IoT-Cloud 융합 가상 기계 시스템에서 정적 프로파일링을 통한 문맥 정보 추출에 대한 연구

김상수*, 손윤식**, 이양선*
*서경대학교 컴퓨터공학과
**동국대학교 컴퓨터공학과
e-mail : better4636@skuniv.ac.kr,
sonbug@dongguk.edu, yslee@skuniv.ac.kr

A Study on Context Information Extraction through Static Profiling in IoT-Cloud Fusion Virtual Machine System

Sangsu Kim*, Yunsik Son**, Yangsun Lee*
*Dept of Computer Engineering, SeoKyeong University
**Dept of Computer Science and Engineering, Dongguk University

1

요 약

IoT-Cloud 융합 가상 기계 시스템은 오프로딩 기법을 사용하여 저성능 사물인터넷 장비에서 고성능 클라우드 서버의 연산력을 제공받는다. 이 경우 오프로딩 실행 대상 프로그램은 사물인터넷 장비와 클라우드 서버의 실행환경에서 일관성이 유지되어야하기 때문에 문맥 동기화가 필요하다. 현재 문맥 동기화 방식은 전체 문맥 동기화를 시도하기 때문에 네트워크 오버헤드가 증가하여 비효율적이다.

본 논문은 오프로딩 실행에 필요한 문맥 정보만을 동기화하는 효율적인 문맥 동기화를 위해서 정적 프로파일링을 통해 오프로딩 실행 대상 작업에 동기화가 필요한 문맥 정보들을 사전에 추출하였다. 추출된 문맥 정보를 기반으로 문맥 동기화가 이뤄지면 오프로딩 실행에 필요한 문맥 정보만을 동기화 하기 때문에 네트워크 통신 오버헤드 감소를 기대할 수 있다.

1. 서론

IoT-Cloud 융합 가상 기계는 저성능 사물인터넷(IoT, Internet of Things) 장비에서 클라우드 서버의 고성능 컴퓨팅 파워를 이용하여 작업의 효율을 높이는 오프로딩(Offloading)[1] 기법을 사용한다. 오프로딩 기법을 사용하는 경우 실행 대상 프로그램은 클라이언트와 서버 사이에 일관성이 유지되어야하기 때문에 문맥 동기화가 이뤄져야 한다. 문맥 동기화 시 발생하는 네트워크 통신 오버헤드는 오프로딩 실행 성능을 감소시킬 수 있기 때문에 효율적인 오프로딩을 위해서는 오프로딩 실행에 필요한 문맥 정보만을 동기화해야 한다.

본 논문에서는 오프로딩 실행에 필요한 문맥 정보만을 동기화하는 효율적인 문맥 동기화를 위해서 정적 프로파일링을 통해 오프로딩 실행에 필요한 문맥 정보들을 추출하였다. 추출된 문맥 정보들을 기반으로 문맥 동기화가 이뤄지면 동기화되는 문맥 정보의 크기가 줄어들기 때문에 네트워크 통신 오버헤드 감소를 기대할 수 있다.

