Journal of Korea Multimedia Society Vol. 10, No. 9, September 2007(pp. 1197-1208)

선반입 LRU-OBL 버퍼 기법을 적용한 자바 카드 프로그램 적재 및 실행 속도 개선에 관한 연구

오세원[†], 최원호[††], 정민수[†††]

요 약

오늘날 대부분의 스마트카드는 자바카드 플랫폼을 채택한 자바카드가 표준안으로 자리매김 하고 있다. 자바카드 기술은 이식성, 플랫폼 독립성, 높은 보안성 기능을 스마트카드에 제공한다. 그러나 자바카드는 일반 스마트카드에 비해 자바 언어 상의 특성으로 인해 실행속도 저하의 단점을 갖고 있다. 실행속도에 영향을 끼치는 요소는 자바 카드가 데이터를 저장하는 방식과 자바카드 설치기가 애플릿을 설치하는 방식 때문이다. 본 논문에서는 자바카드 프로그램의 적재와 실행의 속도를 개선시키기 위한 방안으로 자바카드에서 EEPROM에 데이터에 대한 저장, 갱신, 삭제하는 과정들을 처리 속도가 빠른 RAM을 이용하여 자바카드의 속도를 개선할 수 있다. 이를 위해 자바카드에서 RAM을 활용한 방법으로 자바카드 환경에 맞는 선반입 LRU-OBL 버퍼 캐쉬 기법을 제시한다. 자바카드에서 생성되는 모든 데이터들을 데이터 특성에 따라 버퍼 캐쉬에서 관리함으로써 EEPROM에 대한 기록 횟수를 최대한 줄여 자바카드의 프로그램 적재 및 실행속도를 향상시킨다.

A Study of the Improvement of Execution Speed and Loading of Java Card Program by applying prefetching LRU-OBL Buffer Techique

Se-Won Oh[†], Won-Ho Choi[††], Min-Soo Jung[†††]

ABSTRACT

These days, most of SMART card, JAVA card, picked up the JAVA Card Platform gets the position as a standard. Java Card technology provides implantation, platform portability and high security function to SMART Card. Compared to normal Smart Card, JAVA card has a defect that is a low running speed caused by a distinctive feature of JAVA programming language. Factors that affect JAVA Card execution speed are the method how to save the data and install the applets of JAVA Card installation instrument. In this paper, I will offer the plan to improve JAVA Card program's loading and execution speed. At Java Card program, writing, updating and deleting process for data at EEPROM can be improved of Java Card speed by using high speed RAM. For this, at JAVA Card as a application of RAM, I will present prefetching LRU-OBL Buffer Cache Technique that is suitable for Java Card environment. As a data character, managing all data created from JAVA Card at Buffer Cache, decrease times of recording at maximum for EEPROM so that JAVA Card program upload and execution speed will be improved.

Key words: Java(자바), Java Card(자바 카드), Buffer Cache(버퍼 캐쉬)

1. 서 론

오늘날 스마트카드는 통신, 금융, 교통, 신분 확인, 전자 화폐 등의 여러 응용 서비스에서 널리 사용되고 있으며 점차로 그 용도가 확대되고 있다. 예를 들어, 금융 측면에서는 기존의 신용 카드를 대체하고 다양

한 형태의 장비를 통하여 네트워크와 연결되어 사용되는 추세로 바뀌고 있다.

자바카드 플랫폼을 내장한 스마트카드인 자바카드는 현재 스마트카드에 적용되는 모든 표준을 따르는 전형적인 카드이다. 자바카드는 하위의 운영체제 위에 존재하는 자바카드 가상기계(Java Card Virtual Machine : JCVM)가 자바카드 애플릿의 바이트코드(bytecode)를 수행하고, 메모리, I/O 등과 같은 스마트카드 내의 모든 자원에 대한 접근을 제어한다는 점에서 기존 스마트카드 플랫폼과 차이가 난다. 자바카드 기술은 플랫폼 간에 이진 코드의 이식성(portability) 즉, 상호 운용성(inter-operability)이 뛰어나고 형(type) 검사 등에 의해 악의적 코드에 대한 보안성을 지닌 자바 언어를 스마트카드의 실시간 환경에 대해서 최적화하고 있다. 스마트카드에서 자바카드 가상기계 이용으로 인한 장점은 응용프로그램과 운영체제를 분리하는 개방형(open-platform) 운영체제를 갖는 것이다. 이것은 하드웨어 의존적인 어셈블리 코드가 아닌 상위언어인 자바 언어로 쉽게 응용프로그램을 작성 및 수행할 수 있게 하고, 카드가 최종사용자에게 발급된 이후에도 필요한 응용서비스에 따른 응용프로그램을 자바카드에 적재(post-issuance)도 가능하게 한다. 이는 다수의 다양한 응용 프로그램을 수용할 수 있는 유연성(flexibility)을 가지게 한다. 또한 자바카드에서는 자바 언어 자체의 보안 특성 이외에 응용프로그램간의 방화벽을 제공함으로써 엄격한 보안성을 보장한다. 이러한 안정성 때문에 스마트카드 응용 서비스 시장에서 점진적인 점유율 증가를 나타내고 있다[1-3].

본 논문의 구성은 다음과 같다. 2장에서는 자바카드, 자바카드 가상기계 등 자바 카드 구성에 관련된 연구들을 기술하고, 3장에서는 자바카드 속도 개선을 위한 방안 설계 관하여 기술한다. 그리고 4장에서는 개선 알고리즘의 구현 결과에 대해서 기술한다. 마지막으로 5장에서는 결론에 대해 기술한다.

※ 교신저자(Corresponding Author) : 오세원, 주소 : 경남 마산시 월영동 449번지(630-011), 전화 : 055)249-2217, FAX : 055)248-2554, E-mail : osw530@kyungnam.ac.kr
접수일 : 2007년 3월 6일, 완료일 : 2007년 8월 9일
† 준회원, 경남대학교 컴퓨터공학과
†† 준회원, 경남대학교 컴퓨터공학과

2. 관련 연구

2.1 자바카드

자바카드 기술은 자바 언어로 작성된 프로그램이 스마트카드나 혹은 그 밖에 제한적인 자원을 가진 장치에서 동작을 가능하게 한다. 자바카드 기술의 설계는 스마트카드에 자바 시스템 소프트웨어를 구축하는 것이다. 그리고 해법은 자바 언어의 특징을 부분적으로 지원하고, 분할모델을 적용할 수 있는 자바 가상기계(Java Virtual Machine)를 구현하는 것이다[4-8].

스마트카드에서 사용되는 자바카드는 자바 언어의 장점들인 플랫폼 독립적으로 실행되고, 실행 프로그램에 따라 다목적으로 이용할 수 있고 자바카드 애플릿을 동적으로 다운로드하여 카드에 설치할 수 있다. 이와 같은 자바카드의 내부구조는 그림 1과 같다.

