자바 바이트코드 최적화기의 설계

○
황순명, 오세만
동국대학교 컴퓨터공학과
{smhwang, smoh}@dgu.ac.kr

Design of a Java Bytecode Optimizer

○
Soonmyung Hwang and Seman Oh
Department of Computer Engineering
Dongguk University

요 약

자바 클래스 파일은 각 플랫폼에 독립적인 중간 코드 형태의 바이트코드와 자료 구조로 구성됨으로서 네트워크를 통하여 플랫폼에 독립적으로 인터프리티브 컴파일링 시스템에서 실행된다. 그러나 클래스 파일을 생성하는 자바 컴파일러는 각 플랫폼에 독립적인 바이트코드 표현에 제약을 받기 때문에 효율적인 코드를 생성하는데 한계가 있다. 또한, 자바 가상 기계에서 동적 링킹을 지원하기 위하여 고안된 상수 풀(constant pool)의 크기가 상대적으로 큰 특징을 갖는다. 따라서 자바 클래스 파일이 네트워크와 같은 실행 환경에서 효과적으로 실행되기 위해서는 작은 크기와 효율적인 코드에 대한 변환이 요구된다.

본 논문에서는 자바 클래스 파일이 인터넷 및 분산환경 시스템에서 효율적으로 실행되기 위해서 클래스 파일의 크기를 줄이는 방법과 자바 컴파일러가 생성한 바이트코드에 대해 최적화를 수행하는 최적화 방법론을 제시하고, 코드 최적화기를 설계한다. 최적화된 클래스 파일은 코드 크기를 줄이고, 효율적인 코드를 생성함으로서 네트워크 상의 전송 속도뿐만 아니라 가상 기계에서 좀 더 빠르게 실행할 수 있다.

1. 서 론

자바 프로그래밍 언어는 인터넷 및 분산 환경 시스템에서 효과적으로 응용 프로그램을 작성할 수 있도록 설계된 언어로서 객체지향 패러다임의 특성 및 다양한 개발 환경을 지원하고 있다. 자바 프로그래밍 환경에서는 이기종간의 실행 환경에 적합하도록 가상 기계(Virtual Machine) 코드인 바이트코드를 지원한다. 자바 컴파일러가 생성하는 클래스 파일은 각 플랫폼에 독립적인 중간 코드 형태의 바이트코드와 자료 구조로 구성됨으로서 네트워크를 통하여 플랫폼에 독립적으로 인터프리팅되어 실행된다. 따라서 인터넷을 기반으로 하는 모든 응용 프로그램들은 플랫폼에 독립적인 자바 클래스 파일로 변환하는 방안을 연구하고 있는 추세이다.

현재까지는 자바 프로그램이 네트워크 속도의 제한과 더불어 바이트코드를 인터프리터 방식으로 실행함으로써 속도면에서 많은 제약을 받는다. 또한 자바 컴파일러는 각 플랫폼에 독립적인 중간코드 표현에 제약을 받기 때문에 효율적인 바이트코드를 생성하는데 한계가 있고, 바이트코드는 스택을 기반으로 하는 가상기계 언어로서 또한 많은 단점을 가지고 있다. 또한, 자바 가상 기계(Java Virtual Machine: JVM)에서 동적 링킹(dynamic linking)을 지원하기 위하여 고안된 상수 풀의 크기가 상대적으로 매우 큰 특징을 갖는다. 따라서, 자바 클래스 파일이 네트워크와 같은 실행 환경에서 빠른 전송과 효율적으로 실행될 수 있도록 클래스 파일에 대한 최적화는 중요한 요소이다[8][9].

본 연구는 자바 클래스 파일이 인터넷 및 분산환경 시스템에서 효율적으로 실행되기 위해서 자바 컴파일러가 생성한 클래스 파일의 크기를 줄이는 방안을 제시하고, 바이트코드에 대해 최적화를 수행하는 최적화 방법론을 제시하고, 코드 최적기를 설계한다. 최적화된 클래스 파일은 코드 크기를 줄이고, 효율적인 코드를 생성하므로서 네트워크 상의 전송 속도뿐만 아니라 가상 기계에서 좀 더 빠르게 실행할 수 있다.

