Journal of the Korean Data & Information Science Society
2009, **20**(1), 125–137

한국데이터정보과학회지

스마트카드용 고성능 자바가상기계에 대한 연구[†]

정민수[1]

경남대학교 컴퓨터공학과

접수 2008년 12월 28일, 수정 2009년 1월 13일, 게재확정 2009년 1월 17일

요 약

스마트카드는 작은 크기의 마이크로 컴퓨터칩을 내장하고 있다. 이 칩은 프로세서, RAM, ROM, 클럭, 버스 그리고 암호전용 코프로세서 등을 포함하고 있다. 따라서 이 칩은 RFID 태그와 비교해서 가격이 비싸고, 복잡하지만 안전한 칩이다. 스마트카드의 주요 응용분야는 전자뱅킹이나 안전한 통신 관련 분야이다. 자바카드는 개방형 플랫폼 중 가장 널리 사용되는데 그 이유는 자바카드의 보안성, 플랫폼 독립성, 그리고 빠른 개발 싸이클 때문이다. 하지만 자바카드는 실행속도가 느리기 때문에 자바카드의 성능개선은 중요한 연구 분야가 되어왔다.

본 논문에서는 효과적인 트랜잭션버퍼 관리 방법을 제안하여 자바카드의 성능을 개선시켰으며 실험을 통하여 그 성능을 입증하였다.

주요용어: 스마트카드, 자바카드, 트랜잭션버퍼.

1. 서론

스마트카드는 외형은 신용카드와 같지만 마이크로프로세서, 보안 처리기와 메모리를 내장한 칩을 가지고 있어서 카드 내에서 안전한 정보의 저장과 처리가 가능하다. 최근에 나타난 RFID에 비해서는 가격도 상대적으로 비싸고 무선 인식 범위도 좁지만, 전자상거래 등의 안전한 데이터의 보호가 필요한 분야에 널리 사용되고 있다. 현재 스마트카드는 카드 형태의 금융카드 및 교통카드와 휴대폰에 탑재되는 통신용 태그 형태로 시중에 널리 사용되고 있다.

스마트카드는 크게 폐쇄형 카드와 개방형 카드로 나누어진다 (탁승호, 2004; 한진희, 2003). 폐쇄형 카드는 카드에 탑재된 CPU전용 기계어 프로그램이 탑재되어 있어 연산 처리 속도가 빠른 장점을 가지는 반면, 특정 분야에만 사용되는 단점이 있다. 반면 개방형 카드는 폐쇄형 카드와 달리 연산 처리 속도는 느리지만 다양한 어플리케이션의 탑재가 가능하다는 장점을 가진다. 또한 급격히 바뀌는 스마트카드 칩 플랫폼의 영향을 받지 않아서 프로그램의 개발이 쉽고, 이식이 간편하다. 따라서 현재 및 향후에는 개방형 카드가 주류를 이루고 있을 것이다.

개방형 스마트카드의 운용 플랫폼으로는 멀토스 플랫폼, 스마트카드 윈도우즈, 자바카드 플랫폼이 있다. 자바카드 플랫폼의 경우는 칩 내부에 초소형 자바운영시스템을 탑재한 것으로서, 개방형 API인 자바 카드 API를 제공하는 장점이 있고, 다중 응용 프로그램 지원, 응용 프로그램의 사후 발급 (Post-issuance) 기능, ISO 7816과 같은 국제 표준과의 호환성 제공 등의 장점이 있다. 현재 개방형 스마트카

[†] 이 논문은 2007년도 경남대학교의 연구년 연구비 지원에 의한 것임.
[1] (631-701) 경상남도 마산시 월영동 449번지, 경남대학교 컴퓨터공학과, 교수.
E-mail: msjung@kyungnamu.ac.kr

드 플랫폼의 절반 정도를 자바카드 플랫폼이 차지하고 있다. 하지만 스마트카드의 자원 제약에 상대적으로 무거운 자바 기능을 지원해야 하기 때문에 처리 속도가 느리다는 단점이 있다.

따라서 자바카드의 성능 개선은 대단히 중요하다. 기존의 연구들을 분석해보면 스마트카드 내에 탑재된 자바카드가상기계의 성능을 높이는 여러 가지 방법들이 제안되었으며 대표적으로는, 버퍼 캐시 (원천기술사인 미국 SUN 방법), 더블 버퍼 캐시 (최원호, 2005), RAM 레졸루션 (진민식, 2006a), RAM 이용 객체 (Yang, 2006)방법들의 방법들이 제안되었다.

본 논문에서는 스마트카드용 자바가상기계의 성능을 높이기 위하여 카드 동작과정에서 병목현상을 일으키고 있는 트랜잭션 버퍼의 기록 시간을 줄일 수 있는 방법을 제시하고, 실험을 통해서 기존의 연구와 비교하였다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구를 수록하였고, 3장에서는 스마트카드용 자바가상기계의 성능개선 방법을 제안하였고, 4장에서는 구현 및 실험결과를 보였다. 마지막으로 5장에서는 결론을 기술하였다.