칩 운영체제와 자바카드 가상기계(Java Card Virtual Machine : JCVM)의 일부 코드들은 ROM 메모리에 적재되고, 자바카드 가상기계의 나머지 코드들과 자바카드 애플릿 실행에 필요한 기본적인 API들과 카드로 다운로드 된 애플릿, 패키지 등이 EEPROM 메모리에 저장된다. 그리고 RAM 메모리에는 자바카드 가상기계의 실행 시에 사용되는 각종 메소드들의 호출과 매개변수, 그리고 현재 수행중인 애플릿에 관한 정보들이 저장된다[4-5,9].

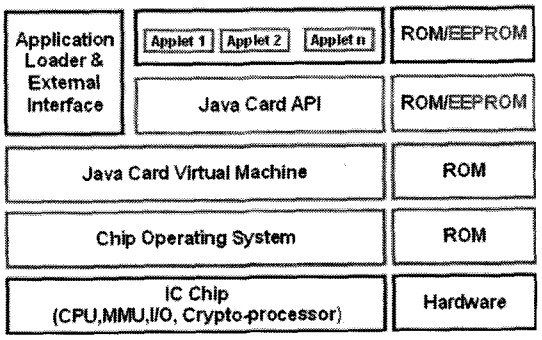


그림 1. 자바 카드 내부 구조

(E-mail :hoya9499@kyungnam.ac.kr)
††† 종신회원, 경남대학교 컴퓨터공학과
(E-mail : msjung@kyungnam.ac.kr)
※ 본 연구는 경남지능형 홈 고급인력 양성 사업팀의 연구지원으로 수행되었음.

2.2 자바카드 가상기계

자바카드 바이트코드를 수행하는 자바카드 가상기계(JCVM)는 자바 가상기계에 비해서 스마트카드에 적합하게 최적화되어 있다. 그림 2와 같은 자바카드 가상기계가 기존의 가상기계와 가장 큰 차이점은 자바카드 가상기계가 오프-카드(Off-Card) 가상기계와 온-카드(On-Card) 가상기계로 나뉘는 분할 가상기계로 이루어진다는 것이다. 기존의 가상기계에서 실행시점에 처리되는 부분 중에서 클래스 적재(class loading), 바이트코드 검증(bytecode verification), 클래스 링킹(linking)과 해석(resolution) 부분은 자원에 제약을 받지 않는 오프-카드에서 처리된다[4,5].

자바카드 가상기계의 좁은 의미는 수행 엔진인 인터프리터를 지칭하며 넓은 의미로는 시스템 클래스 API와 인터프리터, 메모리 관리, 예외처리 및 운영체제와 인터페이스 등을 포함하는 자바카드 수행환경(Java Card Runtime Environment : JCRE)을 의미한다.

변환기는(converter)는 번역을 통해 생성된 클래스 파일을 온-카드 가상기계 상에서 수행 가능한 이진 파일 형식인 CAP 파일과 링크 및 검증을 위한 인터페이스 이진파일 형식인 EXP 파일을 생성하는 오프-카드 가상기계의 한 부분이다. 자바 가상기계는 한 번에 하나의 클래스를 처리하지만 변환기의 변한 단위는 패키지이다. 변환기는 자바카드 가상기계가 수행하는 작업 중에서 클래스 적재 시 수행하는 작업을 담당한다.

자바카드 인터프리터는 바이트코드 명령을 수행함으로써 궁극적으로 애플릿을 실행하고, 메모리 할당을 관리하며 객체를 생성하는 역할을 한다.

2.2.1 애플릿 수행과정

애플릿을 정의하고 있는 패키지가 스마트카드 상에 적재되어 있고, 카드 상의 다른 패키지와 링크된 후, 애플릿의 인스턴스 객체가 자바카드 수행환경에 생성되고 등록되었을 때 애플릿의 수명이 시작된다.

그림 3에 나타낸 바와 같이 애플릿이 설치되었을 경우 최초에는 비활성(inative) 상태이다. 애플릿은 호스트 응용 프로그램에 의해 선택되었을 경우 활성(active) 상태가 된다. 애플릿이 선택되면 일단 호스트 응용 프로그램으로부터 명령을 기다린다. 따라서, 새로운 애플릿이 선택될 때까지 명령과 응답(command-and-response)이 계속된다[1,2,8].

애플릿과 호스트 응용 프로그램은 그림 4과 같이 APDU 패킷을 교환함으로써 통신한다. APDU는 명령 메시지 혹은 응답 메시지를 포함한다. 호스트 응용 프로그램은 명령 메시지를 애플릿에 보내고 애플릿은 필요한 작업을 수행한 후에 응답 메시지를 되돌려 준다. 호스트 응용 프로그램이 실행시킬 애플릿을 선택하려고 한다면, Select 명령과 애플릿의 AID를 포함한 APDU를 보낸다. 자바카드 수행환경은 APDU 패킷을 가로채서 자신의 애플릿에 대한 정보를 가지는 내부 테이블을 검색해서 AID가 일치되는 애플릿을 찾아서 선택한다[1,2].

이후에 전송되는 모든 APDU 패킷은 새로운 애플릿이 선택될 때까지 현재 애플릿에 포워드(forward)된다.

2.3 버퍼 캐쉬 관리 정책

지금까지 연구되어온 버퍼 캐쉬의 선반입 정책들은 선반입할 블록을 결정하는 방법에 따라 미래의 참조 정보를 기반으로 하는 정책과 과거의 참조 정보를

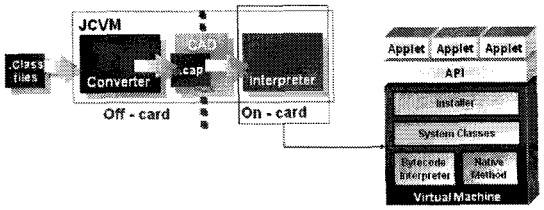


그림 2. 자바카드 가상기계 구조

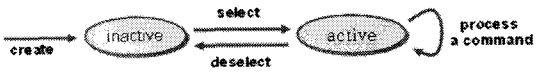


그림 3. 애플릿 실행과정

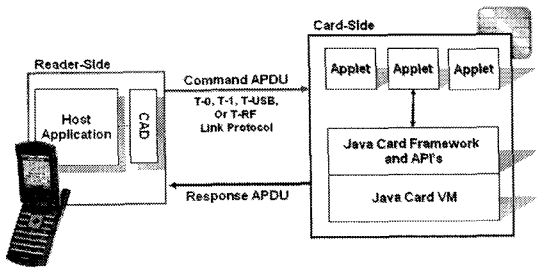


그림 4. APDU을 이용한 애플릿과 통신

바탕으로 하는 정책, 그리고 외부로부터의 어떠한 미래 참조에 관한 힌트도 없고 과거의 참조 정보도 유지하지 않는 정책 등으로 분류 되어질 수 있다[10,12].

