨져 왔다. DRAM은 접근 속도가 빠르고 바이트 단위의 접근이 가능하기 때문에 CPU에서 직접 주소를 접근하여 프로세스가 실행된다. 그러나 전원이 꺼지면 DRAM상에 있는 내용이 사라지기 때문에 변경된 내용을 보존하기 위해서는 저장장치에 저장하는 과정이 필요하다. 그러나 디스크 접근에는 기계적 움직임이 수반되므로 접근 속도가 매우 느리기 때문에 디스크에서 데이터를 읽고 쓰는 데에 많은 제약이 있었다. 반면 SCM은 빠른 접근 속도와 임의 접근성 때문에 메인 메모리로서 사용이 가능하면서도 영속적으로 데이터를 보존할 수 있는 비휘발성이기 때문에 기존 시스템에 있었던 여러 제약들이 제거될 수 있어, 여러 장점과 획기적인 기능들을 제공할 수 있다.

일반적으로 새로운 하드웨어를 시스템에 도입하는 것은 다소 부정적으로 여겨지나, 과거를 돌아보았을 때 새로운 하드웨어의 도입으로 인해 획기적이고 편리한 기능이 제공되는 경우에는 하드웨어의 도입이 용인되고 급속하게 시장이 성장해 나갔음을 알 수 있다. 1980년대의 하드디스크의 도입, 2000년대 초반의 플래시 메모리 시장의 확장은 그러한 역사를 잘 대변하고 있다. SCM은 기업들과 연구 기관들의 기대 속에서 시스템 도입과 급속한 시장 형성이 예상되고 있으므로, 이를 효과적으로 활용하기 위한 연구가 필요하다.

우리는 SCM과 DRAM을 병렬적인 메인 메모리로 구성하고, 영속적 프로세스와 비영속적 프로세스가 공존하는 시스템을 구축하여 이를 시스템 신뢰성 측면에서 활용하고자 한다. 본 논문에서 *영속적 프로세스*는 시스템의 전원이 차단되었을 때에도 모든 프로세스 상태를 보존하며, 시스템에 전원이 공급되어 다시 켜질 때 이전 상태에 이어 실행을 지속하는 프로세스를 지칭한다. 반면, *비영속적 프로세스*는 영속적이지 않은 프로세스, 즉, 프로세스 상태의 전부 또는 일부를 시스템 전원이 차단될 때 잃어, 시스템에 전원이 공급되고 다시 켜졌을 때 실행을 지속할 수 없는 프로세스를 지칭한다. 프로세스를 SCM에서 실행시키면 계속해서 변화하는 프로세스의 상태가 저장장치의 접근 없이도 영속적으로 보존되어 영속적 프로세스 구현이 가능하다. 우리는 커널을 영속적인 프로세스로 실행시켜 부팅 없이 수 초 정도의 즉각적인 시스템 온/오프가 가능하도록 하고, 응용 프로그램에 대해서는 사용자가 영속성을 선택할 수 있도록 하였다.

이러한 시스템에서 시스템을 오프-온 하면 영속적 프로세스는 실행을 지속하고 비영속적 프로세스는 종료되며, 온/오프 시간은 매우 짧으므로 빠르게 시스템에 영속적 프로세스만 남기는 것이 가능하다. 이러한 기능은 시스템의 정상 동작 중에도 유용하지만 장애가 발생했을 때에 사용되면 빠른 복구를 시도 할 수 있다. 만일 비영속적 프로세스의 문맥에서 발생한 장애의 원인이 있었다면 시스템 오프-온으로 장애도 함께 사라지므로 시스템이 빠르게 정상화될 수 있다. 또한, 응용 프로그램에게 완전한 투명성을 제공하면서 프로세스 영속성 여부를 동적으로 선택하도록 하여, 보편적인 응용 프로그램에 적용할 수 있다. 따라서 본 시스템이 PC나 노트북과 같이 상대적으로 신뢰성이 취약한 환경에 적용되면 사용자들에게 더욱 향상된 고신뢰성(dependability)을 제공할 수 있다.

우리는 FeRAM이 장착된 임베디드 개발 보드에 리눅스 커널 2.6.21을 기반으로 본 논문에서 제안하는 시스템을 구현하였으며, 즉각적인 시스템 온/오프와 영속성과 비영속적 프로세스들이 설계된 대로 올바르게 동작함을 확인하였다.

이후 논문의 구성은 다음과 같다. 다음 절에서는 본 연구의 필요성 및 관련된 기존 연구들을 설명하고, 3절에서는 시스템의 설계를 논의한다. 4절에서는 구현 방법과 간략한 실험 결과를 언급하고 5절에서 결론을 제시하며 글을 맺는다.