2. 관련연구

2.1. 스마트카드 및 자바카드

스마트카드는 1968년 Jergen Dethloff가 휴대 가능한 초소형 IC 카드의 개념을 제안한 후, 1974년 Roland Moreno가 최초로 형상화하여 프랑스 전기통신공사에서 처음 도입하여 사용하였다. 스마트카드는 카드 내에 IC칩을 내장하고 있으며, 유럽에서는 SIM (Subscriber Identification Module), 미국에서는 UIM (User Identification Module)이라 부른다. 이들 칩은 5*5 mm 이하의 작은 크기이지만 8, 16, 32 비트 프로세서, 그리고 DES 혹은 RSA 같은 보안지원 프로세서, 내부버스, 클럭, RAM, ROM등의 장치와 이들 장치를 운용하는 간단한 운영체제 (COS; Chip Operating System)를 갖춘 초소형 마이크로시스템이다 (Huang, 2007; Sharp, 2005).

자바카드는 상기의 스마트카드의 운용체제에 자바가상기계와 자바 API를 포함하는 자바플랫폼을 탑재한 것으로서 다음과 같은 장점을 가지고 있다. 첫째, 자바 카드 애플릿은 스마트카드 하드웨어 플랫폼에 독립적이므로, 자바 카드 가상기계가 탑재되어 있다면 자바 카드 애플릿의 구동이 가능하다. 둘째, 높은 보안성을 가진다. 자바 카드가 기반하고 있는 자바 언어 자체의 보안적인 특성의 많은 부분을 수용하고 있고, 자바 카드 내부의 독립적인 애플릿들은 방화벽에 의해 보호된다. 셋째, 애플릿의 개발 시간과 비용이 절감된다. 자바 카드 기술은 자바 언어를 기반으로 하고 있으며, 스마트카드 어플리케이션을 개발하는데 필요한 공통적인 API를 제공함으로써 보다 효율적인 개발환경을 제공한다. 넷째, 애플릿의 추가 및 삭제가 용이하다. 다섯째, 다중 어플리케이션을 탑재할 수 있다. 자바 카드는 한 장의 카드에 다수의 어플리케이션을 탑재하는 것이 가능하며, 이러한 작업을 보다 안전하고 간편하게 함으로써 스마트카드가 더욱 다양한 분야에서 활용될 수 있도록 하고 있다 (Baentsch, 1999; Sun Microsystems, 2003a; Chen, 2000a).

2.2. 자바카드에서의 두 가지 객체

자바카드에서는 제한된 자원을 효율적으로 사용하기 위하여 데이터의 특성에 따라 지속적으로 값을 유지해야하는 영속객체 (persistent object)와 임시적으로 사용하는 임시객체 (transient object)로 구분하여 사용한다. 영속 객체는 카드 세션 동안 객체가 가지는 상태 정보를 유지하는 객체로서, 카드 세션이 끝나더라도 데이터는 EEPROM에 저장되어있다. 카드에 전원이 차단되거나 카드 리더기에서 카드

가 빠지는 경우, 영속 객체는 그 상태와 데이터들을 지속적으로 유지한다. 그러나 임시객체는 데이타 값이 유지되지 않는다. 대신 비휘발성 메모리에 비해 훨씬 빠른 기록이 가능한 RAM을 사용한다.

자바언어에는 <static>으로 선언된 정적 변수와 일반 인스턴스 변수, 그리고 함수 내에서 선언되어 사용되어지는 지역변수의 세 가지 종류가 있다. 이 중에서 정적 변수 및 인스턴스 변수는 영속 객체로 분류하여 그 값을 비휘발성 메모리 내에 저장한다. 반면에 지역변수들은 휘발성 메모리인 RAM 내의 자바스택 구역을 두어 보관하므로 임시객체이다. 그 외에도 자바카드에서는 특별한 함수를 두어 임시객체를 생성할 수 있다.

2.3. 자바카드에서 객체의 저장방법

자바 카드에서 사용되는 두 가지 객체인 영속 객체와 임시 객체의 생성은 저장되는 메모리 종류와 객체의 특성에 따라 차이가 있다. 영속객체는 인터프리터의 'new'라는 바이트 코드 명령에 의해 생성된다. 자바 카드 가상기계는 생성할 객체에 관한 정보를 바이트 코드들을 분석하여 생성할 객체의 형태를 정하고, 객체 내에서 사용될 지역 변수의 타입과 개수를 파악한 후 실제 객체의 헤더와 연산을 하여 객체의 크기를 구한다. 객체의 크기가 구해지면 자바 카드 가상기계가 객체를 위한 메모리 공간을 요구하고, EEPROM의 힙 영역 (Spivak과 Toledo, 2006)에 객체를 생성한다. 마지막으로 객체를 관리하는 객체 테이블에 생성된 객체의 위치 정보를 기록함으로써 객체의 생성과정은 완료된다 (Sun Microsystems, 2003b; Chen, 2000b).

임시객체인 경우에는 객체의 기본적인 정보인 객체 헤더와 지역변수들은 EEPROM에 저장되고, 실제 생성된 객체를 사용할 때에는 RAM에 애플릿 수행 시 필요한 데이터들을 기록하여 사용한다.