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

2. 관련연구

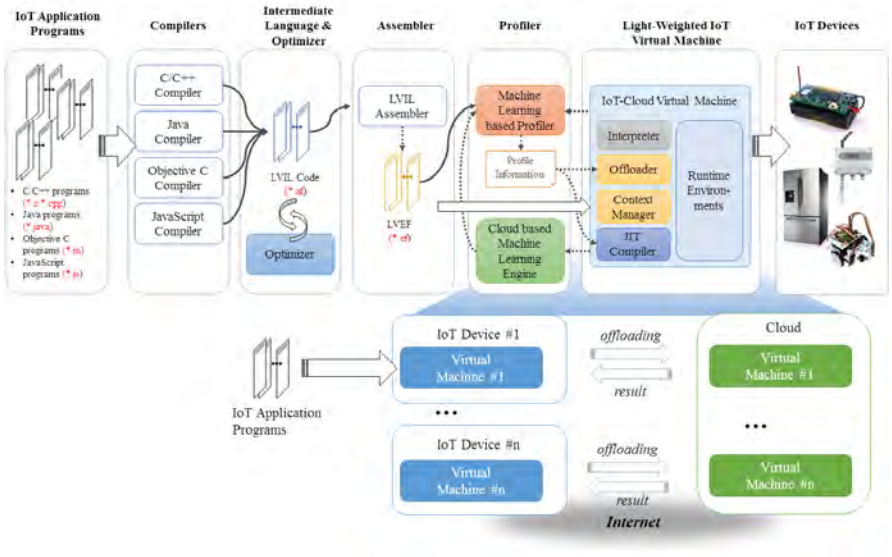
2.1 IoT-Cloud 융합 가상 기계 시스템

IoT-Cloud 융합 가상 기계 시스템[2]은 경량 가상기계, 프로파일러, 오프로딩 기법, JIT 컴파일러를 사용하여 저성능 사물인터넷 장비에 고성능 클라우드 서버의 컴퓨팅 파워를 제공하는 가상 기계 시스템이다. 이 시스템에서 오프로딩 실행 대상 프로그램은 사물인터넷 장비와 클라우드 서버의 실행환경에서 일관성이 유지되어야하기 때문에 문맥 동기화 기법을 사용한다. 문맥 동기화 시 발생하는 네트워크 통신 오버헤드에 따라 오프로딩 실행 성능이 감소할 수 있기 때문에 효율적인 문맥 동기화가 이뤄져야한다. 그림 1은 IoT-Cloud 융합 가상기계 시스템의 전체 구조도이다.

IoT-Cloud 융합 가상기계 시스템을 사용하면 기존 스마트 가상 기계의 플랫폼 독립적 환경 구축의 장점을 저성능의 사물인터넷 장비에 적용하여 다양한 프로그래밍 언어로 작성된 콘텐츠를 수용할 수 있다[3].

2.2 정적 프로파일러

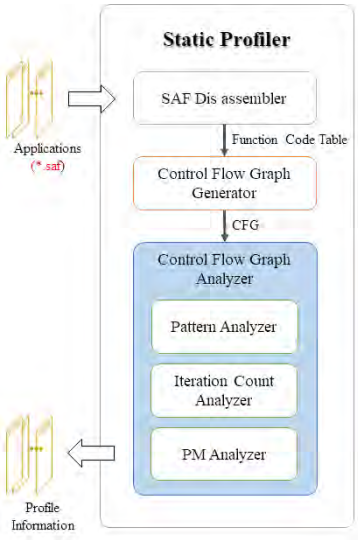
정적 프로파일러(Static Profiler)는 프로그램을 실제로 실행하지 않고 원시 응용 프로그램 또는 메타 데이터를



(그림 1) IoT-Cloud 융합 가상기계 시스템 구조도

탐색하여 응용 프로그램의 성능 및 데이터 흐름을 사전에 분석한다.

IoT-Cloud 융합 가상 기계 시스템에서 정적 프로파일러는 컴파일러의 출력파일인 SAF(Smart Assembly File)을 구성하는 중간 언어인 SIL(Smart Intermediate Language)의 분석을 통해 제어 흐름 그래프 생성기에서 블록 단위로 명령어들을 묶어 제어 흐름 그래프를 생성한다. 생성된 제어 흐름 그래프는 제어 흐름 그래프 분석기를 통해 제어 흐름 그래프가 나타내는 각 패턴을 분석하고 어플리케이션의 성능을 나타내는 복합 메트릭을 분석하여 프로파일 정보를 생성해낸다[4]. 그림 2는 정적 프로파일러의 구조를 나타낸 것이다.



(그림 2) 정적 프로파일러의 구조

3. 정적 프로파일링을 통한 문맥 정보 추출

오프로딩 실행에 필요한 문맥 정보는 실행 함수의 케이스에 따라 다르기 때문에 사전 연구에서 실행 함수의 케이스에 따라 동기화가 필요한 문맥 정보들을 분류하였다[5]. 표 1은 함수 케이스에 따른 필요한 문맥 정보들을 나타낸다.