2. 연구의 필요성과 관련 연구

컴퓨터의 보급과 인터넷의 확장으로 사람들이 컴퓨터를 통해 중요한 작업들을 수행하고 데이터를 관리하는 등 일상생활이 컴퓨터에 더욱 의존하게 됨에 따라, 컴퓨터 시스템은 더욱 높은 수준의 신뢰성을 요구받고 있다. 체크포인팅은 시스템의 신뢰성을 향상시키기 위해서 기존의 DRAM과 디스크로 구성된 시스템에서 가장 일반적으로 사용되고 있는 기법으로, 서버 시스템에서 구동되는 중요 응용 프로그램이나 PC 환경에서의 오피스 프로그램 등에서 활용 되고 있다. 체크포인팅은 주기적으로 프로세스의 상태를 저장장치에 저장하고 장애가 발생했을 때 마지막으로 저장된 체크포인트를 기반으로 프로세스를 복구하는 기법인데, 이러한 방식 때문에 체크포인팅은 아래와 같은 여러 한계점들을 가지고 있다.

첫째, 체크포인팅은 저장장치 접근으로 인한 오버헤드 때문에 자주 수행될 수 없다. 예를 들면 libchk 체크포

인팅의 기본 간격은 30분이며[8], 체크포인팅에 적합하게 재설계된 운영체제인 EROS에서도 기본 간격을 5분으로 하고 있다[9]. 그런데 장애가 발생하면 마지막 체크포인트 내용으로 프로세스를 복구하기 때문에 체크포인트 이후에 발생한 데이터에 대해서는 손실이 발생한다. 체크포인팅의 오버헤드 문제를 해결하기 위하여 여러 연구들이 제안되었는데, 변경된 내용만 체크포인팅을 하여 체크포인팅 분량을 줄임으로써 오버헤드를 줄이기 위한 기법[10], 체크포인팅과 프로세싱을 병렬적으로 수행하여 오버헤드를 감추기 위한 기법[11], 새로운 운영체제 설계[9] 등 다양한 방법들이 있으나 이러한 방법들이 완전한 해결책을 제시하지는 못하였다. Lowell 등은 Rio 캐시와 Vista 트랜잭션을 사용하여 디스크 없이 영속적인 메모리에 체크포인팅을 하는 기법을 제안하였는데[12], 이들은 빠른 메모리에 체크포인팅을 했기 때문에 실행 오버헤드를 매우 줄일 수 있었고, 밀리 초 간격으로까지 체크포인팅을 실행할 수 있었다. 그러나 이러한 기법도 체크포인팅에 수반되는 오버헤드를 완전히 없애지는 못하였다.

둘째, 체크포인팅은 장애 시 복구 시간의 지연을 완전히 제거하지 못한다. 기존의 DRAM 메인 메모리 기반에서는 시스템에 장애가 발생하면 체크포인팅으로 프로세스를 복구하기 이전에, 시스템 부팅이 필요하다. 그러나 부팅은 수십 초 내지 수 분이 소요되므로 복구 시간에 상당한 지연이 발생한다. 또한 부팅 이후에도 체크포인팅에서 사용하는 복구 데이터가 디스크에 저장되어 있으므로 느린 장치에 접근하여 이를 읽어오는 과정이 필요하다.

마지막으로, 체크포인팅 기법은 운영체제와 같은 시스템 계층의 수정으로 구현하기가 어렵다[13]. 따라서 응용 프로그램의 코드를 수정하거나 라이브러리 계층에서 구현하여 이를 링크해야 하기 때문에, 응용 프로그램 수정 또는 컴파일이 불가피하므로 응용 프로그램에 완전한 투명성을 제공하지 못하는 문제가 있다. 이러한 문제는 보편적으로 체크포인팅이 도입되는 데 장애 요소가 되며, 특히 상호 연관성이 있는 여러 프로세스들에 대해 전체적인 일관성을 제공하면서 체크포인팅을 할 때에는 이 과정이 매우 복잡하거나 불가능하게 한다. Laadan 등은 가상 머신을 이용하여 관련된 프로세스들의 동시 체크포인팅 기법을 제안하였으나[14], 이 기법에서는 체크포인트마다 프로세스 전체를 중단해야 하므로 그에 대한 오버헤드가 발생한다.

전반적으로, 우리 연구는 하드웨어의 특성을 활용하여 프로세스의 상태를 보존하기 때문에 체크포인팅에서의 프로세스 상태 사본 관리에 수반되는 문제들이 발생하지 않는다는 점이 체크포인팅과 구별된다. 우리는 디스크 접근 없이 SCM에서의 실행만으로 프로세스 상태를 저장하기 때문에 시간적, 공간적 오버헤드가 없으며, 복구할 이미지가 이미 메모리에 존재하고 부팅이 필요 없기 때문에 복구가 빠르다. 또한 상태 보존을 위한 별도의 명령이 없기 때문에 코드 수정도 필요하지 않아 완전한 응용 프로그램 투명성을 제공한다. 그러나 체크포인팅은 디버깅이나 프로세스 마이그레이션 등을 위해서도 사용될 수 있는데, 체크포인팅을 하지 않는 본 시스템은 그러한 기능을 제공하지는 못한다. 그렇지만 기존의 응용 프로그램이나 라이브러리에서 구현된 체크포인팅과는 충돌을 발생시키지 않으므로 추가로 체크포인팅을 구현하고 활용하는 것은 가능하다.