본 논문의 구성은 2장에서 자바 컴파일러가 생성하는 클래스 파일의 구조와 자바 바이트코드에 대하여 알아보고, 기존에 제시된 최적화 기법 및 보안을 위한 검증기를 설명한다. 3장에서는 자바 클래스 파일의 최적화 시스템의 개요와 바이트코드에 적용된 최적화 기법을 제시한다. 마지막으로 4장에서는 결론과 향후 연구방향에 대해서 기술한다.

본 연구는 정보통신연구관리단의 대학기초연구(과제명: Java 바이트코드를 위한 최적화기의 설계 및 구현) 지원으로 수행되었습니다.

2. 관련 연구

자바 클래스 파일이 JVM에서 효율적으로 실행하기 위한 최적화 방법으로 바이트코드에 대한 최적화를 수행하는 일은 중요하다. 자바 바이트코드는 스택 기반 실행 모델을 사용하기 때문에 최적화를 적용하기에 다소 복잡한 단점을 갖는다. 따라서, 기존에 제시된 바이트코드에 대한 최적화 기법을 알아보고, 최적화된 변환 클래스 파일이 네트워크를 통하여 전송될 때 보안(security)을 위한 검증기(verifier) 과정을 알아본다.

2.1 자바 클래스 파일

자바 컴파일러가 생성하는 클래스 파일은 바이트코드라는 가상 기계 명령어 셋과, JVM에서 동적 링킹을 지원하기 위한 상수 풀로 구성되어 있고, 바이트 스트림으로 이루어져 있다. 모든 16비트, 32비트, 64비트의 데이터들은 8비트 단위로 읽혀져 해당 플랫폼에 알맞은 형태로 번역된다.

클래스 파일의 구조는 대부분 상수 풀과 바이트코드 열로 이루어져 있다. 상수 풀은 여러 가지 형태의 클래스 파일내 상수들을 배열의 형태로 가지고 있고, 상수 내용을 가리키는 부분과 실제 데이터를 가지고 있는 부분으로 구성되어 있다[4][7].

자바 바이트코드는 스택 기반 가상 기계어(abstract machine language)로서 스택에 대한 동작들을 정의한다. 바이트코드는 인터프리트되기 쉬운 장점을 갖지만, 스택 기반 기계는 연산을 일련화하고, 값을 재사용할 수 없기 때문에 불필요한 적재(load)와 저장(store) 명령어가 많아진다는 단점을 가지고 있다. 또한 바이트코드는 구조상 소스 프로그램이 가지는 다양한 정보를 컴파일 과정에서 상실한다. 자바 컴파일러는 기본적으로 간단한 최적화만을 수행하기 때문에 보다 향상된 최적화 기법이 요구된다.

2.2 자바 바이트코드 최적화

자바 바이트코드를 최적화하는 방법은 크게 5가지 범주로 나눌 수 있다. 첫째, 자바 바이트코드에 대한 최적화를 수행한 후 새로운 클래스 파일을 생성하는 방법이다[4]. 둘째, 바이트코드를 분석한 후 추가 정보를 가진 새로운 클래스 파일을 생성하여 JVM에서 효율적으로 실행하도록 하는 방법이다. 셋째, 바이트코드를 조작할 수 있는 프레임을 제공하는 방법이다. 넷째, 자바 애플리케이션 패키지를 이용하는 방법과 마지막으로 네이티브 실행 코드로 컴파일하는 방법이다[3][6][8].

2.3 검증기(verifier)

네트워크를 통하여 다운로드 받는 클래스 파일은 JVM의 보안 모델을 통하여 검증을 받게 된다. JVM에서 검증기는 보안 기법의 핵심 요소이다.

(1) 클래스 파일 검증기(class file verifier)

자바가 원격 사이트로부터 클래스 파일을 다운로드 받을 때마다 JVM은 클래스 파일을 검증하게 된다. 클래스 파일 검증기는 많은 구조적 제약 조건을 클래스 파일에 적용함으로써 클래스 파일 안에 있는 데이터의 레이아웃을 체크한다.