<표 1> 함수 케이스에 따른 필요한 문맥 정보

Case \ Context		Core Context	Stack Context			Address Context		
		gPc, gProcFrameP, sp, op, DisVector, spDisplay,ArDisplay	Operation Stack	Activation Record	Constant Pool	Variable Offset	Variable Base	Variable Data Size
전달 파라미터와 복귀값이 존재하지 않는 경우	주소 값을 사용하지 않는 경우	O	X	X	X	X	X	X
		O	X	X	O	O	X	O
	주소 값을 사용하는 경우	O	X	X	O	O	O	O
		O	X	X	O	O	O	O
		O	X	X	O	O	O	O
전달 파라미터 또는 복귀값이 존재하는 경우	주소 값을 사용하지 않는 경우	O	O	X	X	X	X	X
		O	O	X	O	O	X	O
	주소 값을 사용하는 경우	O	O	O	X	O	O	O
		O	O	O	X	O	O	O
		O	O	X	O	O	O	O
		O	O	X	O	O	O	O

본 논문은 SAF의 SIL코드를 분석하여 사전 연구에서 분류한 함수 실행 케이스마다 필요한 문맥 정보들을 기반으로 정적 프로파일링을 통해 문맥 정보를 추출하는 문맥 정보 추출기를 구현하였다.

3.1 파라미터 정보 추출

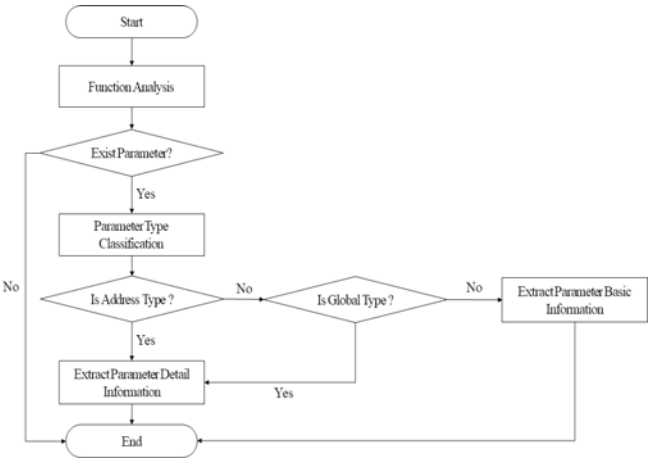
IoT-Cloud 가상 기계에서 전달 파라미터는 연산 스택에 적재되기 때문에 연산 스택 동기화 시 적재된 파라미

터의 정보가 필요하다. 파라미터 정보는 주소 파라미터, 전역 변수 파라미터, 로컬 파라미터로 구분되는 파라미터 타입에 따라 표 1의 스택 문맥 정보와 주소 문맥 정보를 포함한다.

주소 파라미터 또는 전역 변수 파라미터가 존재하지 않는 경우 연산 스택 이외의 다른 스택 정보는 동기화되지 않는다. 따라서 연산 스택 동기화에 필요한 적재된 파라미터의 개수, 복귀 값 존재 여부, 파라미터 존재 여부 정보들의 추출이 필요하다.

주소 파라미터 또는 전역 변수 파라미터가 존재하는 경우 활성화 레코드와 상수 풀이 동기화되기 때문에 파라미터의 세부 정보인 명령어 코드, 변수가 관리되는 위치 정보인 오프셋, 전역 변수와 지역 변수를 구분하기 위한 베이스 주소 정보, 활성화 레코드와 상수 풀의 동기화 범위의 지표인 변수의 데이터 크기 정보 추출이 필요하다.

본 논문의 문맥 정보 추출기는 SAF의 코드 섹션의 함수들을 분석하여 일반 파라미터와 주소 파라미터, 전역 변수 파라미터로 타입을 분류하고, 타입에 따라 필요한 파라미터 정보들을 추출한다. 그림 3은 문맥 정보 추출기의 파라미터 정보 추출 순서도이다.