한편, 체크포인팅 없는 빠른 복구 기법들도 연구된 바 있다. Recovery box는 비휘발성램을 이용하여 빠른 복구를 하는데[15], 운영체제나 응용 프로세스의 다운 시간을 줄이기 위하여 중요하면서도 재생이 어려운 정보를 특정 영역에 저장하고, 장애가 발생하면 이 데이터를 사용하여 대상 프로세스를 빠르게 복원한다. 프로세스의 상태를 비휘발성램에 보존하는 것은 우리 연구와 동일하지만, Recovery box에서는 선별적으로 일부 상태만 보존하고 있으며, 그 정보를 Recovery box에서 접근하기 위해서는 응용 프로그램의 코드가 수정되어야 한다. 또한 어떠한 상태를 보존할 것인가를 결정해야 하기 때문에, 응용 프로그램에 대한 깊은 이해가 필요하므로 이 기법이 보편적으로 적용되기에는 어려움이 있다. 반면 우리 연구는 모든 정보를 비휘발성램에 보존하며, 프로그램 코드 수정이 전혀 없다는 차이점이 있다.

Microreboot은 부분적인 재구동을 통하여 전체 재구동과 같은 효과를 얻고자 하는 기법이다[16]. 본 논문도 Microreboot과 마찬가지로 전체를 재구동하기 전에 적은 비용으로 빠르게 복구를 시도한다. 그러나 Microreboot은 한 응용 프로그램에서의 구성요소 또는 모듈 단위를, 본 논문은 시스템에서 프로세스 단위를 그 대상으로 하는데 차이가 있다. 전체적으로 Microreboot도 응용 프로그램에 대한 높은 수준의 지식을 가지고 코드를 수정해야 하나, 우리 연구는 수정이 필요하지 않는다는 차이점이 있다.

최근 SCM은 DRAM과 디스크의 장점을 모두 가지고 있으면서도 높은 집적도로 인해 이의 활용에 대한 관심이 증가하고 있다. SCM은 배터리로부터 전원을 공급받는 기존의 비휘발성램의 활용 방안과 동일하게 저장장치 계층에서 파일 데이터의 안정적인 저장과 속도 향상을 위해서 사용될 수도 있으나, 최근에 발표되고 있는 여러 SCM의 도입 방안들[1-7]이 메인 메모리로서 활용할 것을 제안하고 있는 것은 주목할 만하다. 여러 SCM 메모리 기술들 중 현재 가장 유망한 기술은 PRAM으로

서 집적도가 월등히 좋아 단위 용량 당 가장 낮은 가격대를 형성할 것으로 예상되고 있다. 그런데 PRAM의 접근 속도(약 100ns)와 쓰기 횟수 제한(10^8-10^12회)이 DRAM보다는 낮기 때문에 Lee 등, Zhou 등, Qureshi 등은 PRAM을 메인 메모리로 활용하기 위해서 이러한 문제들을 아키텍처 관점에서 해결 하고자 하였다[4-6]. Mogul 등은 DRAM과 PRAM을 병렬적으로 구성하고 쓰기가 자주 일어나는 메모리 객체는 DRAM에서, 그렇지 않은 메모리 객체는 PRAM에서 할당되도록 함으로써 PRAM의 쓰기 횟수 제한 문제를 운영체제 계층에서 해결하였다[7]. 그러나 현재까지 제안되었던 PRAM을 비롯한 SCM의 메인 메모리로서의 도입은 DRAM과 가장 차별되는 비휘발성을 제대로 활용하지 않고 있으며, 집적도 측면의 장점만을 활용하였다. 이에 반해 본 논문은 SCM을 메인 메모리로 사용하면서 SCM의 비휘발성을 효과적으로 활용하는데 초점을 맞춘다.

3. 시스템 설계

본 연구는 다음의 두 가지 원칙에 기반을 두고 설계되었다.

- **사용자가 미리 지정한 일부 프로세스에 대한 복구를 목표로 한다.** 즉, 지정된 프로세스에 대해서만 선택적으로 복구를 시도하며, 이는 모든 프로세스에 대한 복구를 목표로 하는 것은 아님을 의미한다.
- **빠른 복구를 목표로 하나, 복구를 보장하는 것은 아니다.** 즉, 지정된 프로세스에 대한 빠른 복구를 목표로 하나, 지정된 프로세스에 장애가 있는 경우 복구에 실패할 수 있고, 그러한 경우 시스템 전체의 재부팅을 필요로 할 수도 있다.