Cao et al.[13,14]과 Patterson et al.[15] 등은 미래의 참조 조사 정보를 기반으로 하는 선 반입 정책을 제안하였다. 이들 정책들에서는 프로그래머가 파일 시스템에 응용 프로그램의 미래 참조에 관한 정보를 제공하면 파일 시스템이 이를 바탕으로 선 반입을 한다.

이들 정책들은 참조될 블록을 예측하는 것이 아니라 실제로 참조될 블록을 선반입하기 때문에 매우 뛰어난 성능을 보이게 된다. 특히 특정 응용 프로그램들에 대해서 이러한 힌트 제공이 가능한 것으로 판단됨에 따라 이러한 접근은 매우 가능성이 있는 것으로 보인다.

그러나 응용 프로그램으로부터 힌트를 받아 처리할 수 있도록 기존의 운영체제에 응용 프로그램과 파일 시스템 간의 인터페이스를 첨가하여 수정해야 한다. 또한 파일 시스템에 미래 참조에 대한 힌트를 제공하는 응용 프로그램을 작성해야 하는 프로그래머에게는 큰 부담을 주게 된다. 이와 관련된 연구로서 프로그래머의 부담을 없애는 방식으로 컴파일러가 운영체제에 참조 정보를 제공하는 방법이 가상 메모리 영역에서 제안된 바가 있다. 그러나 이러한 접근 방법은 컴파일러의 특성상 정적인 정보에 제한되는 한계를 가지고 있으며, 아직 일반적인 시스템 환경에서는 검증된 바가 없다[11,12,16].

이와 반대로 과거의 참조 정보를 바탕으로 미래 참조를 예측하여 선반입하는 정책들[13-15]은 과거 일정 기간의 참조 형태를 분석한 후 이를 바탕으로 미래에 참조될 것으로 예상되는 블록들을 선반입하는 것이다. 이러한 정책에서는 예측 정확도를 높이기 위하여 많은 양의 과거 참조 정보를 유지해야 하므로 많은 기억 장소와 처리 시간을 요구하게 되어 이것이 오히려 성능 저하의 요인이 될 수 있다는 단점이 있다[11,12,16].

위의 두 가지 정책들과 달리 Smith가 제안한 LRU-OBL(Least Recently Used - One Block Lookahead)[17] 정책은 외부로부터의 어떠한 미래 참조에 관한 힌트가 없고 과거의 참조 정보도 유지하지 않은 정책이다. LRU-OBL 정책은 참조된 블록과 함께 단순히 참조된 블록의 논리적 다음 블록이 버퍼 캐쉬에 없다면 선반입한다. LRU-OBL 정책은 일괄 파일들과 임시 파일들에 대한 캐쉬 적중률을 80%정도 증가시키는 성능 향상을 나타낸다[17]. 이러한 성능 향상은 선반입되는 블록의 가치를 더 높음을 암시 하고 있다. 이러한 특징으로 인해 LRU-OBL 정책은 알고리즘의 단순함에 비해 매우 좋은 성능을 보여 현실적으로 실용화 되어있는 유일한 선반입 정책이다.

따라서 이 정책의 장점인 알고리즘으로 구현하기 쉽고 성능이 좋다는 점과 선반입을 위하여 시스템으로부터 어떠한 추가적인 오버헤드를 요구하지 않는다는 점을 기인하여 본 논문에서는 보다 효율적이고 적중률이 높은 LRU-OBL 버퍼 캐쉬 관리 정책을 자바카드 플랫폼의 버퍼 캐쉬에 적용을 시켜 데이터의 입출력 비중을 줄임으로써 자바 카드의 속도를 개선시키고자 한다.

3. 자바카드 속도 개선을 위한 방안 설계

3.1 자바 카드가 사용하는 메모리

자바 카드에서 사용되는 IC 칩에는 세 가지 형태의 메모리, 즉 RAM, ROM, EEPROM가 이용된다. 메모리들은 각각의 특성에 따라 기록하는 데이터들의 종류가 다르다. 그림 5는 자바카드의 IC 칩에서 사용되는 메모리들의 활용 형태이다[1-3,4,9].

먼저 ROM은 비휘발성 저장 매체로서 칩 운영체제 및 자바카드 API들과 자바카드 가상기계에 관련된 코드와 기본적인 애플릿에 관한 바이트코드를 저장한다.

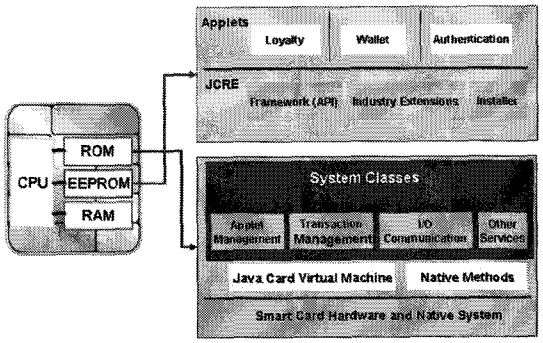


그림 5. 자바카드에서 메모리 사용 형태

두 번째 RAM은 자바에서 메소드 호출관계와 지역 변수들을 관리하는 자바 스택과 임시(transient) 객체, 다중 어플리케이션을 지원하는 논리 채널에 관련된 데이터들을 저장한다. 자바카드에서 사용되는 객체는 영속 객체와 임시 객체 두 가지 형태를 사용한다. 영속 객체는 자바카드 사용자의 계좌 번호, 잔액 등과 같이 애플릿이 설치되기부터 카드에서 완전히 제거될 때까지나 자바카드가 폐기될 때까지 애플릿에서 사용되는 데이터들을 저장하는 객체이고 또한 애플릿 수행에 필요한 기본적인 입출력, 통신, 및 예외 처리에 관련된 객체들이다. 이와 반면에, 임시 객체는 사용자가 카드 사용 시에 필요에 따라 인증기관으로부터 카드에 대한 사용을 승인 받고자 할 때 이용되는 보안에 관련된 객체들이다. 이들 객체는 필요시에만 간단하게 생성이 되고 애플릿 수행이나 카드 사용이 끝나면 메모리에서 제거되는 객체들을 말한다.

마지막으로 EEPROM은 자바카드에서 하드 디스크와 같은 기능을 한다. 자바 카드 내부에서 영구히 보존해야 할 데이터들인 영속 객체들과 카드로 다운로드 된 애플릿들을 모두 EEPROM에서 저장할 뿐만 아니라 자바카드의 환경 설정에 관련된 내용들을 모두 저장한다. 또한 임시 객체에 대한 정보도 EEPROM에서 관리를 한다.