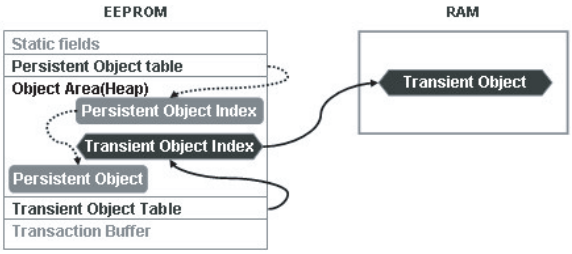

그림 2.1 객체와 테이블 간의 관계

그림 2.1에서 보는 것과 같이 각 객체들은 힙 (Heap)영역에 순서 없이 생성되고, 객체가 생성된 주소는 객체의 종류에 따라 해당 테이블에 저장된다. 영속 객체가 생성될 경우 생성된 객체의 주소가 영속 객체 인덱스 (Persistent Object Index)에 먼저 저장되고, 그 뒤 영속 객체 인덱스에 저장된 객체의 주소를 영속 객체 테이블 (Persistent Object Table)에 저장하고 식별자 (ID)를 할당하여 관리한다. 그리고 임시 객체가 생성될 경우 영속객체와 동일한 순서로 해당 테이블에 저장되고 관리되지만 생성되는 객체의 형태가 다르다. 임시 객체는 EEPROM과 RAM 양쪽에 데이터를 저장하기 때문에 객체를 생성할 때 RAM에는 데이터를 그리고 EEPROM에는 RAM에 저장된 데이터의 위치를 가리키는 참조주소 (Reference)와 함께 객체를 생성한다.

3. 스마트카드용 고성능 자바가상기계

3.1. 자바가상기계에서 트랜잭션버퍼의 중요성

트랜잭션 버퍼는 카드 내부에 영속 객체가 생성되거나 영속 객체가 가지는 값이 변경이 되어서 EEPROM에 값을 저장하고자 할 때에, 데이터를 기록하는 과정 중에 발생할 수 있는 프로그램 상의 오류나 전원 차단 등으로부터 자바 카드의 데이터를 변경 전의 상태로 복구할 수 있도록 해주는 장치이다. EEPROM의 경우, 통상 바이트나 워드 단위가 아닌 수백바이트 크기의 블록단위의 쓰기로 이루어진다. 통상적인 자바카드에서 이 트랜잭션 버퍼는 일반적으로 2 K바이트 정도의 크기를 가진다 (Sun Microsystems, 2003c; Chen, 2000c; Beckert와 Mostowski, 2003). 이 트랜잭션 버퍼의 구조는 그림 3.1과 같다.

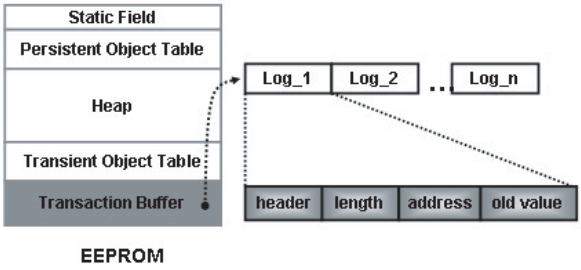

그림 3.1 트랜잭션 버퍼의 구조

트랜잭션 버퍼 (Marche와 Rousset, 2006)는 로그 항목 (Log Entry)들의 열로 구성된다. 그림 3.2는 로그의 구조를 나타낸다.

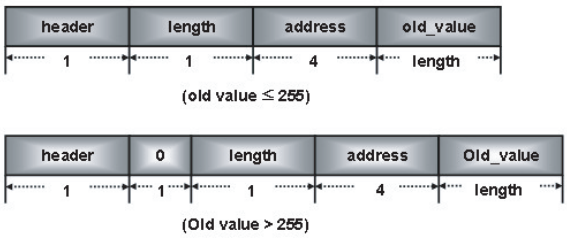

그림 3.2 로그 구조 및 종류

트랜잭션 버퍼를 구성하는 로그 엔트리는 그림 3.2와 같이 값 (Old_value)의 크기에 따라 두 종류로 분류된다. 여기서 현재값이 255 바이트 보다 작을 경우는 4개의 필드로 구성되는데, 트랜잭션 커밋의 상태를 나타내는 1 바이트 길이의 헤더 (header)필드, 이전에 기록되어있던 값의 길이를 나타내는 1 바이트의 길이 (length)필드, EEPROM에 저장된 위치를 나타내는 4 바이트 길이의 주소 (address)필드와 실제 데이터 값을 나타내는 값 필드로 구성된다.

본 연구에서는 EEPROM에 데이터를 기록하는 속도가 RAM에 데이터를 쓰는 속도에 비해 1만 배 이상 느리다는 점에 착안하여, EEPROM에 데이터를 쓰는 횟수를 줄이는 방법을 연구하였다. 이를 위하여 EEPROM을 5개의 세부 구역으로 분류하고, 각 구역별 쓰기 횟수를 정밀 분석한 결과 표 3.1에서

보는 바와 같이 EEPROM 의 전체 기록 횟수 중 70%이상이 트랜잭션 버퍼에서 발생한다는 사실을 발견하였다 (진민식, 2006b).