첫 번째 원칙은 프로세스의 중요도에 따른 차별적 복구 정책을 의미한다. 시스템의 모든 프로세스가 동등하게 중요하지는 않으며, 복구되지 않아도 무방한 프로세스와 꼭 복구되어야 하는 프로세스가 혼재한다. 따라서 복구되지 않아도 무방한 프로세스들을 빠르게 희생시켜 문제 환경을 최대한 축소하고 중요한 프로세스들에 집중한다. 지정된 프로세스의 내부적 결함에 대한 대처를 하기 위해서는, 해당 프로그램 코드 레벨에서 Recovery Box[15], Microreboot[16], 체크포인팅 등 빠른 복구 기법을 적용하여 복구 시간을 단축시키거나, 효과적인 디버깅 등으로 결함을 줄여 장애를 예방하는 방법, 결함을 효과적으로 검출하기 위한 테스팅 및 진단 방법 등 여러 신뢰성 기법들을 적용하여 신뢰성을 향상시킬 수 있다. 그러나 이 부분은 독립적인 연구 주제 이므로 본 논문에서는 이후 더 이상 언급하지 않는다.

두 번째 원칙은 본 논문에서 제안하는 시스템이 복구를 보장하지는 않지만 장애 상황에서 재부팅에 앞서 첫 번째로 시도해 볼 수 있는 치료법을 제공한다는 것을 의미한다. 우리는 시간이 오래 걸리면서도 프로세스의 상태를 잃는 재부팅 이전에, 상당한 가능성을 가지고 데이터의 손실 없이 복구를 할 수 있으면서도 소요 시간이 짧은 방법을 먼저 시도하여 시스템의 가용성을 향상시킨다.

본 연구는 앞서 언급한 원칙하에 다음의 네 가지 목표를 가지고 설계되었다.

- **지정 프로세스에 대한 완전한 복구:** 장애 발생 시 그 원인이 영속성이 지정된 프로세스들의 문맥상에 있지 않으면, 영속성이 지정된 프로세스들은 데이터 손실이 전혀 없이 복구되어야 한다.
- **정상 실행 동안 적은 오버헤드:** 정상 실행 동안 영속성을 제공하기 위하여 필요한 실행 시간 및 공간의 오버헤드는 최소화되어야 한다.
- **완전한 응용 프로그램 투명성:** 기존의 응용 프로그램들의 수정 없이 영속성 지원이 가능하도록 한다. 즉, 응용 프로그램 코드 수정 및 재 컴파일 없이 영속성을 지원하도록 하여 완전하게 응용 프로그램 투명성을 제공한다.
- **빠른 장애 복구:** 복구 시도 시간이 매우 짧아야 하며, 만일 제안하는 방법으로 시스템이 정상화 되지 않더라도 전체 복구 시간에 큰 영향을 주지 않아야 한다.

이러한 목표들을 만족시키기 위하여 우리는 그림 1과 같이 SCM을 DRAM과 병렬적인 메인 메모리로 구성하였다. 이러한 시스템 구성의 결과, 전원을 완전히 차단한 이후 시스템을 가동할 때 부팅이 필요 없는 즉각적인 시스템 온/오프, 프로세스 영속성의 동적인 선택, 빠른 시스템 정상화 등의 가능들이 제공된다. 다음에서 이들 각 기능에 대해 설명한다.

3.1 즉각적인 시스템 온/오프

우리는 시스템의 전원이 완전히 꺼진 상태를 시스템 오프, 시스템의 전원이 공급된 후 운영체제가 서비스를 할 수 있는 가용 상태를 시스템 온으로 지칭한다. 본 시스템에서는 부팅과 셧다운 과정을 수반하지 않는 시스템 온/오프를 제공하기 때문에 시스템 온/오프와 부팅 및 셧다운은 구분된다.

본 연구는 SCM을 메인 메모리로 사용하고 커널을 SCM에서 실행시킴으로써 즉각적인 시스템 온/오프를 제공한다. 한번 SCM에 커널이 부팅된 후, 커널 스레드가 SCM에서 실행되면 레지스터를 제외한 커널 상태는 전원이 꺼지더라도 비휘발성인 SCM에 보존된다. 이후, 전원을 끄기 전에 휘발성 데이터인 레지스터 값을 SCM 특정 영역에 저장하면, 일관된 상태를 보존하면서 시스템을 오프할 수 있다. 시스템 오프 명령을 수행시키면 명시적으로 안전하게 시스템 오프를 할 수 있으며, 갑작스럽게 전원이 차단되었거나 전원 버튼을 눌러 비 명시