3.1.1 RAM과 EEPROM의 속도

일반적인 스마트카드에서는 RAM과 EEPROM을 이용해서 데이터를 저장한다. 또한 지속적인 데이터들을 관리하기 위해서 EEPROM을 사용한다. 자바카드에서도 생성되는 가장 기본적인 데이터인 객체, 전역 변수, 지역 변수 및 카드 환경 설정에 관련된 모든 데이터들뿐만 아니라 각 데이터에 대한 갱신도 EEPROM에 저장된다.

자바 카드가 EEPROM을 사용한다는 것은 데이터를 지속적으로 사용할 목적이지만, 이런 방식 때문에 자바카드에서 애플릿의 설치 및 수행속도가 저하된다. 자바카드에서 실행속도가 저하는 되는 이유는 RAM에 비해 쓰기(write)와 삭제(erase) 작업의 속도가 10^4배 정도 느린 특성을 갖는 EEPROM에 데이터들을 저장하기 때문이다. 그리고 카드에서 객체가 생성될 때 완전한 데이터 값들을 한꺼번에 EEPROM에 저장을 하는 것이 아니라 변할 때마다 계속적으로 EEPROM의 내용을 갱신하기 때문에 수행 속도가 느려진다.

자바 카드에서 객체를 EEPROM이 아니라 기록속도가 10^4배 정도 빠른 RAM을 이용해서 생성한다면 자바 카드의 수행 속도를 보다 빠르게 향상시킬 수 있다. 하지만 모든 객체들을 RAM에서 생성하여 사용할 경우에는 자바 카드 운용에 많은 문제점들을 야기 시킬 수 있다. 예를 들어, 사용자가 카드를 사용하여 결제를 한다면 자바카드에 설치되어 있는 애플릿이 수행을 시작하게 된다. 일반적인 자바 카드 애플릿들은 대부분 객체들을 통해서만 실행이 된다. 이때 애플릿에서 만들어질 객체를 RAM에 생성을 한다면 기록 속도가 EEPROM보다 빠르기 때문에 카드 실행 시간을 단축시킬 수 있다. 하지만, 사용자나 카드 판독기가 작동 상의 오류로 카드 결제 작업 중에 카드를 카드 판독기로부터 제거를 시키거나 혹은 카드 판독기에서 오류가 발생한다면 애플릿이 실행되는 동안 생성한 모든 객체들은 RAM 메모리로부터 소멸되고 사용자가 요청한 작업은 더 이상 진행을 할 수 없게 된다. 따라서 중요한 카드의 처리 단계들이 종료될 때마다 RAM 상의 데이터들을 EEPROM로 옮겨놓음으로 인해 항상 마지막으로 처리된 결과를 유지할 수 있게 한다.

자바 카드 내부에서 가장 많이 사용되는 객체를 RAM에 생성한다는 것은 카드의 수행 속도 측면에서 획기적인 향상을 가져올 수 있지만 카드가 지녀야 할 기본적인 기능인 중요한 개인 정보 저장 기능이 상실된다고 할 수 있다. 따라서 객체를 생성시킬 때 정책을 수립하여 EEPROM과 RAM을 효과적으로 활용하는 것이 자바 카드의 수행 속도를 향상시킬 수 있다는 것이다.

3.1.2 기존 버퍼링 방식

스마트카드에서도 수행 속도를 향상시키기 위해서 다양한 방법을 활용하고 있으면 그 중에 대표적인 방식이 EEPROM의 특성을 고려한 데이터 버퍼링 방식이다. 또한 자바 카드에서도 가상 기계가 수행하는 데이터를 EEPROM에 기록하는 것에 대부분의 수행 시간들을 낭비하는 것을 막고자 버퍼링 기법을 사용한다.

대부분의 EEPROM을 저장 매체로서 이용하는 소형 디지털 장치에서는 데이터를 기록할 때 페이지

(page)라는 단위를 사용하고 있다. 이러한 페이지 단위는 데이터를 메모리에 저장할 때 최대 128byte정도를 한 번에 기록할 수 있다. 자바 카드도 역시 데이터를 저장하기 위해서 EEPROM의 페이지 단위를 이용한 버퍼링 방법을 사용한다.

자바 카드의 기존 버퍼링 기법은 가상 기계가 애플릿 실행 중에 필요한 객체들과 기타 데이터 값들 생성할 때, 즉시 EEPROM에 값을 기록하는 것이 아니라 속도가 빠른 RAM 영역 내부에 위치한 페이지 단위 크기만큼의 공간이 할당된 버퍼에 먼저 저장을 한다. 그런 다음 128byte 크기의 버퍼에 저장된 데이터를 EEPROM으로 옮겨 기록 작업을 마친다.

그림 6은 자바카드에서 사용되고 있는 버퍼링 기법을 나타낸 것이다.

자바카드의 수행 속도를 향상시키기 위해서 사용되고 있는 기존 버퍼링 기법의 동작 과정은 비효율적으로 운용되고 있다. 기존 버퍼링 기법의 목적은 가상기계에서 발생한 데이터를 페이지 크기인 128byte만큼 채워 가능한 EEPROM의 기록 횟수를 줄이는 것이다.

하지만 기존 버퍼링 기법은 EEPROM에 기록할 한 byte의 데이터가 있으면 먼저 버퍼에 저장한 다음, 버퍼에 기록된 데이터의 크기가 페이지 단위인 128byte만큼 채워지지 않더라도 EEPROM에 기록을 한다. 이와 같은 버퍼링 기법은 데이터가 발생할 때마다 그 즉시 EEPROM에 쓰기 작업을 하게 되므로 자바 카드의 성능 향상에는 아무런 도움이 되지 않는다. 이에 자바카드의 실행 속도 향상을 위해서는 보다 나은 버퍼링 기법이 필요하여 본 논문에서는 EEPROM에 기록되는 데이터의 특성을 고려하고 RAM 내부의 버퍼를 보다 더 효과적으로 활용할 수 있는 선반입 LRU-OBL 방식을 이용한 버퍼 캐쉬 기법을 제안한다.

3.2 속도개선을 위한 방안 설계

자바 카드 프로그램의 적재와 실행 속도에 가장 많은 영향을 끼치는 것은 중요한 모든 데이터를 보존하기 위해서 RAM보다 속도가 10^4배정도 느린 EEPROM을 사용을 한다는 것이다.

그림 6. 자바 카드의 기존 버퍼링 방식

따라서 자바 카드 속도를 개선을 위해 제안하는 주된 내용은 자바 카드에서 생성되는 데이터 등과 APDU를 통해서 전달되는 애플릿을 처리 속도가 느린 EEPROM에 바로 기록을 하는 것이 아니라 속도가 빠른 RAM을 활용함으로 EEPROM에 대한 접근을 최대한 줄이고자 한다.