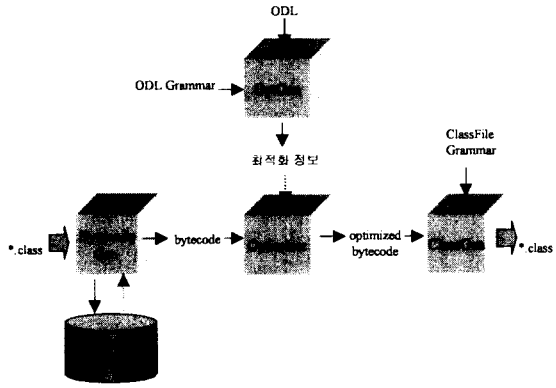
(2) 바이트코드 검증기(bytecode verifier)

바이트코드 검증기는 메소드 안에 있는 바이트코드를 체크하는 프로그램으로서 올바르게 동작할 수 있도록 보장한다. 바이트코드 검증기는 각 명령어의 스택, 지역변수에 대한 값과 타입을 검사한다[4][7].

3. 자바 바이트코드의 최적화기의 설계 및 구현

3.1 바이트코드 최적화기의 설계

자바 바이트코드 최적화 시스템의 개요는 [그림 1]과 같은 구조를 통하여 자바 클래스 파일을 입력으로 받아 최적화된 클래스 파일을 출력한다.



[그림 3] 자바 바이트코드 최적화 시스템 개요

최적화기의 구성은 크게 BytecodeGen, Optimizer, 그리고 ClassGen으로 구성되어 있다. 입력으로 받은 클래스 파일은 BytecodeGen을 통하여 상수 풀에 있는 정보를 포함한 바이트코드 열을 출력한다. Optimizer는 바이트코드의 명령어를 최적화하고, 최적화된 바이트코드를 통하여 새로운 클래스 파일을 생성하는 ClassGen으로 구성되어 있다.

3.2 BytecodeGen

BytecodeGen은 클래스 파일을 입력으로 받아 상수 풀에 있는 모든 상수 정보를 포함하는 [그림 2]와 같은 클래스 파일 포맷에 따라 코드를 생성한다.

```
[abstract] [final] [public] class classname
[extends superclassname] [implements interfacename {,
    interfacename}]
{
    [field_declarations]
    [method_declarations]
}
```

[그림 2] BytecodeGen 출력 포맷

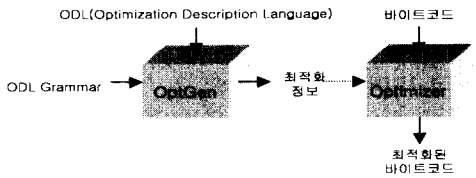
위 [method_declarations] 부분이 바이트코드가 들어가는 부분이고, 다음 단계인 Optmizer에서 최적화가 이루어진다. [그림 3]은 BytecodeGen의 출력 예를 보여준다.

```
class HelloWorld
{
  method public static void main(java.lang.String[])
  max_stack 2
  {
      getstatic  java.io.PrintStream java.lang.System.out
      ldc  "Hello World!"
      invokevirtual  void
      java.io.PrintStream.println(java.lang.String)
      return
  }
}
```

[그림 3] BytecodeGen의 출력 예

3.3 Optimizer

Optimizer는 본 논문의 핵심 단계로서 바이트코드를 최적화하는 역할을 담당한다. 바이트코드를 최적화하는 기법은 크게 두가지로서 일반적인 최적화 기법과 자바 바이트코드에 의존적인 최적화 기법으로 분류된다. 일반적인 최적화 기법은 핍홀 최적화 기법을 적용하여 OptGen에 의하여 최적화를 수행하고, 바이트코드에 의존적인 최적화는 Optimizer에 의하여 수행된다. Optimizer에서 수행되는 기법은 선행연구[10]에서 제시된 기법을 적용하고, 객체지향 언어의 단점인 메소드 호출 resolution의 시간을 단축시키기 위하여 메소드 인라인 기법을 적용한다.



[그림 4] 최적화기의 구성

(1) OptGen

OptGen은 바이트코드의 핍홀 최적화를 수행하기 위하여 자동으로 최적화 정보를 생성하는 최적화기 생성기이다. 최적화에 필요한 정보를 기술하는 테이블은 [그림 5]와 같고, 최적화 패턴을 기술하는 부분이 엔트리 부분이다. 따라서, [그림 4]와 같이 ODL을 입력으로 받아 Optimizer에서 이용할 수 있도록 최적화기 정보를 자동 생성한다.

```
테이블  →  매개변수 선언
           '%%'
           변수 선언
           '%%'
           엔트리
           '%%'
           사용자 정의 루틴.
```

[그림 5] 최적화 패턴 기술 테이블

3.4 ClassGen

ClassGen은 Optimizer에서 최적화된 바이트코드에 대하여 올바른 클래스 파일을 새롭게 생성하는 역할을 한다. 네트워크를 통하여 다운로드 받는 클래스 파일은 JVM의 보안 모델을 통하여 엄격히 검증을 받는다. 따라서, ClassGen은 JVM의 클래스 검증기와 바이트코드 검증기를 통과할 수 있도록 클래스를 생성한다.