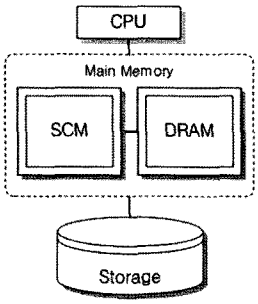

그림 1 즉각적인 시스템 온/오프

적으로 시스템이 오프될 때는 하드웨어의 도움을 받아 전압 감소가 감지되면 축전기로 전원을 공급받으며 인터럽트를 발생시켜 레지스터를 보존하는 루틴을 수행시키도록 하는 방법을 통해 시스템 오프를 할 수 있다. 이후, 시스템이 온 될 때 수정된 부트로더가 SCM의 특정 영역에서 레지스터 값을 제 위치에 되돌려 놓으면 시스템을 오프하기 이전 상태로 돌아가 운영체제가 가용상태가 된다. 이러한 시스템 온 절차에는 부팅 과정이 포함되지 않으므로 매우 빠른 온/오프가 가능하다. 한편, 전원이 공급된 후 부트로더가 구동될 때, 만일 SCM의 특정 영역에 레지스터 값이 없으면 부트로더는 일반적인 부팅을 위한 루틴을 시작한다. 본 연구의 즉각적인 온/오프는 SOONN[17]의 온/오프 과정과 동일하나, SOONN에서의 환경은 모든 메인 메모리가 SCM으로 대체 되어 있고 본 논문에서는 SCM과 DRAM이 메인 메모리상에 공존하고 있다는 차이점이 있다.

3.2 프로세스의 영속성과 비영속성의 동적인 선택

우리가 제안하는 시스템에서 프로세스의 영속성은 프로세스의 메모리를 SCM에서 할당 받아 실행시킴으로써 가능하다. 프로세스의 상태는 주소공간, 커널 상태, 레지스터 값의 세 가지 형태로 구성된다. 주소공간과 커널 상태는 비휘발성인 SCM에서 실행하는 동시에 영속적으로 보존되며, 프로세스의 레지스터 값은 문맥교환 과정에서 SCM에 보존되고 시스템 오프 시 실행 중이던 프로세스의 레지스터 값은 시스템 오프 과정에서 SCM에 저장된다. 비영속적 프로세스는 기존 운영체제에서와 같이 DRAM에서 실행됨으로써 구현된다.

이와 같이 시스템 계층에서 비휘발성램을 할당하여 프로세스를 실행함으로써 영속성을 제공하면 응용 프로그램이 수정될 필요가 전혀 없다. 우리 시스템은 프로세스의 영속성에 따라 시스템에서 정책적으로 SCM을 할당하며, 이러한 할당은 응용 프로그램에 노출되지 않으므로 응용 프로그램 코드를 수정하지 않을 뿐만 아니라 컴파일 과정도 필요하지 않아 완전한 응용 프로그램 투명성을 제공한다. 따라서 응용 프로그램에 제한없이 영속성이 제공될 수 있다.

완전한 응용 프로그램 투명성은 운용 환경에 따라 동적으로 응용 프로세스의 영속성/비영속성을 선택하는 것이 가능하게 한다. 그러면 이러한 선택을 누가할 것인가에 대한 결정이 필요한데, 그 결정은 구현에 따라 다양할 수 있지만 우리는 사용자가 유저 프로세스의 영속성을 결정하도록 하였다. 운용 환경의 다양성을 고려할 때 사용자에게 지정 권한이 있으면 효과적으로 시스템을 운용할 수 있다. 기본적으로 어떠한 프로세스에도 영속성을 지정할 수 있지만 중요한 데이터를 다루는 응용 프로그램이나 신뢰성이 어느 정도 검증된 응용 프로그램에 영속성을 지정하면 시스템을 좀 더 효과적으로 활용할 수 있다.

다만, SCM에서 할당 받아야 하는 영속적 프로세스가 너무 많으면 동작이 느려질 수 있으므로, 이를 고려하여 적절한 개수의 영속적 프로세스를 시스템에 유지해야 한다. 특히, SCM의 도입 초기에는 높은 가격으로 인해 시스템에 구성되는 SCM의 크기가 제한될 수 있어 영속적 프로세스의 개수에도 제한이 있을 수 있다. 우리 기법은 이처럼 DRAM 대비 적은 양의 SCM으로 시스템을 구성할 수 있기 때문에, SCM 도입 초기에 더욱 적절한 활용 모델이라고 할 수 있다. 뿐만 아니라 2020년경에는 SCM이 디스크를 대체할 정도의 가격 경쟁력을 갖출 것으로 전망되므로[2], 가격적인 부분은 시간이 지남에 따라 무난히 해결될 것으로 예상된다.

