사물인터넷을 위한 경량 가상기계의 설계 및 구현

최찬휘*, 손윤식**, 정준호***, 이양선*
*서경대학교 컴퓨터공학과
**동국대학교 컴퓨터공학과
***동국대학교 경주캠퍼스 전자상거래연구소
e-mail:sftblw@skuniv.ac.kr

Design and Implementation of the Light-Weight Virtual Machine for the Internet of Things

Chanwhi Choi*, Yunsik Son**, Junho Jeong***, Yangsun Lee*
*Dept of Computer Engineering, SeoKyeong University
**Dept of Computer Engineering, Dongguk University
***Electronic Commerce Institute, Dongguk University Gyeongju Campus

요 약

다양한 사물인터넷 장치 환경간의 응용 프로그램의 플랫폼 독립성을 지원하기 위해 가상기계 기술을 사용할 수 있다. 그러나 사물인터넷 장치 환경은 가용 메모리가 한정적이므로 사물인터넷 환경에서 가상기계의 동작이 가능하게 하려면 적은 메모리를 사용하도록 경량화해야 한다. 본 논문에서는 경량의 가상기계를 설계하여 가용 메모리가 적은 저성능 사물인터넷 기기에서도 동작할 수 있게 하였다. 또한 가상기계를 구조적으로 설계하여 다양한 사물인터넷 장치의 성능에 따라 적합한 구성으로 이식할 수 있다.

1. 서론

사물인터넷 장치의 환경은 다양하여 한 번 작성한 프로그램을 다른 환경에서 재사용하기 어렵다. 응용 프로그램의 플랫폼 독립성을 지원하기 위해 가상기계 기술을 사용할 수 있으나, 사물인터넷 환경은 개인용 컴퓨터에 비해 가용 메모리가 부족하여 가상기계가 메모리를 적게 사용하도록 경량화해야 할 필요가 있다.

본 논문에서는 다양한 사물인터넷 장치에서 동작할 수 있는 경량의 가상기계를 설계하고 구현한다. 경량 가상기계는 기존 연구결과로 개발된 스마트 가상기계[2,3]를 사물인터넷 장치에 적합하도록 경량화한 것으로, 가용 메모리가 적은 저성능 사물인터넷 기기에서도 동작할 수 있으며, 구조적인 설계로 다양한 사물인터넷 기기의 성능에 따라 적합한 구성으로 이식할 수 있다.

2. IoT-Cloud 융합 가상기계 시스템

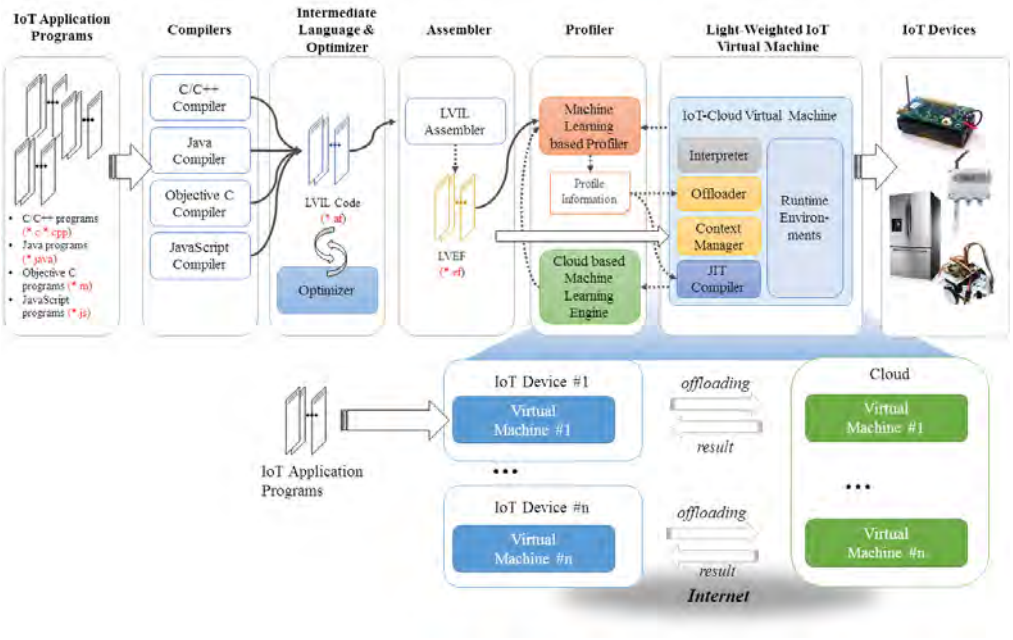
IoT-Cloud 융합 가상기계 시스템은 기존의 저성능 사물인터넷 장치에 클라우드 서버의 연산력을 결합하여 단말장치의 변경 없이도 높은 성능을 부여하는 시스템으로, 별도의 응용 프로그램 수정 없이 실행 성능을 향상시키도록 설계되었다. 그림 1은 IoT-Cloud 융합 가상기계 시스템의 전체 구조도이다.

