IoT-Cloud 융합 가상 기계에서 효율적인 오프로딩 실행을 위한 문맥 동기화기의 설계 및 구현

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

Design and Implementation of Context Synchronizer for Efficient Offloading Execution in IoT-Cloud Fusion Virtual Machine

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 저성능의 사물인터넷 장비에서 고성능 클라우드 서버의 연산력을 제공받는 오프로딩 기법을 사용한다. 오프로딩 기법을 사용하는 경우 실행 대상 프로그램은 사물인터넷 장비와 클라우드 서버 사이에 일관성이 유지되어야하기 때문에 문맥 동기화가 필요하다. 기존 IoT-Cloud 융합 가상 기계의 문맥 동기화 방식은 전체 문맥 동기화를 시도하기 때문에 네트워크 오버헤드가 증가하여 비효율적이다. 네트워크 오버헤드는 오프로딩 실행 성능을 기존보다 감소시킬 수 있기 때문에 효율적인 오프로딩을 위해서는 오프로딩 실행에 필요한 문맥 정보만을 동기화하여 네트워크 오버헤드를 줄여야 한다.

본 논문에서는 효율적인 오프로딩 실행을 위해 정적 프로파일링을 통해 추출된 문맥 정보를 기반으로 오프로딩 실행에 필요한 문맥 정보만을 동기화하는 문맥 동기화기를 설계 및 구현하였다. 오프로딩 실행에 필요한 문맥 정보만 동기화가 이뤄지면 문맥 동기화 시 발생하는 네트워크 오버헤드의 크기가 줄어들기 때문에 효율적인 오프로딩 실행이 가능하다.

1. 서론

IoT-Cloud 융합 가상 기계 시스템[1]은 저성능의 사물인터넷(IoT, Internet of Things) 장비에서 고성능 클라우드 서버의 연산력을 제공받는 오프로딩(Offloading) 기법을 사용한다. 오프로딩 실행 대상 프로그램은 사물인터넷 장비와 클라우드 서버 사이에 일관성이 유지되어야하기 때문에 문맥 동기화(Context Synchronization)가 이뤄져야한다. 기존 IoT-Cloud 가상 기계 시스템의 문맥 동기화기는 전체 문맥 동기화를 시도하기 때문에 네트워크 오버헤드가 증가하는 비효율적인 구조였다. 문맥 동기화 시 발생하는 네트워크 오버헤드 비용은 오프로딩 성능을 감소시킬 수 있기 때문에 오프로딩 실행에 필요한 문맥 정보만을 동기화하여 네트워크 오버헤드의 크기를 줄이는 문맥 동기화기가 필요하다.

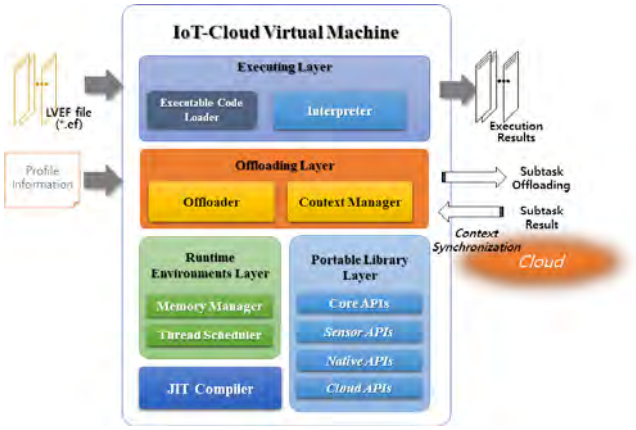
본 논문은 정적 프로파일링을 통해 추출된 문맥 정보들을 기반으로 필요한 문맥 정보들을 동기화하는 문맥 동기화기를 설계 및 구현하였다. 본 논문의 문맥 동기화기는 필요한 문맥 정보들만 동기화하기 때문에 네트워크 오버헤드 크기가 줄어 효율적으로 오프로딩을 실행할 수 있다.