표 3.1 EEPROM의 각 영역별 기록 횟수

EEPROM Area	The number of EEPROM write
Static fields	1,681
Persistent Object Table	161
Heap	1,275
Transient Object Table	39
Transient Buffer	10,121
Total	13,227

따라서 자바카드의 성능을 개선하는 가장 좋은 방법의 하나는 EEPROM에 데이터를 쓰는 횟수를 줄이는 것이고, 이 방법 중에서 가장 효과적인 방법은 EEPROM 중에서 트랜잭션 버퍼에 쓰기 횟수를 줄이는 방법이라는 결론에 도달했다.

3.2. 기존의 자바카드 트랜잭션버퍼 처리 모델

자바카드에서는 프로그램의 설치나 실행 도중에 객체의 값을 비휘발성 메모리인 EEPROM에 중요한 데이터들을 저장하여 전원 공급이 끊어진 상황에 대비한다. 객체에 새로운 값을 저장하는 경우에 일어나는 기존의 자바카드 트랜잭션버퍼 처리 모델을 정상적인 상황 그림 3.3과 오류가 일어난 상황 그림 3.4로 나누어 볼 수 있다.

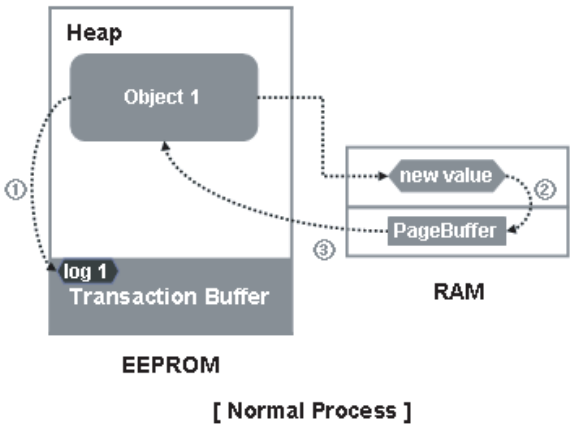

그림 3.3 데이터 수정에 따른 정상적 트랜잭션의 흐름

만일 EEPROM의 힙 구역에 있는 객체의 값을 변경시킬 필요가 있으면, 그림 3.3과 같이 아래의 세 과정의 절차를 따른다.

(과정1) 객체의 현재 값을 트랜잭션 버퍼에 미리 복사해둔다.

(과정2) 갱신하고자 하는 값을 128 바이트 크기의 페이지 크기에 해당하는 RAM 내부의 버퍼 (PageBuffer)에 저장해둔다.

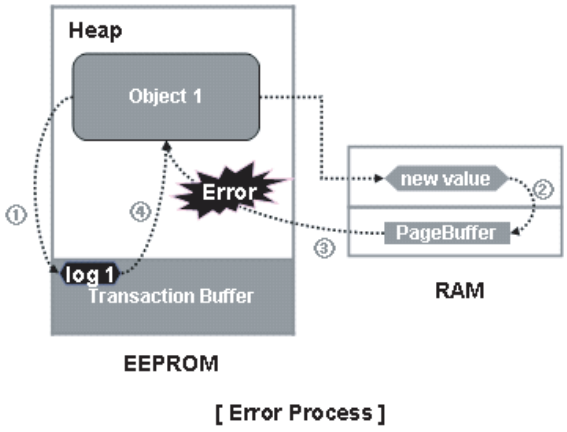

그림 3.4 데이터 수정에 따른 비정상적 트랜잭션의 흐름

(과정3) RAM 내부 버퍼의 내용이 차면 한꺼번에 EEPROM의 힙 구역에 저장한다.

만일, 데이터를 갱신하기 위해 트랜잭션 처리를 하고 있을 때 프로그램 상의 오류로 실행취소 메소드가 호출되면 트랜잭션 버퍼에 저장된 이전의 데이터로 바로 복구시키고, 전원 차단으로 인해 트랜잭션이 중지되었을 경우에는 전원이 재인가 될 때 트랜잭션 버퍼에 저장된 데이터를 이용하여 이전 값으로 복구시킨다. (그림 3.4) (Sun Microsystems, 2003d; Chen, 2000d). 그 과정은 아래의 네 가지 단계의 절차를 따른다.

(과정1) 객체의 현재 값을 트랜잭션 버퍼에 미리 복사해둔다.

(과정2) 갱신하고자 하는 값을 128 바이트 크기의 페이지 크기에 해당하는 RAM 내부의 버퍼에 저장해둔다.

(과정3) RAM 내부 버퍼의 내용이 차면 한꺼번에 EEPROM의 힙 구역에 저장하기 전에 에러가 발생하여, 객체 값이 갱신되지 않는다.

(과정4) 이 경우 트랜잭션 버퍼에 미리 저장해 둔 값을 이용하여 트랜잭션이 발생하기 이전의 상태로 복귀하고, 트랜잭션을 무효화 한다.

그러나 이러한 기존의 방법은 과정 1의 EEPROM의 트랜잭션 버퍼에 많은 쓰기 동작을 일으키고 있다. 표 3.1에 따르면 전체 EEPROM 쓰기 횟수 중에서 70% 가량이 과정1에 집중되고 있다. 따라서 본 연구에서는 과정1의 횟수를 줄일 수 있는 새로운 방법을 제시하였다.