본 논문에서는 자바 카드 성능 개선을 위해서 EEPROM 보다 속도가 빠른 RAM을 활용하는 방안으로 설계한다.

3.2.1 버퍼 캐쉬 기법 설계

기존 자바 카드 플랫폼은 EEPROM의 페이지 특성을 이용한 128byte 크기의 버퍼를 활용하였다. 하지만 이 버퍼는 EEPROM에 기록이 되는 데이터의 집중률(locality)을 전혀 고려하지 않고 사용하고 있는 형태였다. 또한 EEPROM에 기록할 1byte의 데이터가 발생하더라도 그 때마다 바로 EEPROM에 기록하여 자바 카드의 실행속도에 영향을 미친다.

본 논문에서는 자바 카드의 속도 향상을 위해서 EEPROM에 기록될 데이터 집중률을 고려해서 버퍼를 두 가지 형태로 구분하여 사용하는 방법을 제안한다. 두 가지 형태의 버퍼는 데이터 집중률이 떨어지는 데이터들만 기록하는 테이블 버퍼(table buffer)와 데이터 집중률이 높은 데이터만을 기록하는 오브젝트 버퍼이다.

두 가지 형태의 버퍼를 설계하는 것은 자바 카드에서 생성되고 EEPROM에 저장되는 데이터들이 특성을 고려한 것이다. 자바 카드가 생성하는 데이터들 중에서 집중률이 가장 높은 데이터의 종류는 객체에 관한 데이터이다. EEPROM에 저장이 되는 객체 데이터들은 연속적으로 EEPROM의 힙 영역에 배치가 된다. 그림 7은 EEPROM에 저장된 데이터들 내용을 나타내고 있다.

그림 7에서 보는 바와 같이 객체 데이터들이 연속적인 형태로 나타나는 것을 확인할 수 있다. 또한 본 논문에서 제안하는 버퍼 캐쉬는 기존 자바 카드에서 사용한 버퍼를 집중률이 떨어지는 데이터인 객체 위치 정보와 전역 변수 등과 같은 값을 저장하는 테이블 버퍼로서 활용하고 객체와 같은 집중률이 높은 데이터를 저장하기 위해서 크기가 256Byte의 버퍼를 그림 8과 같이 사용한다.

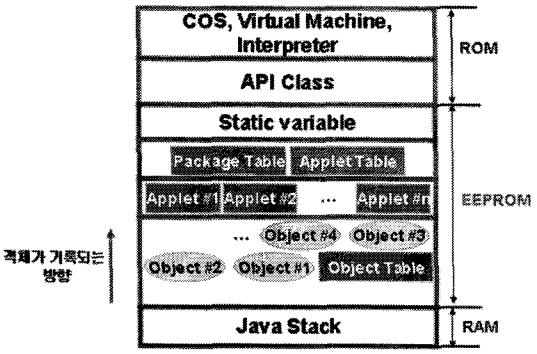


그림 7. EEPROM에 기록 되는 데이터 형태

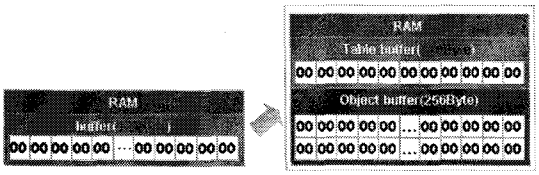


그림 8. 테이블 버퍼와 오브젝트 버퍼

그리고 가상 기계에 의해서 생성된 객체들은 데이터 필드를 가질 수도 있고 객체 상호간에 참조되고 또 다른 객체의 멤버 변수로서 이용된다. 예를 들어, 그림 9와 같이 객체 배열이 먼저 생성되고 그리고 객체 배열의 요소로서 이용되는 객체들은 객체 배열이 저장된 영역에 연속적으로 만들어진다. 배열 요소로 생성된 객체의 ID가 객체 배열에 저장된다.

객체 배열의 경우처럼 요소들이 생성될 때마다 생성된 객체 ID가 계속해서 객체 배열에 갱신된다. 이와 같이 객체 배열이 EEPROM에 있을 경우에는 객체 ID가 만들어질 때마다 EEPROM에 반영을 시켜야 되고 이로 인해 처리 속도가 느려지게 된다. 반면에 오브젝트 버퍼에 객체가 적중되어 있을 경우 RAM에 바로 반영을 할 수 있기 때문에 처리 속도를 줄일 수 있다.

따라서 앞선 경우와 같이 데이터 갱신이 요구되는 객체들 오브젝트 버퍼에 적재하고 있을 경우에 수행

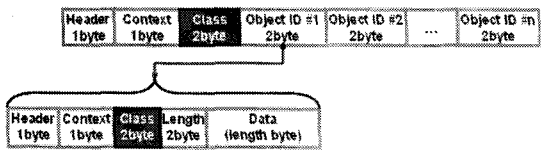


그림 9. 객체를 요소로서 갖는 객체 배열

속도가 향상될 수 있다는 것을 기대할 수 있다. 그러므로 객체데이터 특성을 고려하여 오브젝트 버퍼의 버퍼링 기능에 데이터의 적중률을 높일 수 있는 캐쉬 기능을 추가함으로 보다 더 효율적으로 버퍼를 활용할 수 있도록 본 논문에서 제안된 RAM을 활용하는 버퍼 캐쉬 기법은 선반입 LRU-OBL 방식을 자바카드 환경에 알맞게 개선시켜 적용한다. 따라서 오브젝트 버퍼는 항상 새로이 생성된 데이터를 버퍼에 저장하기 전에 생성된 데이터가 기록될 EEPROM의 메모리 주소 값을 이용해서 미리 주변 데이터들을 오브젝트 버퍼에 복사한 후에 생성된 데이터를 오브젝트 버퍼에 저장한다.

본 논문에서 자바카드 성능 개선을 위해 설계한 버퍼 캐쉬 기법의 수행 형태는 다음과 같다.

3.2.1.1 버퍼로서의 기능

따라서 그림 10에서 보는 바와 같이 자바카드에서 데이터가 생성이 되면 먼저 테이블 버퍼에 저장을 한다. 그런 후에 자바 힙 영역에 기록이 될 데이터인 경우에만 테이블 버퍼의 값을 오브젝트 버퍼로 이동을 시킨다. 256byte 크기의 오브젝트 버퍼를 효율적으로 활용하고 또한 오브젝트 버퍼의 데이터를 실제 EEPROM에 기록하기 위해서 'max'와 'min'이라는 포인터를 두어 버퍼를 관리한다. 이들 두 가지의 포인터들은 오브젝트 버퍼에 저장된 데이터들을 언제 EEPROM에 기록할 것인지에 대한 정보를 제공한다.