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

2. 관련연구

2.1 IoT-Cloud 융합 가상 기계

IoT-Cloud 융합 가상 기계는 기존 스마트 가상기계[2,3]의 플랫폼 독립적 환경 구축의 장점을 저성능의 사물인터넷 장비에 적용하여 다양한 프로그래밍 언어로 작성된 콘텐츠를 수용할 수 있다. 또한 저성능의 사물인터넷 장비에서 높은 컴퓨팅 파워를 요구하는 작업을 고성능의 클라우드에 위임하여 실행하는 오프로딩 기법을 적용하여 사물인터넷 장비의 성능제한 없이 작업을 실행할 수 있다. 그림 1은 IoT-Cloud 융합 가상 기계의 구조를 나타낸다.

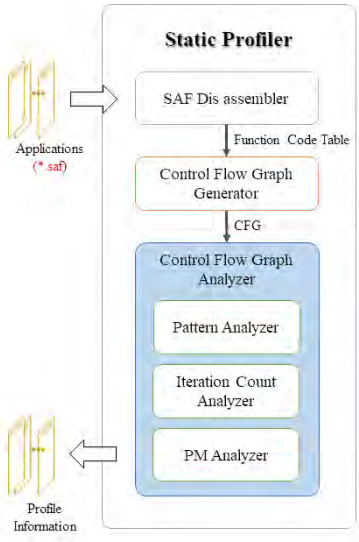


(그림 1) IoT-Cloud 가상기계의 구조

2.2 오프로딩

오프로딩 기법은 저성능의 장비에서 실행하기 어려운 높은 컴퓨팅 파워를 요구하는 작업을 고성능 클라우드 서버에 위임하여 작업의 효율을 극대화하는 클라우드 컴퓨팅 기법이다[3,4]. 오프로딩은 서버의 고성능 컴퓨팅 파워를 이용할 수 있다는 장점이 있지만, 네트워크 오버헤드 비용에 따라 오히려 성능이 감소하는 경우가 있다. 네트워크 오버헤드 비용은 동기화되는 문맥 정보 크기에 비례하기 때문에 동기화되는 문맥 정보의 크기를 최소화할 방법이 필요하다.

2.3 정적 프로파일러



(그림 2) 정적 프로파일러의 구조

정적 프로파일러(Static Profiler)는 프로그램을 실제로 실행하지 않고 원시 응용 프로그램 또는 메타 데이터를 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) 코드 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[5]. 그림 2는 정적 프로파일러의 구조를 나타낸 것이다.

3. 효율적인 오프로딩 실행을 위한 문맥 동기화기

본 논문에서 구현된 문맥 동기화기는 정적 프로파일러가 분석한 프로파일 정보를 기반으로 오프로딩 대상 함수에 필요한 문맥 정보만을 동기화한다. 함수 실행에 필요한 문맥 정보만을 동기화하면 네트워크 통신 오버헤드가 줄어들어 보다 효율적인 오프로딩 실행이 가능하다.

3.1 필수 문맥 정보 동기화기

필수 문맥 정보(Core Context Information)는 IoT-Cloud 융합 가상 기계의 인터프리터가 프로그램 실행을 위해 항상 필요한 문맥 정보들로, 명령어 버퍼, 함수의 시작 프레임을 가리키는 프레임 포인터, 각 문맥 정보들에 접근하기 위한 포인터들로 구성된다. 필수 문맥 정보들은 프로그램 실행에 항상 필요하기 때문에 오프로딩 실행 시 항상 동기화 되어야한다.

본 논문에서 구현한 필수 문맥 정보 동기화기는 사물인터넷 가상 기계의 오프로더가 오프로딩 요청을 받으면 필수 문맥 정보들을 수집하여 직렬화 과정을 실행하여 전송하고, 서버 가상 기계에서 전송받은 정보들을 역직렬화 과정을 실행하여 동기화한다.