그 아이디어는 과정1에서 하나의 로그 항에 포함되어 있는 헤더 필드, 길이 필드, 주소 필드, 값 필드 네 가지를 네 번에 걸쳐서 트랜잭션 버퍼에 저장하고 있음을 발견했다. EEPROM에서는 워드 단위가 아닌 페이지 블록 단위의 쓰기를 사용하므로, 128 바이트 크기의 페이지 단위의 쓰기 속도와 한 바이트를 기록하는데 거리는 시간은 같다.

따라서, 연속된 4가지 필드로 구성된 로그 항을 네 번에 걸친 기존의 기록 방법을 한 번에 모아서 처리하면 EEPROM 쓰기 횟수를 4분의 1로 줄일 수 있다. 또한 하나의 로그항의 길이가 2 바이트 정수객체인 경우, 총 8 바이트 (헤더 필드 1바이트 + 길이 필드 1바이트 + 주소 필드 4바이트 + 값 필드 2바이트)를 차지하므로, 128 바이트의 페이지 버퍼는 16개의 로그 항을 저장할 수 있다. 이 경우에는 16개 로그항목 당 4번의 기록이 요구되므로 총 64번의 EEPROM 쓰기 횟수를 1번에 해결할 수 있다.

3.3. 개선된 자바카드 트랜잭션 처리 모델

전술한 바와 같이 여러 개의 로그 항을 모아서 한 번에 트랜잭션 버퍼에 저장하기 위하여 기존의 PageBuffer를 객체를 위한 OPageBuffer로 하고, 트랜잭션의 로그 항들을 저장하는 TPageBuffer를 추가하였다. 객체에 새로운 값을 저장하는 경우에 일어나는 개선된 자바카드 트랜잭션버퍼 처리 모델을 정상적인 상황과 오류가 일어난 상황으로 나누어 보면 다음과 같다.

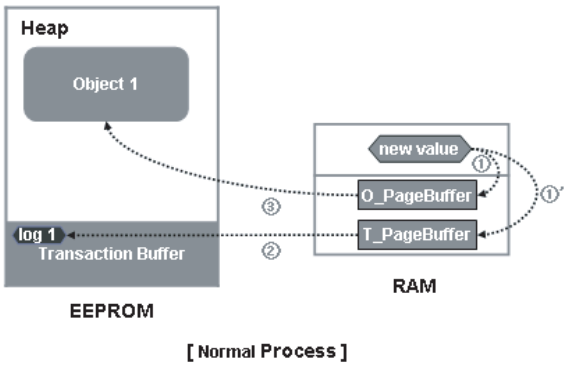

그림 3.5 데이터 수정에 따른 정상적 트랜잭션의 흐름

만일 EEPROM의 힙 구역에 있는 객체의 값을 변경시킬 필요가 있으면, 그림 3.5와 같이 아래의 세 과정의 절차를 따른다.

(과정1) 갱신하고자 하는 값을 128바이트 크기의 페이지 크기에 해당하는 RAM 내부의 버퍼인 OPageBuffer에 저장해둔다.

(과정1') 갱신하고자 하는 값을 헤더, 길이, 주소, 데이터의 로그 항에 맞도록 구성하여 RAM 내부의 TPageBuffer에 저장해둔다.

(과정2) TPageBuffer의 내용이 차면 한꺼번에 EEPROM의 트랜잭션 버퍼에 저장한다.

(과정3) OPageBuffer의 내용이 차면 한꺼번에 EEPROM의 힙 구역에 저장한다.

여기서 실제 과정2의 TPageBuffer는 하나가 아닌 여러 개의 TPageBuffer를 둘 수 있으며 그 개수는 하나의 OPageBuffer에 저장된 데이터들과 같은 수의 로그 항들을 저장할 수 있는 만큼이면 된다. 그리고 과정3의 OPageBuffer의 내용을 힙 구역에 저장하는 과정이 일어나기 직전까지 과정2를 늦출수록 트랜잭션 버퍼로의 쓰기 횟수를 줄일 수 있다. 본 논문에서는 하나의 TPageBuffer를 사용하여 설계 및 구현하였다.

만일, 데이터를 갱신하기 위해 트랜잭션 처리를 하고 있을 때 프로그램 상의 오류로 실행취소 메소드가 호출되면 트랜잭션 버퍼에 저장된 이전의 데이터로 바로 복구시키고, 전원 차단으로 인해 트랜잭션이 중지되었을 경우에는 전원이 재인가 될 때 트랜잭션 버퍼에 저장된 데이터를 이용하여 이전 값으로 복구시킨다. 에러 발생 시 처리 과정은 에러의 발생되는 구간에 따라 크게 두 부분으로 분류하는데 트랜잭션 버퍼에 저장 시 에러 그림 3.6과 힙 영역에 저장 시 에러 그림 3.7로 분류한다. 우선, 트랜잭션 버퍼에 저장 시 에러는 다음과 같은 단계의 절차를 따른다.

(과정1) 갱신하고자 하는 값을 128 바이트 크기의 페이지 크기에 해당하는 RAM 내부의 버퍼인 OPageBuffer에 저장해둔다.