3.2.1.2 캐쉬의 기능

그림 11은 오브젝트 버퍼가 객체에 관련된 데이터들을 캐쉬하고 있는 상태에서 객체의 필드 값을 변경한 경우 데이터를 EEPROM에 직접 기록하는 것이 아니라 버퍼 캐쉬에 저장되어 있는 객체의 필드 부분을 변경하도록 버퍼 캐쉬를 구성하였다. 이러한 형태로 버퍼 캐쉬를 운영함으로써 EEPROM에 대한 기록 횟수를 줄일 수 있다.

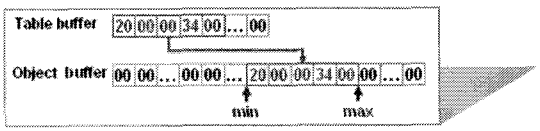


그림 10. 버퍼로서 수행 형태

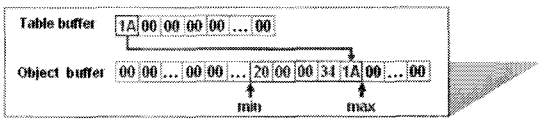


그림 11. 캐쉬 수행 형태

캐쉬의 기능에 가장 중요한 역할을 하는 것은 데이터에 대한 적중률이다. 데이터에 대한 적중률이 높을수록 EEPROM에 대한 기록 횟수를 줄일 수 있어 속도 개선에 영향을 끼칠 수 있지만, 데이터의 적중률이 낮아질 경우는 오히려 수행 속도에 악영향을 끼칠 수 있다.

자바 카드에서 특히 객체들은 내부의 필드가 변경이 되면 그 주변의 객체들에 대한 필드도 같이 갱신되는 경우 많이 발생한다. 따라서 오브젝트 버퍼가 어떠한 영역의 객체 관련 데이터들을 가지고 있느냐에 따라 캐쉬의 적중률을 높일 수 있게 된다.

3.2.1.3 데이터 기록과정

오브젝트 버퍼에서 관리되는 데이터가 실제 EEPROM 메모리에 기록이 되는 경우를 설명하고 있는 것이 그림 12이다. EEPROM에 대한 기록 시기를 결정하는 것은 버퍼를 관리하는 두 개의 포인터인 'max'와 'min'에 의해서 결정된다.

가장 기본적인 동작 방식은 'max'와 'min' 사이의 거리가 페이지 단위인 128byte가 될 때에 EEPROM에 오브젝트 버퍼 내부의 데이터들을 기록한다. 또한 발생된 데이터의 실제 주소 값이 오브젝트 버퍼에서 관리하는 주소 범위를 벗어나게 되면 오브젝트 버퍼의 데이터들을 EEPROM에 기록하고 오브젝트 버퍼를 비운 후에 새로 생성된 데이터를 저장한다.

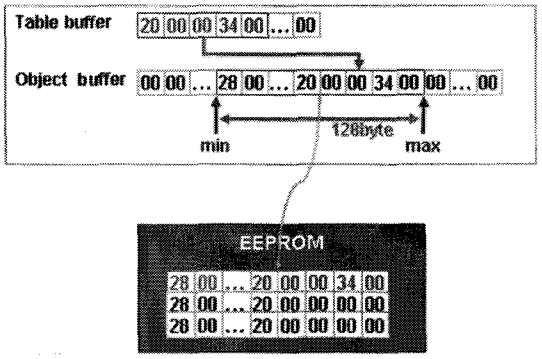


그림 12. EEPROM에 데이터 기록과정

위와 같은 방식으로 본 논문의 버퍼 캐쉬가 동작을 하게 된다.

3.3 문제점을 개선한 향상된 버퍼 캐쉬 설계

본 논문에서 제안한 버퍼 캐쉬 기법을 이용해서 간단한 실험을 하였다. 그 결과 자바 카드에 발생한 데이터를 RAM 상의 버퍼 캐쉬를 이용하여 관리함으로서 EEPROM의 기록 횟수를 줄일 수 있었다. 표 1은 본 논문에서 제안한 버퍼 캐쉬의 성능과 기존 방식의 EEPROM 기록 횟수를 비교한 것이다.

간단한 실험결과에서 보는 바와 같이 본 논문에서 제안된 버퍼 캐쉬 이용한 방법이 기존 단순 버퍼를 활용한 방법보다 약 700번 정도 EEPROM에 대한 기록 횟수를 줄임으로써 자바카드의 속도를 향상 시킬 수 있다는 것을 입증할 수 있다.

하지만 이와 같은 결과는 오브젝트 버퍼를 이용하여 집중률이 높은 객체 데이터만을 관리함으로써 자바카드 속도 개선에 상당한 영향을 미칠 것이라는 기대에는 못 미쳤다. 자바 카드 성능 향상을 위해 사용되었던 선반입 LRU-OBL 버퍼 캐쉬 기법은 속도 개선에 효율적이나 버퍼 캐쉬에서 관리되는 데이터가 실제 자바카드 실행에는 많이 이용이 되지 않는다는 결론을 얻을 수 있었다. 따라서 자바 카드의 애플릿이 실행될 때 EEPROM 메모리의 어떤 영역을 가장 많이 사용하는 지를 분석하였다. 동일한 애플릿을 이용하여 기존 버퍼링 기법을 사용하는 자바카드에서 분석한 결과는 표 2이다.

표 2 결과에서 나타나는 바와 같이 EEPROM 메모리 영역들 중 객체들이 저장되는 힙 영역을 제외하고 EEPROM에 대한 접근이 가장 많은 부분이 트랜잭션(transaction) 영역이다.

표 1. EEPROM 기록 횟수 비교

Applet 종류	EEPROM 기록 횟수	
	기존 방식	제안 방식
EMV Applet	6721	6019

표 2. EEPROM 메모리 각 영역의 기록 횟수

Applet 종류	EEPROM 기록 횟수		
	Static field	Heap	Transaction
EMV Applet	875	1236	4610

자바 카드에서 트랜잭션은 카드 내부에 데이터가 생성되거나 특정한 값이 변경이 되어져 EEPROM에 값을 저장하는 경우, 데이터를 기록하는 중에 발생할 수 있는 오류나 전원 차단 등으로부터 자바 카드를 변경 전의 상태로 복구할 수 있게 해주는 기능이다. 따라서 객체가 생성되거나 생성된 객체의 필드가 변경이 될 경우 트랜잭션은 변경 전의 데이터의 주소와 실제 기록되어 있는 값을 EEPROM의 트랜잭션 영역에 복사한 후 새로 갱신된 데이터가 저장된다. 데이터에 대한 갱신이 이루어지는 동안 아무런 오류가 발생하지 않으면 EEPROM 영역에 기록되었던 트랜잭션 데이터는 제거된다. 다음 그림 13은 트랜잭션의 수행 과정을 나타낸 것이다.