3.2 스택 문맥 정보 동기화기

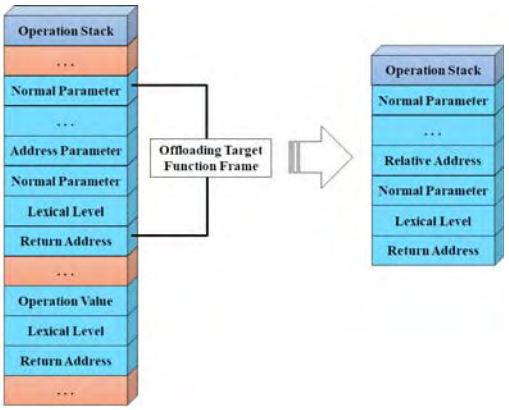
스택 문맥 정보(Stack Context Information)는 명령어 연산이 실행되는 연산 스택, 지역 변수가 관리되는 활성화 레코드, 전역 변수 및 리터럴 상수가 관리되는 상수 풀로 구성된다. 본 단락에서는 스택 문맥 정보들의 효율적인 동기화를 위해 구현된 스택 문맥 정보 동기화기의 연산 스택 동기화, 활성화 레코드 동기화, 상수 풀 동기화 방식에 대해 설명한다.

3.2.1 연산 스택 동기화

전달 파라미터 또는 복귀 값의 경우 연산 스택(Operation Stack)에 푸시 되어 전달된다. 전달 파라미터 푸시연산 이후 작업은 오프로딩 실행을 통해 서버에서 이뤄지며, 복귀 값 푸시 연산 이후 작업은 클라이언트에서 이뤄진다. 따라서 연산 스택의 동기화는 전달 파라미터 또는 복귀 값이 존재하는 경우에만 이뤄지면 된다.

본 논문에서 구현한 스택 동기화기의 연산 스택 동기화는 정적 프로파일링을 통해 추출된 파라미터 문맥 정보들을 기반으로 오프로딩 실행 대상 함수의 전달 파라미터

또는 복귀 값 정보를 사전에 판단하여 파라미터 값이 주소 값이 아닌 경우 오프로딩 실행 대상 함수의 프레임 영역을 동기화하고, 주소 값인 경우 푸시 되어있는 절대 주소 값을 상대 주소 값으로 계산한 후 프레임 영역을 동기화한다. 그림 3은 연산 스택의 동기화되는 영역을 나타낸다.



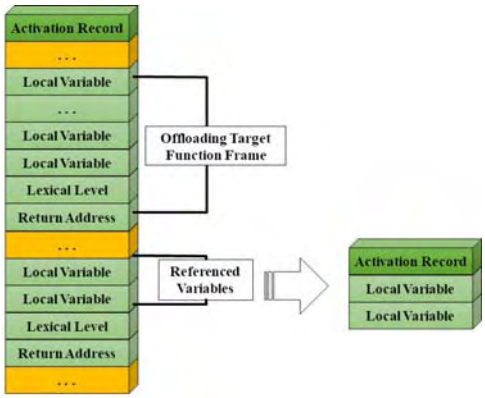
(그림 3) 연산 스택의 동기화되는 영역

3.2.2 활성화 레코드 동기화

활성화 레코드(Activation Record)는 IoT-Cloud 융합 가상 기계에서 지역 변수를 관리하는 자료 구조이다. 활성화 레코드는 값에 의한 호출의 경우 함수 내부에서 사용하는 지역 변수가 활성화 레코드의 대상 함수를 위한 프레임 영역에서 관리되고 함수 실행을 마치면 프레임 영역은 사라지기 때문에 동기화가 필요하지 않다.

참조에 의한 호출의 경우 대상 함수의 프레임 영역이 아닌 다른 프레임 영역에서 관리되는 값을 참조할 수 있기 때문에 함수 내부에서 참조하는 활성화 레코드 영역의 동기화가 필요하다.

본 논문에서 구현한 활성화 레코드 동기화는 정적 프로파일링을 통해 추출된 지역 변수들의 정보를 기반으로 오프로딩 대상 함수가 참조에 의한 호출인 경우에만 상대 주소 값 계산을 통해 활성화 레코드를 동기화한다. 그림 4는 활성화 레코드의 동기화되는 영역을 나타낸다.