(과정1') 갱신하고자 하는 값을 헤더, 길이, 주소, 데이터의 로그 항에 맞도록 구성하여 RAM 내부의

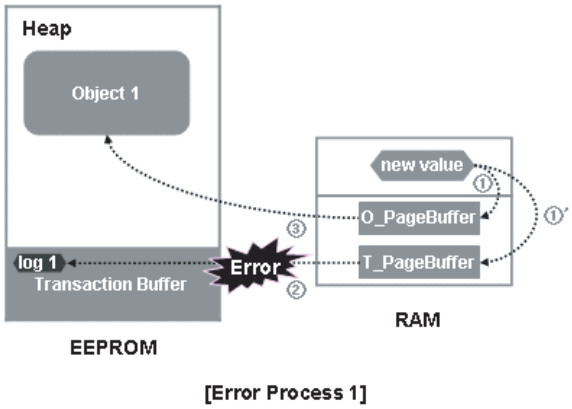

그림 **3.6** 데이터 수정에 따른 비정상적 트랜잭션의 흐름

TPageBuffer에 저장해둔다.

(과정2) TPageBuffer의 내용이 차서 EEPROM의 트랜잭션 버퍼에 저장하는 과정에서 오류가 발생한 경우에는 해당 작업을 무효화하고 과정1부터 새로 시작한다.

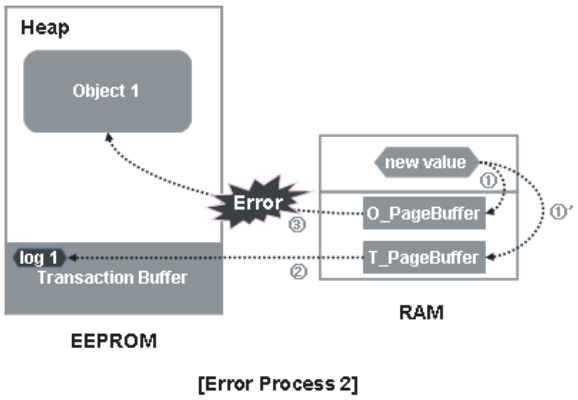

그림 **3.7** 데이터 수정에 따른 비정상적 트랜잭션의 흐름

다음으로 힙 영역에 저장 시 에러는 다음과 같은 단계의 절차를 따른다.

(과정1) 갱신하고자 하는 값을 128 바이트 크기의 페이지 크기에 해당하는 RAM 내부의 버퍼인 OPageBuffer에 저장해둔다.

(과정1') 갱신하고자 하는 값을 헤더, 길이, 주소, 데이터의 로그 항에 맞도록 구성하여 RAM 내부의 TPageBuffer에 저장해둔다.

(과정2) TPageBuffer의 내용이 차면 한꺼번에 EEPROM의 트랜잭션 버퍼에 저장한다.

(과정3) OPageBuffer의 내용이 차서 한꺼번에 EEPROM의 힙 구역에 저장하는 과정에 오류가 발생한다.

(과정4) 이 경우 트랜잭션 버퍼에 미리 저장해 둔 값을 이용하여 트랜잭션이 발생하기 이전의 상태로

복귀하고, 트랜잭션을 무효화 한다.

3.4. 개선된 자바카드 트랜잭션 처리 모델의 단점

본 논문에서 제안한 자바카드 트랜잭션 처리 모델은 크게 두 가지의 단점을 가지고 있다. 첫째는 스마트카드 칩 내에 TPageBuffer를 위한 별도의 RAM 공간이 필요하다는 것이다. RAM은 다른 메모리에 비해 반도체내 면적을 많이 요구하는 장치이다. 일반적으로 스마트카드의 RAM크기는 6 K바이트 정도이다.

표 3.2 스마트카드의 하드웨어 자원

Hardware	일반적 제품	신제품
CPU	8 비트	8, 16, 32 비트
ROM	24 K바이트	128 384 K바이트
EEPROM	16 K바이트	64 256 K바이트
RAM	1 K바이트	4 10 K바이트

따라서 자바스택, 암호처리용 C스택, 임시 객체 데이타 구역, 버퍼구역 등을 잘 정의해서 아껴쓰고 있는 RAM 구역에 TPageBuffer를 위한 추가의 RAM 사용이 필요하다.

둘째, 개선된 자바카드 트랜잭션 처리 모델에서 과정 1' 의 추가적 작업이 필요하다. 이 작업은 하나의 데이터에 대해 헤더, 길이, 주소를 더해서 하나의 로그 항으로 구성하는 작업이 된다.

4. 구현 및 성능평가

4.1. 실험환경

본 논문에서 구현된 개선된 자바카드 트랜잭션 처리 모델의 성능을 평가하기 위하여 자바카드 원천기술 보유사인 썬사의 자바카드와 본 논문에서 제안한 방법을 적용시킨 자바 카드를 비교하였다. 비교를 위한 테스트 데이터로는 Sun Micro Systems에서 제공하는 자바 카드 애플릿 샘플인 ChannelDemo, JavaLoyalty, JavaPurse, ObjDelete, PackageA, PackageB, PackageC, PhotoCard, RMIDemo의 9개 샘플을 이용하였다.