3.3 빠른 시스템 정상화

즉각적인 시스템 온/오프와 프로세스의 영속성/비영속성의 동적인 선택을 통해 우리는 장애에서 시스템을 빠르게 정상화 할 수 있다. 시스템에 장애가 발생하면 시스템을 셧다운 하기에 앞서, 즉각적인 온/오프로 영속적 프로세스를 보존시키고 비영속적 프로세스는 종료한다. 그러면, 만일 비영속적 프로세스에 시스템의 장애를 일으킨 원인이 있었던 경우, 장애가 비영속적 프로세스와 함께 사라진다. 본 시스템은 즉각적인 시스템 온/오프가 가능하므로 매우 빠른 시간 안에 이 모든 과정이 이뤄질 수 있다. 그러나, 만일 영속적 프로세스에 장애 원인이 있었을 경우에는 전원 차단 이후에도 장애가 지속되므로 부팅이 필요할 수 있다. 선택적 프로세스 영속성을 지원하는 시스템에서의 시스템 온/오프는 빠른 DRAM의 메모리 리쥬브네이션(rejuvenation)을 가능하게 한다. 메모리 리쥬브네이션은 메모리 공간을 초기화 시키는 것인데, 이 과정만으로도 메모리 누수(leak) 등의 메모리 관련 오류를 예방하는 등 장애 예방 효과가 높다[8]. 우리 시스템은 영속적 프로세스를 지속적으로 구동하면서도 휘발성램에 대해서 빠른 시간 안에 초기화 할 수 있기 때문에, 중요한 프로세스의 멈

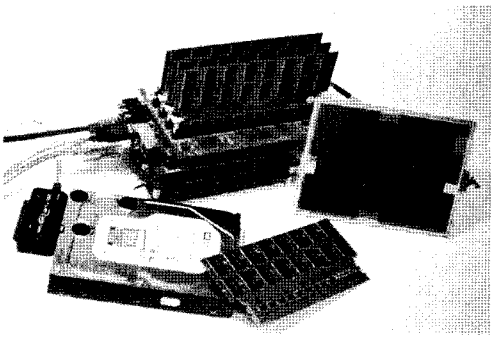

그림 2 EZ-X5 개발보드

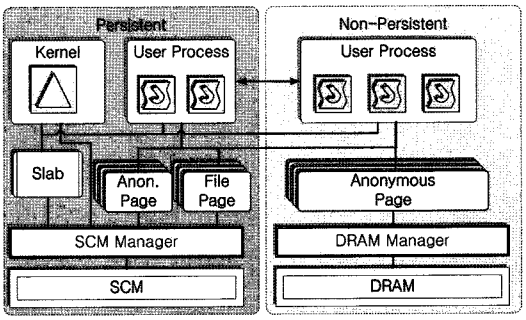

그림 3 메모리 시스템의 구조

춤 없이 효과적으로 DRAM 리쥬브네이션이 가능하다. 리쥬브네이션은 시스템 장애가 발생하지 않은 상황에서 장애 예방으로 수행될 수 있지만, 장애가 발생한 경우에도 메모리 초기화로 비영속적 프로세스의 재구동 환경이 변화되어 이전에 장애를 발생시킨 결함이 활성화되지 않을 수 있다[8].

4. 구현 및 검증

본 논문에서 우리는 SCM과 DRAM이 공존하는 메인 메모리에서 시스템을 설계하였으나, 현재 DRAM과 SCM이 공존하는 보드가 상용화되어 있지 않기 때문에, 우리는 그림 2와 같이 임베디드 시스템 개발 보드를 설계하였다. 개발보드는 EZ-X5 PXA255 XScale 400MHz MCU, 64MB의 SDRAM, 64MB NAND Flash 메모리를 내장하고 있으며, 도터보드(daughter board)에는 네 개의 슬롯을 두어 16MB~64MB의 FeRAM을 장착할 수 있는데, 본 논문에서는 커널을 SCM에서 실행시키기 위하여 32MB 이상을 장착하였다.

우리는 제안하는 시스템을 리눅스 커널 2.6.21을 기반으로 구현하였는데, 리눅스를 선택한 이유는 리눅스가 널리 사용되고 있는 오픈 소스 운영체제이기 때문이다. 본 논문에서는 약 3000 줄의 코드만을 수정 또는 추가하였으므로 수정 복잡도가 높지 않다. 본 논문에서는 지면의 제약으로 인해 구현내용은 간략히 설명하고 있으나, 좀 더 자세한 내용은 [18]에서 볼 수 있다.

그림 3은 영속적 프로세스와 비영속적 프로세스가 공존하기 위한 SCM/DRAM 메인 메모리 시스템의 구조를 간략히 도식화하여 보여준다. 구현에 있어서, 개발보드에서 두 가지 메모리를 동시에 인식하지는 못하기 때문에 우리는 SCM을 커널의 기본 메모리로 인식되게 하여 관리하고, DRAM은 부팅 과정에서 ioremap()을 사용하여 추가적으로 인식한 후, 리눅스의 버디 시스템 코드를 활용하여 작성된 추가적 메모리 관리자가 관리하도록 하였다. SCM에는 커널 쓰레드를 비롯한 영속적 프로세스들의 모든 객체를 할당하고, 비영속적 프로세스의 파일 페이지와 슬랩 캐시도 SCM에서 할당된다. 현재 구현에서는 비영속적 프로세스의 익명 페이지만 DRAM에서 할당하고 있지만, 향후에 SCM을 효율적으로 사용하기 위하여 클린 파일 페이지들은 DRAM에서 할당하도록 변경할 예정이다. 모든 슬랩 캐시를 SCM에서 할당하는 이유는 커널의 무수한 자료 구조가 슬랩 캐시로부터 메모리를 할당 받기 때문에 호출 빈도는 매우 높으나 할당받는 메모리의 크기는 작기 때문으로, 코드 수정의 복잡성과 메모리 사용량의 효용을 고려하여 결정하였다.