(그림 3) 파라미터 정보 추출 순서도

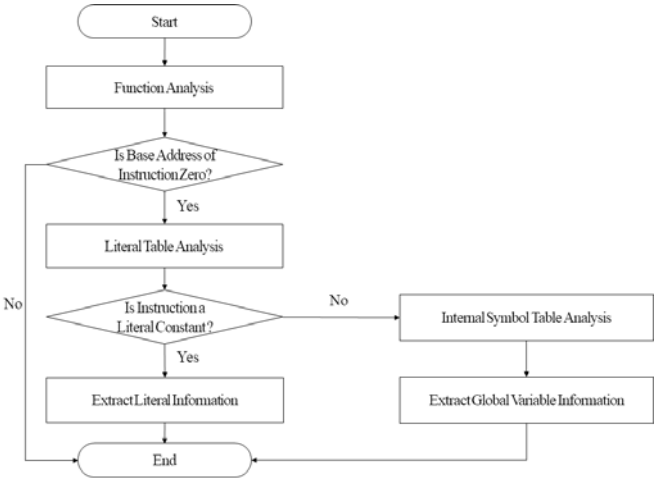
3.2 전역 변수 및 리터럴 상수 정보 추출

IoT-Cloud 가상 기계에서 리터럴 상수 및 전역 변수들은 상수 풀에서 관리되기 때문에 상수 풀 동기화 시 전역 변수와 리터럴 상수의 정보가 필요하다.

리터럴 상수는 읽기 전용 정보이기 때문에 첫 번째 오프로딩 실행 시 리터럴 상수가 관리되는 상수 풀은 한 번의 동기화가 필요하다. 리터럴 상수의 데이터 크기는 리터럴이 관리되는 상수 풀 동기화 범위를 나타낸다. 따라서 전체 리터럴 상수의 데이터 크기 정보를 추출하면 리터럴이 관리되는 상수 풀은 효율적인 동기화가 가능하다.

전역 변수는 함수 내부에서 사용이 있는 경우마다 상수 풀의 동기화가 필요하다. 명령어의 오프셋 주소는 전역 변수가 관리되는 상수 풀의 위치를 나타내고, 전역 변수의 데이터 크기는 상수 풀이 동기화되어야 할 범위를 나타낸다. 따라서 전역 변수의 오프셋 주소와 데이터 크기를 추출하면 상수 풀의 동기화가 필요한 영역 정보를 사전에 파악하여 효율적인 동기화가 가능하다.

본 논문의 문맥 정보 추출기는 SAF의 데이터 섹션을 분석하여 리터럴 상수 및 전역 변수 정보들을 추출한다. 리터럴 상수와 전역 변수는 함수 내부에서 사용될 경우 명령어의 베이스 주소가 0이다. 추출기는 코드 섹션의 함수를 분석하여 베이스 주소가 0인 명령어가 사용되는 경우 데이터 섹션의 리터럴 테이블과 내부 심볼 테이블을 분석하여 리터럴 정보와 전역 변수 정보를 추출한다. 그림 4는 전역 변수 및 리터럴 상수 정보 추출 순서도이다.



(그림 4) 리터럴 상수 및 전역 변수 추출 순서도

4. 실험 결과

본 논문에서 구현한 문맥 정보 추출기가 정상적으로 문맥 정보를 추출하는지 확인한다. 표 2는 버블 정렬 알고리즘 실행 프로그램의 SAF와 SAF 분석을 통해 문맥 정보를 추출한 결과 파일이다.

<표 2> 버블 정렬 알고리즘 실행 프로그램의 SAF 및 문맥 정보 추출 결과 파일

BubbleSort.saf

```
%%HeaderSectionStart
                                ..중략..
%%HeaderSectionEnd
%%CodeSectionStart
    %FunctionStart
        .func_name  &bubbleSort
        .opcode_start
            proc     16  1    1
            str.p    1    0
            ldc.i    0
            str.i    1    4
                                ..중략..
        %Label  ##8
            ldp
            lda 0    @2
            calls    40
            ret
        .opcode_end
    %FunctionEnd
%FunctionStart
        .func_name  &main
        .opcode_start
            proc     40  1    1
```

```
            ldp
            lda 1    0
            call     &bubbleSort
            ret
        .opcode_end
    %FunctionEnd
%%CodeSectionEnd
%%DataSectionStart
    %LiteralTableStart
        .literal_start  @0    0   14
                              ..중략..
        .literal_end
        .literal_start  @1    0    4
                              ..중략..
        .literal_end
        .literal_start  @2    0    3
                              ..중략..
        .literal_end
    %LiteralTableEnd
    %InternalSymbolTableStart
        .var_start  $length 0    4    1    1    (4)
                              ..중략..
        .var_end
    %InternalSymbolTableEnd
    %ExternalSymbolTableStart
    %ExternalSymbolTableEnd
%%DataSectionEnd
```