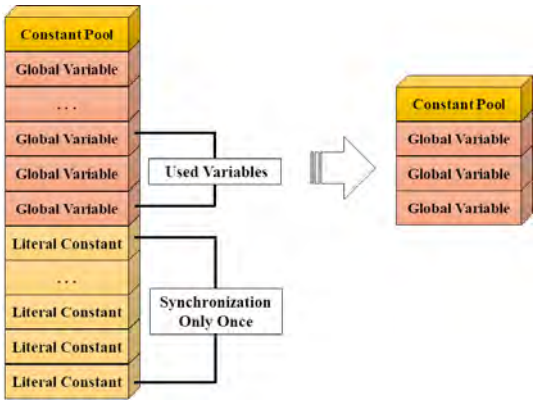


(그림 4) 활성화 레코드의 동기화되는 영역

3.2.3 상수 풀 동기화

상수 풀(Constant Pool)은 IoT-Cloud 융합 가상 기계에서 전역 변수와 리터럴 상수를 관리하는 자료구조이다. 상수 풀은 크게 리터럴 상수를 관리하는 영역과 전역 변수를 관리하는 영역으로 구분된다. 상수 풀의 전역 변수가 관리되는 영역은 오프로딩 대상 함수 내부에 전역 변수의 사용이 있는 경우에만 동기화가 필요하고, 리터럴 상수가 관리되는 영역은 리터럴 상수가 읽기 전용 정보이기 때문에 첫 번째 오프로딩 실행 시에 한 번의 동기화가 필요하다.

본 논문에서 구현한 상수 풀 동기화는 정적 프로파일링을 통해 추출된 전역 변수들의 정보와 리터럴 정보를 기반으로 전역 변수의 사용이 있는 경우에만 전역 변수가 관리되는 영역 중 필요한 영역만을 동기화하고, 첫 번째 오프로딩 실행인 경우 리터럴 상수가 관리되는 영역을 동기화한다. 그림 5은 상수 풀의 동기화되는 영역을 나타낸다.



(그림 5) 상수 풀의 동기화되는 영역

4. 실험결과 및 분석

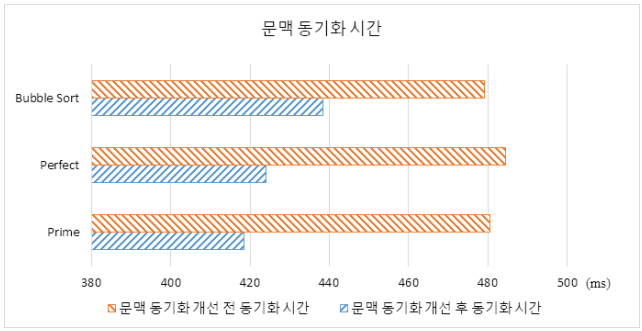
본 논문에서 구현한 문맥 동기화기를 사용했을 때 감소된 네트워크 오버헤드와 동기화되는 문맥 정보의 크기를 확인하기 위해서 기존 문맥 동기화기를 사용하는 가상 기계와 비교한다. 표 1은 문맥 동기화 비교 실험 환경을 나타낸다.

<표 1> 문맥 동기화 비교 실험 환경

Server	
Processor	Intel(R) Core(TM) i7-4770K 3.50GHz
Memory	16GB RAM
Operating System	Microsoft Windows 10

IoT Device	
Processor	BCM2837 64Bit QUAD Core 1.2GHz
Memory	1GB RAM
Operating System	Raspbian

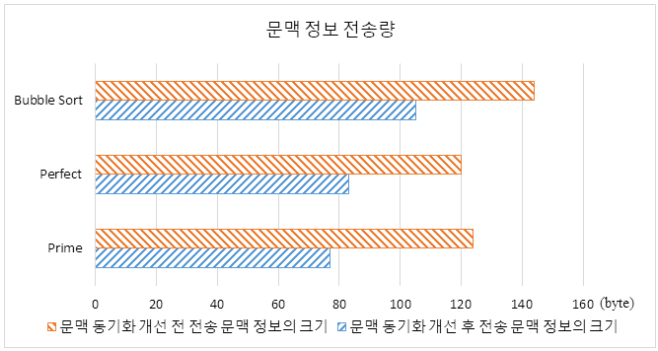
그림 6은 소수, 완전수, 버블 정렬 알고리즘의 실행을 위해 필요한 문맥 동기화 시간을 비교한 것이다.