프로세스는 부모 프로세스의 영속성을 고려하여 생성된다. 기본적으로 영속적 프로세스는 자식 프로세스에게 영속성을 상속하고, 비영속적 프로세스는 비영속성을 상속한다. 그러나 쉘 프로세스는 예외적이다. 우리는 빠른 시스템 온/오프를 위하여 커널 프로세스와 함께 쉘처럼 부팅 중에 생성되는 유저 프로세스를 영속적으로 실행되도록 하였는데, 만일 쉘의 자식 프로세스들이 영속성을 부여받으면 시스템의 모든 프로세스가 영속적으로 실행된다. 따라서 우리는 쉘 프로세스는 영속성을 상속하지 않도록 하고 부팅 이후 생성되는 유저 프로세스들은 기본적으로 비영속성을 갖도록 하였다.

우리는 영속성을 지정 또는 해제하는 명령 프로그램을 제공하는데, 이 명령 프로그램을 통해 프로세스에 영속성이 지정되면 대상 프로세스에 대하여 DRAM에 할당되어 있는 페이지들을 SCM으로 옮기고, 프로세스의 비영속성을 영속성으로 변경하여 표시한다. 이 때, 운용의 편리성을 위하여 대상 프로세스에 자식 프로세스들이 있으면 한 번의 영속성 지정으로 자식 프로세스들까지 함께 영속적 프로세스로 변경한다. 이러한 과정은 프로세스의 페이지 테이블 스캔 및 기 할당된 페이지에 대한 메모리 복사를 야기한다. 우리 개발 환경에서는 수 밀리 초 범위 내에서 이루어졌으며, 프로세스에 할당된 페이지의 수 및 자식 프로세스의 수 등의 요소가 소요 시간에 영향을 미친다. 그러나 기대 소요 시간이 매우

짧으며 더욱이 일회성으로 발생한다.

우리는 이와 같이 구현된 시스템에 대해 몇 가지 초기적 실험을 하여 본 논문에서 제안하고 있는 고신뢰 시스템을 검증하였다. 다만, 현재는 전원 버튼을 누르거나 전원이 차단되었을 때 전압의 감소를 감지하고 레지스터 정보를 보존하여 시스템을 오프하는 기능이 구현되지 않았기 때문에, 장애 복구에 대한 실험은 수행하지 못하였다. 그러나 명시적인 명령을 통해 시스템 오프-온을 수행한 다음의 실험들을 통해 빠른 장애 복구의 가능성을 확인하였다. 본 논문의 구성 환경에서 시스템 온/오프는 네트워크 연결이 없을 때에는 0.03초 이내, 네트워크 연결을 포함하면 약 1초정도 소요되었다. 또한, 프로세스에 대한 영속성이 잘 지원 되는가를 확인하기 위해 커널 소스 디렉토리에서 tar 명령을 수행하는 동안 시스템을 오프-온해 보았다. 그리고 다시 그 결과로 생성된 파일을 untar를 하면서 디렉토리 전체에 대해 tar/untar가 올바르게 되었는지 확인하였다. 그리고 tar 명령으로 생성된 커널 코드 파일에 대해 gzip과 gunzip을 같은 방법으로 실험하여 올바르게 동작하고 있음을 확인하였다. 그리고 관련 있는 여러 프로세스들의 동적인 영속성 지정 및 해제가 올바르게 이뤄지는지 검증하기 위해 httpd 데몬에 영속성을 지정한 후 시스템 오프-온을 하고, 영속성을 해제하고 시스템 오프-온을 하여 httpd 데몬이 각각 영속적과 비영속적으로 올바르게 동작함을 확인하였다.

5. 결 론

본 연구에서는 SCM을 시스템의 신뢰성을 향상시키는데 활용하기 위하여 동적으로 프로세스의 선택적 영속성을 제공하고, 영속성이 지정된 프로세스들에 대해 가용성을 향상시키는 기법을 제안하였다. 제안된 시스템을 검증하기 위해 우리는 리눅스 커널 2.6.21을 기반으로 SCM과 DRAM이 공존하는 메인 메모리 시스템을 구축하고, 영속적 프로세스는 SCM에서, 비영속적 프로세스는 DRAM에서 각각 실행되도록 하였다. 이 시스템에서는 즉각적인 시스템 온/오프를 제공하며, 응용 프로그램의 코드 수정 및 재컴파일 없이 영속성을 지원한다. 따라서 복구하고자 하는 중요한 프로세스에 영속성을 지정하여 시스템 장애가 발생했을 때 데이터 손실과 지연 없이 빠른 복구를 시도해볼 수 있다. 이러한 기능이 제공되는 시스템에서는 시스템 전체적인 관점에서 봤을 때 가용성이 향상되어 사용자에게 고신뢰성을 제공할 것으로 기대된다.