이 시스템은 경량의 가상기계, 기계학습 기반 프로파일러(profiler), 오프로딩(offloading) 기술[4], JIT 컴파일러를 활용하여 효율적인 실행환경을 제공한다. 경량의 가상기계에 의해 다양한 사물인터넷 플랫폼에 무관하게 응용 프로그램을 개발 및 재사용할 수 있으며, 오프로딩 기술로 클라우드의 연산력을 단말장치에 부여하여 보다 높은 성능을 요구하는 작업을 응용 프로그램의 수정 없이 수행할 수 있다. 효과적인 오프로딩을 위해 정적, 동적, 기계학습 기반 프로파일러와 콘텍스트 관리 기술을 사용할 것이며, JIT(Just-In-Time) 컴파일러를 구현하여 가상기계가 인터프리트 기반으로 실행되어 네이티브보다 느리게 실행되는 단점을 보완할 것이다.

3. 경량 가상기계

경량 가상기계는 구조적 설계와 적은 메모리 사용으로 다양한 사물인터넷 장치에서 동작할 수 있는 가상기계 솔루션으로, 구조적으로 가상기계를 설계하여 장치에 적합한 구성으로 이식할 수 있고, 명령어 집합을 제약하지 않고도 보다 적은 메모리를 사용하여 명령어를 로드하는 방법을 적용하여 실행 성능을 거의 유지하면서도 응용 프로그램

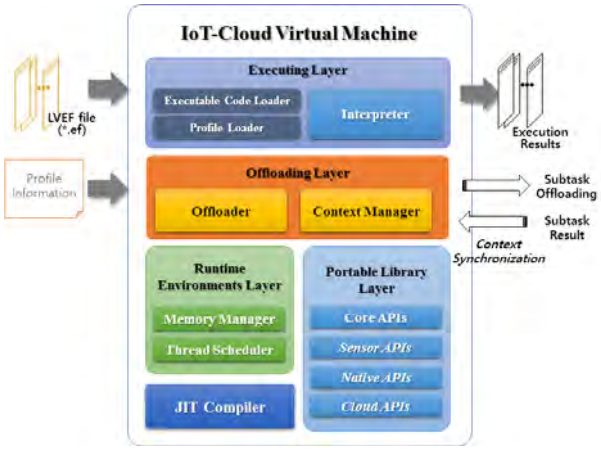
"이 논문은 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (No.2016R1A2B4008392)"



(그림 1) IoT-Cloud 융합 가상기계 시스템 구성도

로드에 더 적은 메모리를 사용한다.

경량 가상기계는 현재 실행파일 로더, 인터프리터, 실행 환경, 포터블 라이브러리로 구성되며 이후 IoT-Cloud 융합 가상기계 시스템을 위해 오프로딩과 JIT 컴파일러를 개발하여 확장할 예정이다. 그림 2는 경량 가상기계의 전체 시스템 구성도이다.



(그림 2) 경량 가상기계의 전체 시스템 구성도

3.1 가상기계 구성요소의 구조적 설계

사물인터넷 장치는 기기 종류와 성능 편차가 다양하며 응용 프로그램의 목적도 각기 다르다. 그렇기 때문에 가상기계를 구조적으로 설계하여 특정 기기의 환경과 응용 프로그램의 목적에 적합하게 구성하여 이식할 수 있도록 하였다.