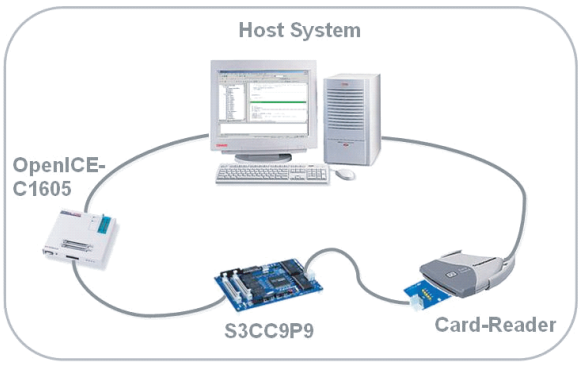

그림 4.1 실험 환경

이를 위한 실험환경은 그림 4.1과 같이 OpenICE-C1605 에뮬레이터 장비와 연결된 타겟 보드와 스마트 카드 리더기를 이용하여 실험하였다.

첫 번째 실험은 상기 9가지의 표준 예제 프로그램에 대하여 EEPROM내의 트랜잭션버퍼에 쓰기가 얼마나 집중되는지를 조사하였다. 이 조사의 목적은 자바카드의 성능개선을 위해 EEPROM 쓰기 횟수를 줄이는 것이 중요하고 (3장 1절참조), EEPROM 쓰기 횟수 중에서 트랜잭션 버퍼에 얼마나 많은 쓰기가 집중되는지를 조사하였다.

표 4.1 EEPROM 영역별 기록 횟수

Applet	메모리 영역						
	Static 영역		Heap 영역		Transaction Buffer		total
	횟수	백분율	횟수	백분율	횟수	백분율	횟수
Wallet	752	13%	1121	20%	3768	67%	5641
ChannelDemo	961	13%	1452	19%	5139	68%	7552
JavaLoyalty	1141	16%	1144	16%	5006	69%	7291
JavaPurse	2850	13%	3994	18%	15868	70%	22712
ObjDelete	2193	13%	2637	16%	11586	71%	16416
PackageA	1096	11%	1860	19%	6729	69%	9685
PackageB	1168	15%	1223	16%	5305	69%	7696
PackageC	519	15%	709	20%	2269	65%	3497
PhotoCard	905	13%	1283	19%	4549	68%	6737
RMIDemo	839	14%	1161	19%	4119	67%	6119
합계	12424	13%	16584	18%	64338	69%	93346
평균	1242	14%	1658	18%	6434	68%	9335

표 4.1에서 나타난 바와 같이 약 70%의 쓰기가 트랜잭션 버퍼에 집중되고 있었다. 이 표에서 Heap 영역의 쓰기 횟수는 그림 2.1의 persistent table, transient table 영역을 모두 포함한 값이다.

이 실험 결과가 본 논문의 동기가 되어, 3장에서 설명한 개선된 트랜잭션 버퍼 처리방법을 제안하였고, 기존의 트랜잭션 버퍼 운용과 새로운 운용방법에 따른 EEPROM 쓰기 횟수를 비교한 결과 표 4.2와 같은 실험 결과를 얻었다.

표 4.2 개선된 알고리즘을 적용한 EEPROM 쓰기 횟수 비교표

Applet	기존 Transactionbuffer 운용	개선 Transactionbuffer 운용	개선율
Wallet	3768	2476	24%
ChannelDemo	5139	3385	23%
JavaLoyalty	5006	3296	24%
JavaPurse	15868	10501	24%
PackageA	6729	4455	23%
PackageB	5305	3497	24%
PackageC	2269	1485	23%
PhotoCard	4549	2991	24%
RMIDemo	4119	2706	23%

표 4.2에 따르면 본 논문에서 제안한 방법은 기존의 방법에 비해 약 23% 정도의 EEPROM 쓰기를 줄일 수 있었다. 이 결과를 도표로 나타내면 그림 4.2와 같다.

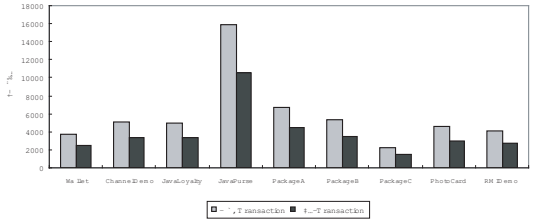

그림 4.2 개선된 알고리즘을 적용한 EEPROM 쓰기 횟수 비교 그림

5. 결론

본 논문에서는 스마트카드용 자바가상기계의 성능을 높이기 위하여 카드 동작과정에서 병목현상을 일으키고 있는 트랜잭션 버퍼의 기록 시간을 줄일 수 있는 방법으로서 보다 효율적인 트랜잭션 버퍼운용방법을 제시하였고, 실험을 통해서 기존의 연구와 비교하여 그 성능의 우수성을 입증하였다.

스마트카드는 현재 전 세계적으로 50억 개가 발매되고 있으며, 스마트카드 제조에는 메모리 비메모리 통합 반도체 기술, 그리고 초소형 운영체제 기술들이 요구되어지며, 우리나라의 삼성전자가 세계 1위의 스마트카드 칩 제조업체이다. 현재 GSM 휴대폰이나 IMT2000, 3세대 휴대폰에 모두 이 칩이 채택되었으며, 따라서 본 연구의 결과는 우리나라의 산업발전에도 기여할 수 있는 기술이다.