본 논문에서 제안한 버퍼 캐쉬 기법을 보다 더 효율적으로 활용하여 자바 카드의 실행 속도를 향상시키고자 이러한 트랜잭션 과정을 실제 EEPROM이 아닌 RAM 상에서 관리 하고자 한다. 트랜잭션을 RAM 상에서 관리하기 위해 오브젝트 버퍼와 동일한 형태인 LRU-OBL 기반의 트랜잭션 버퍼를 추가하여 객체 관련 데이터와 트랜잭션 관련 데이터를 구분하여 각각의 버퍼 캐쉬에서 관리하도록 한다.

따라서, 두 가지의 버퍼 캐쉬를 이용함으로써 자바 카드에서 프로그램의 설치 및 실행속도를 보다 더 개선될 수 있을 것이고 판단한다. 본 논문에서 제시되었던 버퍼 캐쉬의 문제점을 분석하고, 그 결과 EEPROM에 대한 데이터 접근이 가장 많은 힙 영역에 저장되는 객체데이터들을 관리하는 오브젝트 버퍼와 트랜잭션 영역에 저장되는 데이터를 관리하는 트랜잭션 버퍼를 활용함으로서 보다 더 향상된 속도 개선을 위한 버퍼 캐쉬 방안을 기대할 수 있다. 그림 14와 그림 15는 추가된 트랜잭션 버퍼와 알고리즘을 나타내고 있다.

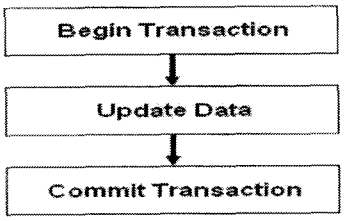


그림 13. 트랜잭션 수행과정

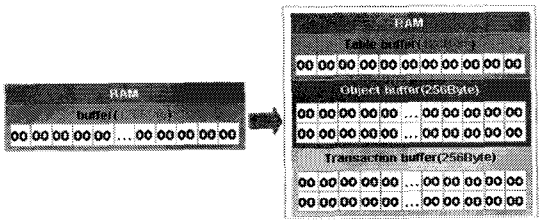


그림 14. 추가된 트랜잭션 버퍼

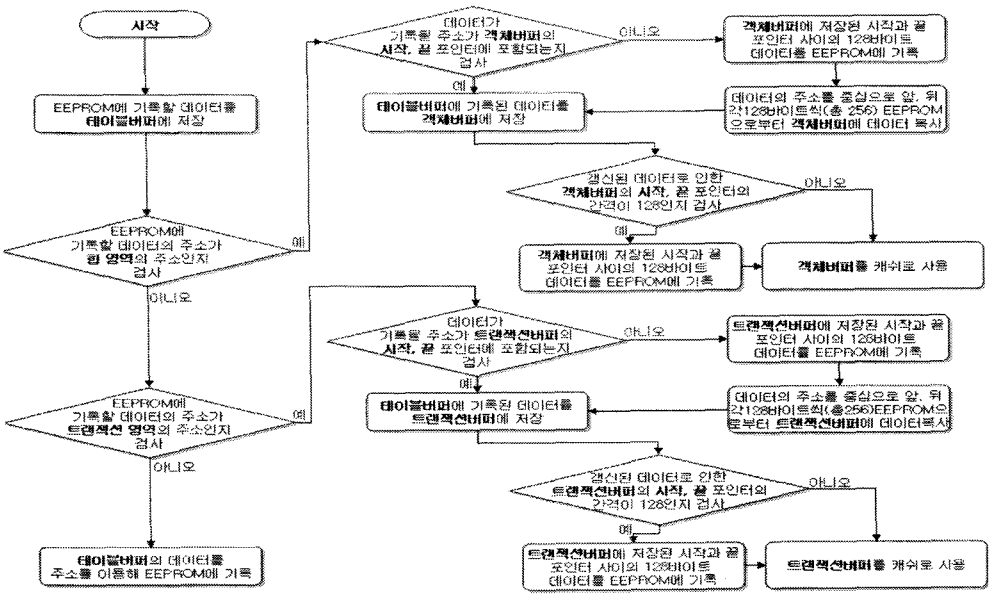


그림 15. 개선된 알고리즘

4. 실험 결과

4.1 EEPROM 기록 횟수 비교

본 논문에서 제시한 RAM을 활용한 버퍼 캐쉬 방법에 대한 결과를 기존의 방법을 사용하는 자바 카드에서 애플릿 설치까지의 EEPROM 기록 횟수를 비교하였다.

표 3에서 예를 든 애플릿들은 자바 카드에서 기본적으로 제공하는 애플릿만으로 테스트를 해본 결과이다.

표 3에 나타난 결과와 같이 본 논문에서 제시한 버퍼 캐쉬 기법이 기존 방식[2,3]보다 평균 적으로 약 80% 정도 EEPROM에 대한 기록 횟수를 줄이는 것을 알 수 있다.

4.2 수행시간 비교

본 논문에서 제시한 RAM을 활용한 버퍼 캐쉬 개선에 대한 결과를 기존의 방법을 사용하는 자바카드에서 애플릿 설치까지의 각 단계의 수행시간들을 비교한다.

표 4에 제시된 결과에서 보는 바와 같이 본 논문에서 제시한 RAM을 활용하는 버퍼 캐쉬 방법이 자바카드에 프로그램을 적재와 실행속도에서 평균 약 30% 정도 기존 방법 보다 효과적인 것을 알 수 있다.

표 3. EEPROM 기록 횟수 비교

애플릿	기존 방식	제안 방식	비율
ChannelDemo	7552	1596	79%
JavaLoyalty	7291	1322	82%
JavaPurse	22712	4537	80%
ObjDelete	16416	3025	82%
PackageA	9685	2000	79%
PackageB	7698	1406	82%
PackageC	3497	745	79%
PhotoCard	6737	1409	79%
RMIDemo	6119	1261	79%
Wallet	5641	1190	79%
EMV Small	6721	1419	79%
EMV Large	11461	2433	79%
average			80%

표 4. 애플릿 설치까지의 수행시간 비교 (단위 : ms)

애플릿	기존 방식	제안 방식	비율
ChannelDemo	76140	55406	27%
JavaLoyalty	72703	52079	28%
JavaPurse	232109	169250	27%
ObjDelete	159420	112016	30%
PackageA	90530	63172	30%
PackageB	74859	52703	30%
PackageC	32734	23188	29%
PhotoCard	64608	46172	29%
RMIDemo	57328	40437	29%
Wallet	57140	41844	27%
EMV Small	61766	43187	30%
EMV Large	119812	88671	26%
average			30%

본 논문에서 제시한 오브젝트 버퍼와 트랜잭션 버퍼 변경을 통한 자바카드의 속도 개선에 대한 실험결과를 살펴보면 상당히 효과적인 것을 알 수 있다.