경량 가상기계는 코어 모듈에 스레드 모듈과 오프로딩 모듈을 필요에 따라 결합하여 현재 총 4가지 구성으로 이식할 수 있다. 코어 모듈은 가상기계 실행에 필수적인 구성요소인 실행파일 로더, 인터프리터, 포터블 라이브러리로 구성되고, 스레드 모듈은 실행환경 구성요소의 스레드 스케줄링 기능으로 구성되며, 오프로딩 모듈은 오프로딩 및 콘텍스트 관리 기능으로 구성된다.

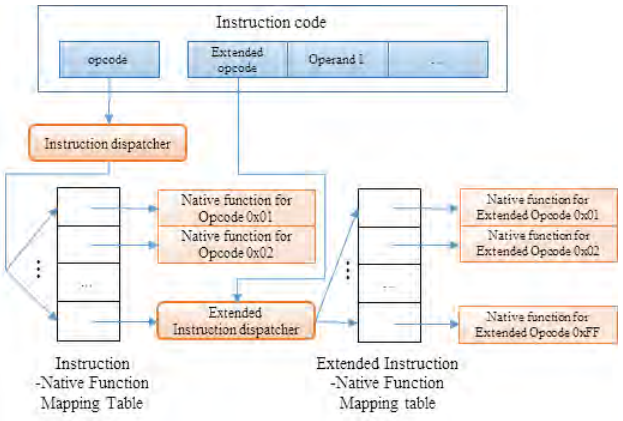
3.2 응용 프로그램의 메모리 사용량 경량화

응용 프로그램이 커질수록 가상기계에 로드된 응용 프로그램이 차지하는 메모리가 커진다. 그러므로 가상기계가 응용 프로그램의 로드에 더 적은 메모리를 사용하도록 경량화할 필요가 있다.

가상기계에 로드되는 응용 프로그램의 메모리 사용량을 줄이기 위해 명령어 중 연산 코드(Opcode)가 적재되는 공간을 기존의 2byte에서 1byte로 줄일 수 있다. 그러나 경량 가상기계의 명령어 집합은 기본 명령어 204가지와 최적화 명령어 203가지로 구성되며[5-6], 연산 코드의 적재 공간을 1byte로 축소할 경우 1byte에 적재할 수 있는 256가지 명령어 중 기본 명령어를 모두 적재하였을 때 최적화 명령어를 52가지만 적재할 수 있다. 따라서 응용 프로그램의 실행 성능이 크게 감소할 수 있다.

사용 가능한 명령어를 256가지 이내로 제약하는 대신 명령어를 일차 명령어와 확장 명령어 두 수준으로 나누어 실행 속도의 큰 저해 없이 모든 종류의 명령어를 적재할 수 있도록 하였다. 일차 명령어의 연산 코드가 0xFF인 경우 피연산자 영역에 추가로 1byte를 할당하여 확장 연산 코드로 사용하도록 구성하였다. 이 때 일차 명령어의 연산 코드가 0xFF인지 구분하기 위한 목적으로 조건문을 사용하여 일차 명령어의 실행 성능도 감소하는 상황을 피하기 위해 조건문을 추가하는 대신 일차 명령어 0xFF의 실행부에서 확장 연산 코드를 이차적으로 디스패치(dispatch)

하여 실행한다. 그림 3은 연산 코드를 두 단계로 나누어 실행하는 절차에 대한 구조도이다.



(그림 3) 연산 코드의 두 단계 구분 실행 구조도

4. 실험 및 분석

표 1은 200까지의 완전수를 구하여 출력하는 프로그램을 Raspberry Pi B+에서 코어 모듈만으로 구성된 경량 가상기계로 실행한 결과이다.

<표 1> 완전수를 구하는 프로그램 perfect.c의 컴파일 및 경량 가상기계에서의 실행 화면

perfect.c

```
#include "sys_lib.h"

const int max = 200;
void main()
{
   int i, j, k, rem, sum;
   i = 2;
   while (i <= max) {
      sum = 0;
      k = i / 2;
      j = 1;
      while (j <= k) {
         rem = i % j;
         if (rem == 0)
            sum += j;
         ++j;
      }
      if (i == sum)
         printf("%d\n", i);
      ++i;
   }
}
```

perfect.lvaf

```
%%HeaderSectionStart
                    ...중략...
%%HeaderSectionEnd
%%CodeSectionStart
   %FunctionStart
                    ...중략...
      .opcode_start
         proc       20      1       1
         ldc.i      2
         str.i      1       0
      %Label        ##0
         lod.i      1       0
         lod.i      0       $max
                    ...중략...
         ujp ##0
      %Label        ##1
         ret
      .opcode_end
   %FunctionEnd
%%CodeSectionEnd
%%DataSectionStart
                    ...후략...
```