참고문헌

알앤디비즈 (2005). <스마트카드 시장분석 및 전망>, 알앤디비즈 기술 보고서.

알앤디비즈 (2006). <스마트 카드 (Smart Card) 시장동향 리포트>, 알앤디비즈 기술 보고서.

진민식 (2006). An enhanced java card system for fast execution of applet. <박사논문>, 경남대학교 대학원.

최원호 (2005). 자바 카드 프로그램 적재 및 실행 속도 개선에 관한 연구. <박사 논문 >, 경남대학교 대학원.

탁승호 (2004). <Let's Smart Card>, 성안당.

한진희 (2003). 스마트카드 플랫폼 기술. <TTA저널>, **90**, 75-82.

Baentsch, M., Buhler, P., Eirich, T., Horing, F. and Oestreicher, M. (1999). Java card-from hype to reality. *IEEE Concurrency*, 36-43.

Beckert, B. and Mostowski, W. (2003). A program logic for handling java card's transaction mechanism, In Pezze, M., ed.: Fundamental approaches to software engineering. *FASE' 2003*, **2621**, 246-260

Chen, Z. (2000). *Java card technology for smart cards*, Addison-Wesley.

Gal, E. and Toledo, S. (2005), Algorithms and data structure for flash memories. *ACM Computing Surveys*, **37**, 138-163.

Hansmann, U., Nicklous, M. S., Schack, T. and Seliger, F. (2002). *Smart Card Application Development Using Java*, Springer.

Hähnle, R. and Mostowski, W. (2005). Verification of safety properties in the presence of transactions. *CASSIS 2005*, **3362**, 151-171.

Huang, A. J. R. (2007). Dynamic flash-memory allocation for smartcards: how to cope with limited space (in a short life). *2007 5th IEEE International Conference*, **2**, 835-840.

Hwang, S. Y. and Hahm, J. H. (2003), MDPREF and perceptual map via INDSCAL method. *Journal of the Korean Data & Information Science Society*, **14**, 501-510.

Jin, M. and Jung, M. (2005). A study on how to reduce time and space by redefining new bytecode for java card. *RTCSA 2005*, 551-554.

John Wiley & Sons (2003). *Wolfgang Rankl and Wolfgang Effing*, Smart Card Handbook.

Marche, C. and Rousset, N. (2006). Verification of JAVA CARD applets behavior with respect to transactions and card tears. *SEFM'06*, 137-146.

Oestreicher, M. (1999). Transactions in java card. *In 15th Annual Computer Security Applications Conferrence*, 291-298.

Qabs, U. M. and Al-Naima, F. M. (2008). Design and implementation of a smart card simulator. *Computer and Communication Engineering*, **ICCCE 2008**, 217-220.

Ryu, K. H. and Park, H. C. (2006), Web-based DNA microarray data analysis tool. *Journal of the Korean Data & Information Science Society*, **17**, 1161-1167.

Sharp (2005). Addressing security concerns of flash memory in smart cards. *Application Note SMA04036*.

Shasha, Nir, Toledo and Sivan (2007). Storing a persistent transactional object heap on flash memory. *Software-Science, Technology & Engineering*, **SwSTE 2007**, 66-76.

Spivak, M. and Toledo, S. (2006). Storing a persistent transactional object heap on flash memory. *ACM SIGPLAN Notices*, 22-33.

Sun Microsystems (2003a). *The Java CardTM2.2.1 Virtual Machine Specification*, SUN.

Sun Microsystems (2003b). *The Java CardTM2.2.1 Runtime Enviroment (JCRE) Specification*, SUN.

Venners, B. (1997). *Inside the Java Virtual Machine*, McGraw-Hill.

Yang, Y., Choi, W., Jin, M., Hwang, C. and Jung, M. (2006). An advanced java card system architecture for smart card based on large RAM memory. *Hybrid Information Technology 2006*, **2**, 646-650.

Journal of the Korean Data & Information Science Society
2009, **20**(1), 125–137

한국데이터정보과학회지

A study on high performance Java virtual machine for smart card[†]

Min-Soo Jung[1]

Kyungnam University

Received 28 December 2008, revised 13 January 2009, accepted 17 January 2009

Abstract

Smart card has a small sized micro computer chip. This chip contains processor, RAM, ROM, clock, bus system and crypto-co-processor. Hence it is more expensive, complicated and secure chip compared with RFID tag. The main application area of smart card is e-banking and secure communications. There are two kinds of smart card platforms; open platform and closed one. Java card is the most popular open platform because of its security, platform independency, fast developing cycle. However, the speed of Java card is slower than other ones, hence there have been hot research topics to improve the performance of Java card. In this paper, we propose an efficient transaction buffer management to improve the performance of Java card. The experimental result shows the advantage of our method.

Keywords: Java card, smart card, transaction buffer.

[†] This paper was supported by Research Year Fund of Kyungnam University in 2007.
[1] Professor, Division of Computer Engineering, Kyungnam University, Masan, Kyungnam 631-701, Korea. E-mail: msjung@kyungnamu.ac.kr