참 고 문 헌

[1] G. W. Burr, B. N. Kurdi, J. C. Scott, C. H. Lam, K. Gopalakrishnan, and R. S. Shenoy, "Overview of Candidate Device Technologies for Storage-Class Memory," *IBM Journal of Research and Development*, vol.52, no.4, pp.449-464, 2008.

[2] R. F. Freitas and W. W. Wilcke, "Storage-Class Memory: the Next Storage System Technology," *IBM Journal of Research and Development*, vol. 52, no.4, pp.439 - 447, 2008.

[3] R. F. Freitas, W. W. Wilcke, B. Kurdi, and G. Burr, "Storage Class Memory, Technology and Uses," Tutorial In USENIX FAST, 2009.

[4] B. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," In *Proceedings of the ACM ISCA*, pp.2-13, 2009.

[5] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology," In *Proceedings of the ACM ISCA*, pp.14-23, 2009.

[6] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable High Performance Main Memory System Using Phase-Change Memory Technology," In *Proceedings of the ACM ISCA*, pp.24-33, 2009.

[7] J. C. Mogul, E. Argollo, M. Shah, and P. Faraboschi, "Operating System Support for NVM+ DRAM Hybrid Main Memory," In *Proceedings of the USENIX Workshop on Hot Topics in Operating Systems*, 2009.

[8] Y. M. Wang, Y. Huang, K. P. Vo, P. Y. Chung, and R. Kintala, "Checkpointing and Its Applications," In *Proceedings of the IEEE Fault Tolerant Computing Symposium*, pp.22-31, 1995.

[9] J. S. Shapiro and N. Hardy, "EROS: A Principle-Driven Operating System from the Ground Up," *IEEE Software*, vol.19, no.1, pp.26-33, 2002.

[10] E. N. Elnozahy, D. B. Johnson, and W. Zwaenepoel, "The Performance of Consistent Checkpointing," In *Proceedings of the Symposium on Reliable Distributed Systems*, pp.39-47, 1992.

[11] K. Li, J. F. Naughton, and J. S. Plank, "Low-Latency, Concurrent Checkpointing for Parallel Programs," *IEEE Transactions on Parallel and Distributed Systems*, vol.5, no.8, pp.874-879, 1994.

[12] D. E. Lowell and P. M. Chen, "Discount Checking: Transparent, Low-Overhead Recovery for General Applications," Technical Report CSE-TR-410-99, University of Michigan, December 1998.

[13] G. Bronevetsky, D. Marques, K. Pingali, P. Szwed, and M. Schulz, "Application-level Checkpointing for Shared Memory Programs," In *Proceedings of the ACM ASPLOS*, pp.235-247, 2004.

[14] O. Laadan and J. Nieh, "Transparent Checkpoint-Restart of Multiple Processes on Commodity Operating Systems," In *Proceedings of the USENIX Annual Technical Conference*, pp.323-336, 2007.

[15] M. Baker and M. Sullivan, "The Recovery Box:

Using Fast Recovery to Provide High Availability in the UNIX Environment," In *Proceedings of the USENIX Summer Conference*, pp.31-43, 1992.

[16] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox, "Microreboot - A Technique for Cheap Recovery," In *Proceedings of the USENIX OSDI*, pp.31-44, 2004.

[17] Y. J. Moon, I. H. Doh, J. Park, and S. H. Noh "Development of an Instant On System Using Storage Class Memory," In *Proceedings of the KIISE Korea Computer Congress*, vol.36, no.1(A), pp.336-337, 2009 (in Korean).

[18] H. Kim, E. Kim J. Choi, D. Lee, and S. H. Noh, "Design and Implementation of Selective Process Persistence by Exploiting Storage Class Memory," In *Proceedings of the KIISE Korea Computer Congress 2009*, vol.36, no.1(A), pp.338-343, 2009 (in Korean).

김 효 진

2002년 홍익대학교 컴퓨터공학과(학사) 2007년 홍익대학교 컴퓨터공학과(석사) 2007년~현재 홍익대학교 컴퓨터공학과 박사과정. 관심분야는 운영체제, 차세대 비휘발성램, 고신뢰 시스템

노 삼 혁

1986년 서울대학교 컴퓨터공학과(학사) 1993년 메릴랜드 대학교 컴퓨터과학과(박사), 1993년~1994년 조지워싱턴대학교 객원조교수. 1994년~현재 홍익대학교 정보컴퓨터공학부 교수. 관심분야는 운영체제, 플래시 메모리 소프트웨어, 차세대 비휘발성램