실행 결과

```
pi@raspberrypi:~/lwvm $ ./LWVM_core perfect.lvef
6
28
pi@raspberrypi:~/lwvm $
```

가상기계를 핵심 구성요소만 포함하도록 빌드한 경우 50KB대의 크기로 가용 메모리가 적은 환경에서도 가상기계를 구동할 수 있다. 표 2는 경량 가상기계의 구성요소 포함여부에 따른 실행파일 크기를 나열한 표이다. Raspberry Pi B+, g++ 4.9.2 환경에서 Boost 라이브러리 1.55 버전을 사용하여 컴파일하였고, 실행에 불필요한 심볼 정보를 제거하였다.

<표 2> 구성요소의 포함 여부별 가상기계 실행파일의 크기

경량 가상기계의 구성요소 포함 여부	크기 (KB)
코어	50.00
코어 + 스레드	80.49
코어 + 오프로딩	118.54
코어 + 스레드 + 오프로딩	142.58

기존 가상기계와 비교하여 경량 가상기계는 응용 프로그램을 로드하는데 보다 적은 메모리를 사용한다. 표 3은 응용 프로그램의 로드 크기 경량화 기법의 적용 여부에 따른 메모리 사용량을 나타낸 표이다. 경량화 기법의 타당성을 검증하기 위해 기존 가상기계에 경량 가상기계에 적용한 방법과 동일한 경량화 기법을 적용한 뒤 게임 콘텐츠를 로드하여 메모리 사용량을 비교하였다.

<표 3> 응용 프로그램의 메모리 사용량 경량화 기법 적용 여부에 따른 가상기계의 메모리 사용량

대상 프로그램	경량화 기법 미적용(Byte)	경량화 기법 적용(Byte)	메모리 사용 증감(Byte)
Gaza	238,024	211,562	-26,642
Aiolos	1,633,184	1,388,016	-245,168
Joro	1,552,284	1,388,821	-163,463

5. 결론

본 논문에서는 다양한 사물인터넷 장치에서 동작할 수 있는 경량의 가상기계를 설계하고 구현하였다. 경량 가상기계는 구조적인 설계로 다양한 사물인터넷 장치에 적합한 구성으로 이식할 수 있으며, 가용 메모리가 적은 사물인터넷 장치에서 실행될 수 있도록 응용 프로그램 로드에 더 적은 메모리를 사용한다.

개발한 가상기계를 기반으로 사물인터넷 기기를 위한

API, 클라우드의 컴퓨팅 파워를 사용하기 위한 오프로딩 모듈, 가상기계의 실행성능을 향상시키기 위한 JIT 컴파일러 모듈 등을 개발 및 개선하여 가상기계에서의 실행성능이 네이티브에 비해 부족한 문제점을 해결하고, 더 나아가 오프로딩 지원을 강화하여 기기에 높은 성능을 부여하는 효과를 얻도록 할 것이다.

참고문헌

[1] Botta, Alessio, et al. "Integration of cloud computing and internet of things: a survey," Future Generation Computer Systems, Vol.56, pp.684-700, 2016

[2] Y.S. Lee, Y.S. Son "A Study on the Smart Virtual Machine for Executing Virtual Machine Codes on Smart Platforms," International Journal of Smart Home, SERSC, Vol.6, No.4, pp.93-105, 2012.

[3] Y.S. Lee, Y.S. Son, "A Study on the Smart Virtual Machine for Smart Devices," Information-an International Interdisciplinary Journal, Vol.16, No.2, International Information Institute, pp.1465-1472, 2013.

[4] KUMAR, Karthik, et al. A survey of computation offloading for mobile systems. Mobile Networks and Applications, 18.1, pp.129-140, 2013.

[5] Y.S. Son, Y.S. Lee, "A study on optimization techniques for the smart virtual machine platform," International Conference on Future Generation Information Technology, Springer Berlin Heidelberg, pp.167-172. 2012.

[6] 김병은, 손윤식, 이양선, "스마트 가상기계에서 코드 최적화에 대한 연구," 한국멀티미디어학회 논문지, Vol.18, No.2, pp.601-602, 2015.