또한, 클래스 파일에서 많은 크기를 차지하는 상수 풀의 공간을 최대한 작게 구성한다. 소스 레벨에서 프로그래머가 지정하는 명칭(identifier)의 길이가 길면 길수록 상수 풀의 크기는 증가하지만 프로그램 실행에 전혀 영향을 미치지 않는다. 따라서, ClassGen은 명칭 테이블을 관리함으로써 내부적으로 긴 명칭을 짧은 명칭으로 대체한다. 따라서 클래스 파일의 크기를 크게 줄임으로서 네트워크를 통한 전송시간을 단축시킨다.

4. 결론 및 향후 연구 방향

본 연구는 자바 클래스 파일이 인터넷 및 분산환경 시스템에서 효율적으로 실행되기 위해서 클래스 파일의 크기를 줄이는 방안을 제시하였고, 자바 컴파일러가 생성한 바이트코드에 대해 최적화를 수행하는 최적화 방법론을 제시하였으며, 코드 최적화기를 설계하였다. 최적화된 클래스 파일은 코드 크기를 줄이고, 효율적인 코드를 생성함으로써 네트워크 상의 전송 속도 뿐만 아니라 JVM에서 효율적으로 실행될 수 있다.

본 시스템은 자바 컴파일러에 의해 생성된 코드를 오프라인으로 최적화를 수행하기 때문에, 바이트코드에 대한 충분한 분석과 최적화가 가능하다. 최적화된 바이트코드는 코드의 크기를 줄이고, 효율적인 코드를 생성하므로서 네트워크 상의 전송 속도뿐만 아니라 가상 기계에서 빠르게 실행할 수 있는 잇점을 얻는다. 특히, 최근 자바를 이용한 웹 응용프로그램들이 보편화되고 다양해짐에 따라 자바 클래스 파일의 크기와 속도를 효율적으로 최적화한다면 이를 이용하는 응용 범위는 매우 넓어질 것이다.

향후 연구 방향은 자바 바이트코드의 특성에 맞는 최적화 기법들을 구현하고 병행적으로 연구하며, 최적화 패턴을 찾아 테이블에 계속 반영하고자 한다. 또한 최적화된 클래스 파일에 대한 충분한 실험과 적합한 성능 평가 방법을 고안할 예정이다.

5. 참고 문헌

[1] Jack W. Davidson and Cristopher W. Fraser, Automatic Generation of Peephole Optimizations, Proceedings of the ACM SIGPLAN '84 Symposium on the Compiler Construction SIGPLAN Notices, Vol. 19, No.6, pp.111-116, 1984.

[2] J.Hummel, A.Azevedo, D.Kolson and A.Nicolau, Annotating the Java Bytecodes in Support of Optimization, 1997.

[3] Jonathan Hardwick, "Java Optimization", 1997

[4] Jon Meyer and Troy Downing, Java Virtual Machine, O'REILLY, 1997.

[5] Lars R. Clausen, A Java Bytecode Optimizer Using Side-effect Analysis, Lars Raeder Clausen, Aarhus University, Denmark, 1997.

[6] Raja Vallée-Rai, Phong Co, Etienne Gagnon, Laurie Hendren, Patrick Lam, and Vijay Sundaresan, Soot - a Java Bytecode Optimization Framework, Sable Reaearch Group, School of Computer Science, McGill University.

[7] Tim Lindholm and Frank Yellin, The Java™ Virtual Machine Specification, Addison Wesley, 1996.

[8] Th.Kistler and M.Franz, A Tree-Based Alternative to Java Byte-Codes, In Proceedings of International WorkShop on Seurity and Efficiency Aspects of Java '97, 1997.

[9] Steven Muchnick, Advanced Compiler Design and Implementation, Morgan Kaufman publishers, 1997.

[10] 황순명,조창오,오세만, 자바 바이트코드 최적화, 한국정보처리학회 98추계 학술발표논문집, 5권 2호, pp.1264~1267, 1998.