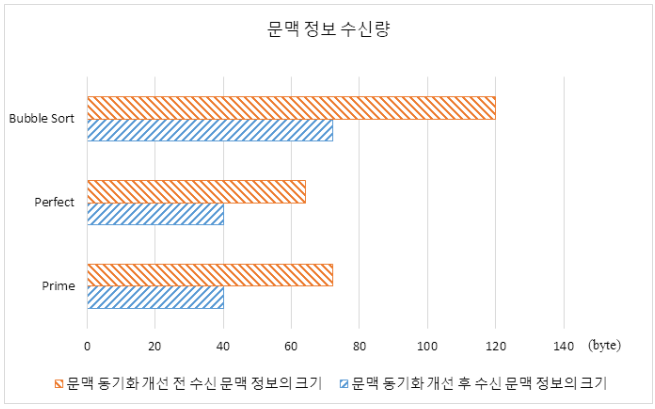
(그림 6) 소수, 완전수, 버블 정렬 알고리즘 실행을 위해 필요한 문맥 동기화 시간

그림 7의 문맥 동기화 시간은 각 알고리즘을 1000번 실행한 문맥 동기화 시간의 평균이다. 그림 7의 그래프를 통해 기존 문맥 동기화기를 사용한 경우 보다 본 논문에서 구현한 문맥 동기화기를 사용한 것이 전체 동기화 시간을 약 10% 이상 감소시켰음을 확인할 수 있다.

그림 7, 8은 소수, 완전수, 버블 정렬 알고리즘 실행을 위해 전송되는 문맥 정보와 수신되는 문맥 정보의 크기를 비교한 것이다.



(그림 7) 소수, 완전수, 버블 정렬 알고리즘 실행을 위해 전송되는 문맥 정보 전송량



(그림 8) 소수, 완전수, 버블 정렬 알고리즘 실행 후 수신하는 문맥 정보 수신량

그림 7, 8를 통해 본 논문에서 구현한 문맥 동기화기를 사용했을 시 문맥 정보들의 전송량은 약 30% 수신량은 약 40% 감소했음을 확인할 수 있다.

5. 결론 및 향후연구

본 논문에서는 문맥 정보 동기화에서 발생하는 네트워크 오버헤드 비용을 줄이기 위해서 IoT-Cloud 융합 가상 기계의 문맥 동기화기를 설계 및 구현하였다. 구현된 문맥 정보 동기화기의 사용은 오프로딩 실행에 필요한 문맥 정보만을 동기화하기 때문에 네트워크 오버헤드 비용이 감소하여 효율적인 오프로딩이 가능하다.

현재 연구 중인 IoT-Cloud 가상 기계 시스템은 정적 프로파일링을 통해 분석한 함수 복잡도가 일정 수치를 넘어가면 오프로딩이 실행되는 단순한 구조를 가진다. 오프로딩 실행은 네트워크 상황, 클라이언트의 상황, 서버의 상황 등 여러 가지 상황이 성능에 영향을 줄 수 있기 때문에 향후 본 연구팀은 머신러닝 기법을 사용하여 클라이언트가 여러 가지 상황을 고려하여 능동적으로 오프로딩 실행을 결정하도록 연구할 예정이다.

참고문헌

[1] 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.

[2] Yunsik Son, Yangsun Lee "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.

[3] Yunsik Son, Yangsun Lee "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.

[4] Kumar, Karthik, "A survey of computation offloading for mobile systems," Mobile Networks and Applications Vol. 18, No. 1, pp. 129-140, 2013.

[5] Shi, Cong, "Cosmos: computation offloading as a service for mobile devices," Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing. ACM, pp. 287-296, 2014.

[6] 서동현, 저성능 IoT 디바이스의 효율적인 오프로딩을 위한 정적 프로파일러에 대한 연구, 서경대학교 공학석사 학위논문, 2017