하지만 표 5에서 나타난 결과에서 보면 본 논문에서 제시된 버퍼 캐쉬 방법이 기존의 방법보다는 EEPROM에 기록하는 횟수는 약 80% 정도 줄인 반면에 수행시간은 약 30% 밖에 줄어들질 않는다. 결과에서 보듯이 확연히 줄어든 EEPROM 기록 횟수에 비하면 애플릿 설치 시간은 개선은 그다지 크다고 볼 수 없다.

이러한 결과가 나타나는 가장 큰 원인으로는 자바카드에서 데이터를 EEPROM에 기록하기까지 반드시 거쳐야 하는 모듈 호출과 시스템 연산에 관한 수행들이 많이 발생하기 때문이다.

하지만 본 논문에서 제시한 선반입 기반의 버퍼 캐쉬를 적용한 자바카드의 실행속도가 기존 버퍼 방식을 사용하는 자바카드의 실행속도 보다 약 30% 정도 시간을 단축시켰다.

표 5. 개선된 성능

	EEPROM 기록 횟수	애플릿 설치 시간
개선된 성능	80%	30%

5. 결 론

본 논문에서는 자바카드 프로그램 적재 및 실행속도 향상을 위해서 RAM을 활용하여 EEPROM에 대한 기록 횟수를 줄이는 방안을 제시하고 구현하였다.

기존 자바카드에서는 데이터가 발생할 때마다 즉시 EEPROM에 기록하는 단순한 버퍼링 방식을 사용하여 자바카드의 수행 속도에 영향을 끼쳤다. 하지만 본 논문에서 제안한 버퍼링 방식은 EEPROM에 기록되는 데이터의 특성, 집중률, 적중률을 고려한 선반입 LRU-OBL 방식의 버퍼 캐쉬 방법이 효과적이라는 것을 실험 결과로서 알 수 있었다.

따라서 버퍼 캐쉬 변경은 자바카드 애플릿 설치 및 수행 속도를 개선시킬 수 있다는 것을 알 수 있었다. 이러한 방법을 활용한 자바카드는 기존 버퍼링 방식만을 사용한 것에 비해 수행 속도의 평균 30% 정도 줄어들었다는 것을 실험을 통해 알 수 있었다.

이러한 연구 결과를 통해 스마트카드뿐만 아니라 SIM/USIM Chip 등에 보다 안정적이고 빠른 자바카드 기술을 공급할 수 있게 될 것이다.

EEPROM과 같은 비휘발성 저장매체를 사용하는 임베디드 시스템 개발에서 수행 속도 개선에 대한 새로운 방안을 제시할 수 있다.

참 고 문 헌

[1] Uwe Hansmann, Martin S. Nicklous, Thomas Schack, Frank Seliger, *Smart Card Application Development Using Java*, Springer, 2002.

[2] Zhiqun Chen, *Java Card Technology for Smart Cards*, Addison-Wesley, 2000.

[3] Wolfgang Rankl and Wolfgang Effing, *Smart Card Handbook*, John Wiley & Sons, 2003.

[4] Sun Microsystems, Inc., cJDK_Users_Guides, SUN, 2002.

[5] Sun Microsystems, Inc., *The Java CardTM 2.2 Runtime Environment(JCRE) Specification*, SUN, 2002.

[6] Sun Microsystems, Inc., *The Java CardTM 2.2 Virtual Machine Specification*, SUN, 2002.

[7] T. Lindholm and F. Yellin, *The JavaTM Virtual Machine Specification ADDISON-WESLEY*, 1997.

[8] 탁승호, Let's Smart Card 스마트 카드, 성안당, 서울, 2004.

[9] http://java.sun.com/, Sun Microsystem, Java Home Page.

[10] Yoon-Young. Lee, Chei-Yol Kim, and Dae-Wha Seo, "Dynamic File Prefetching Scheme based of File Access Patterns," *Journal of KISS*, Vol.29, No.7, pp. 384-393, August 2002.

[11] H. Seok Jeon and Sam H. Noh, "An Efficient Buffer Cache Management Alogorithm based on Prefetching," *Journal of KISS*, Vol.27, No. 5, pp. 529-539, May 2000.

[12] Jae-Deok Lim, Jong-Hyun Cho, and Dae-Wha Seo, "Prefetching Policy based on File Access Pattern And Cache Size," *KIPS*, Vol. 7, No.2, pp. 625-628, May 2000.

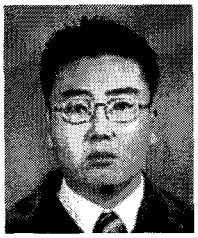
[13] Pei Cao and Edward W. Felton, "Implementation and Performance of Integrated Application-Controlled File Caching, Prefetching, and Disk Scheduling," *ACM Transactions on Computer Systems*, Vol.14, No.4, pp. 311-343, November 1996.

[14] Pei Cao, Edward W. Felton, Anna R. Karlin, and Kai Li, "A Study of Integrated Prefetching and Caching Strategies," *ACM SIGMETRICS*, Vol.23, No.1, pp. 188-197, May 1995.

[15] R. Hugo Patterson, garth A. Gibson, Eka Ginting, Daniel Stodolsky, and Jim Zelenka, "Informed Prefetching and Caching," *ACM-SIGOPS*, Vol.29, No.5, pp. 79-95, December 1995.

[16] H. Seok Jeon and Sam H. Noh, "Dynamic Buffer Cache Management Scheme Based on Simple and Aggressive Prefetching," *USENIX ALS Conference*, Vol.4, pp. 27-38, October 2000.

[17] Alan Jay Smith. "Disk cache-miss ratio analysis and design considerations," *ACM Transactions on Computer Systems*, Vol.3, No.3, pp. 61-203, August 1985.



오 세 원

2006년 2월 경남대학교 컴퓨터공학부 학사
2006년 3월~현재 경남대학교 컴퓨터공학과 석사 과정
관심분야 : RFID Technology, Smart Card, Java Card, 모바일 컴퓨팅

최 원 호

1999년 경남대학교 전산통계학과 학사
2001년 경남대학교 컴퓨터공학과 석사
2005년 경남대학교 컴퓨터공학과 박사
관심분야 : Embedded, Java Card, Java Machine

정 민 수

1986년 서울대학교 컴퓨터공학과 학사
1988년 한국과학기술원 전산학과 석사
1994년 한국과학기술원 전산학과 박사
1990년~현재 경남대학교 컴퓨터공학부 교수
관심분야 : Java Technology, Java Machine, Home-Networking