BubbleSort.json

```
{
  "Literal": {
    "ltr_size[0]": 14,
    "ltr_size[1]": 4,
    "ltr_size[2]": 3,
    "total_size": 21
  },
  "&bubbleSort": {
    "offset": 0,
    "paramCnt": 1,
    "exist_param": true,
    "exist_retv": false,
    "ParamInfo": {
      "call[0]": {
        "parm[0]": {
          "callerOffset": 36,
          "opCode": 39,
          "base": 1,
          "offset": 0,
          "dataSize": 40
        }
      }
    },
    "GlobalVariable": {
      "var[0]": {
        "opCode": 19,
        "offset": 24,
        "dataSize": 4
      }
    }
  },
  "&main": {
    "offset": 548,
    "paramCnt": 0,
    "exist_param": false,
    "exist_retv": false
  }
}
```

표 2를 통해 함수 실행에 필요한 파라미터 정보, 리터럴 및 전역 변수 정보들이 정상적으로 추출되었음을 확인할 수 있다.

표 3 문맥 정보 추출 시 감소되는 문맥 정보의 크기

구분 / 크기(byte)	기존 문맥 동기화	제안하는 문맥 동기화	감소량
전송 문맥 정보	144	105	39
수신 문맥 정보	120	72	48

표 3은 버블 정렬 실행 프로그램 실행 시 기존 문맥 동기화와 본 논문에서 추출한 문맥 정보를 기반으로 한 문맥 동기화 시 동기화되는 문맥 정보의 크기를 비교한 것이다.

표 3을 통해 문맥 정보 추출을 기반으로 한 문맥 동기화는 전송 문맥 정보는 약 27% 수신 문맥 정보는 약 40% 감소를 기대할 수 있다.

5. 결론 및 향후 연구

IoT-Cloud 융합 가상 기계 시스템은 오프로딩 기법을 사용하여 저성능 사물인터넷 장비에서 고성능 클라우드 서버의 연산력을 제공받는다. 이 경우 오프로딩 실행 대상 프로그램은 사물인터넷 장비와 클라우드 서버 사이에 일관성이 유지되어야하기 때문에 문맥 동기화가 이뤄져야한다. 문맥 동기화 시 발생하는 네트워크 통신 오버헤드는 오프로딩 실행 성능을 감소시킬 수 있기 때문에 효율적인 문맥 동기화가 필요하다.

본 논문은 효율적인 문맥 동기화를 위해서 정적 프로파일링을 통해 오프로딩 실행에 필요한 문맥 정보를 추출하였다. 정적프로파일링을 통해 추출된 문맥정보를 기반으로 한 문맥 동기화는 오프로딩 실행 대상 작업에 필요한 문맥 정보만 동기화를 시도하기 때문에 네트워크 통신 오버헤드 감소를 기대할 수 있다.

현재 IoT-Cloud 가상 기계의 문맥동기화는 전체 문맥 동기화를 시도하기 때문에 네트워크 통신 오버헤드가 증가하여 비효율적이다. 향후 본 연구팀은 본 논문에서 추출한 문맥 정보들을 기반으로 IoT-Cloud 가상 기계가 필요한 문맥 정보만을 동기화를 시도하도록 연구할 예정이다.

참고문헌

[1] Kumar, Karthik, “A survey of computation off-loading for mobile systems,″ Mobile Networks and Applications Vol. 18, No. 1, pp. 129-140, 2013.

[2] Yunsik Son, Yangsun Lee, ″Offloading Method for Efficient Use of Local Computational Resources in Mobile Location-Based Services Using Clouds,” Mobile Information Systems, vol. 2017, 9 pages, 2017. Netherlands Hindawi Publishing Corp.

[3] Yangsun Lee, Yunsik 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] 서동현, “저성능 IoT 디바이스의 효율적인 오프로딩을 위한 정적 프로파일러에 대한 연구,” 서경대학교 공학석사 학위논문, 2017.

[5] 김상수, 손윤식, 이양선, “IoT-Cloud 융합 가상 기계에서 효율적인 오프로딩을 위한 문맥 정보 분류에 대한 연구,” 2017년도 한국멀티미디어학회 춘계학술발표대회 논문집, Vol.20, No.1, pp.772-773, 2017.