pISSN: 2287-5891

http://dx.doi.org/10.3745/KTCCS.2013.2.5.203

SorMob: Computation Offloading Framework based on AOP

Yeongpil Cho† · Doosan Cho†† · Yunheung Paek†††

ABSTRACT

As smartphones are rapidly and widely spread, their applications request gradually larger computation power. Recently, in the personal computer, computing power of hardware has exceeded performance requirement of software sometimes. Computing power of smartphone, however, will not grow at the same pace as demand of applications because of form factor to seek thinner devices and power limitation by relatively slow technical progress of battery. Computation offloading is getting huge attention as one of solution for the problem. It has not commonly used technology in spite of advantages for performance and power consumption since the existing offloading frameworks are difficult for application developer to utilize. This paper presents an application developer-friendly offloading framework, named SorMob. Based on Aspect Oriented Programming model, SorMob provides a convenient environment for application development, and its performance was verified by comparing with the existing offloading framework.

Keywords : Computation Offloading, AOP

SorMob: AOP 기반의 연산 오프로딩 프레임워크

조 영 필† · 조 두 산†† · 백 윤 흥†††

요 약

스마트폰이 널리 사용됨에 따라 이에 탑재되는 어플리케이션이 점차 고도화 되고 있다. 일면 하드웨어의 성능이 소프트웨어의 요구사항을 능가한 모습도 보이는 Personal Computer와는 달리 스마트폰의 경우 보다 얇은 것을 추구하는 디자인적 한계점과 여타 하드웨어에 비해 더딘 발전 속도를 보이는 배터리에 의해 저전력을 추구해야 한다는 한계점으로 인해 하드웨어의 성능이 소프트웨어의 요구사항을 충족시키지 못하는 모습이다. 이를 보완하기 위한 대표적인 기술로 연산 오프로딩이 각광받고 있다. 하지만, 확실히 성능 및 전력 소모에 있어서 이점을 가져다 준다는 연구에도 불구하고 오프로딩은 현재 널리 사용되는 기술이 아니다. 이는 기존 오프로딩 프레임워크는 어플리케이션 개발자가 사용하기에 난해한 점이 있기 때문이다. 따라서 본 연구는 어플리케이션 개발자 친화적인 오프로딩 프레임워크인 SorMob을 소개한다. SorMob은 안드로이드 상에서 동작하며, Aspect Oriented Programming 개념을 차용하여 개발자 친화적인 환경을 구축할 수 있었으며 실험을 통해 기존의 오프로딩 프레임워크에 뒤떨어지지 않는 성능을 가지고 있음을 확인할 수 있었다.

키워드 : 연산 오프로딩, AOP

1. 서 론

최근 스마트폰의 어플리케이션이 고도화 되고 있다. 점차 어플리케이션에서 요구하는 연산의 양이 스마트폰의 그것을 초과하거나, 그렇지 않더라도 상당한 전력 소모를 야기하여 스마트폰의 배터리 라이프를 줄이는 문제가 발생하고 있다. 이에 따라 전력소모를 최소화 함과 동시에 스마트폰의 연산 능력을 최대화 하기 위한 하드웨어 및 소프트웨어 차원의 노력이 지속되고 있다. 이러한 대표적인 하드웨어적 노력으로 ARM Cortex-15에서 도입한 big.LITTL [1]등이 있다. 반면 소프트웨어적 접근방법으로써 상응하는 성능/전력 요구를 충족시키기 위한 노력으로는 전통적인 서버-클라이언트 모델을 활용하거나 스마트폰을 모니터, 마우스, 키보드와 같은 기본적인 I/O장치만을 가진 thin-클라이언트로 보고 클라우드 서버에서 풍부한 리소스를 할당받아 사용하는 연구 또한 진행되고 있다. 이러한 연구들 외에 또다른 접근방식의 하나로 오프로딩이 있다. 스마트폰은 상대적으로 서버에 비해 리소스(CPU, 메모리, 네트워크 등)가 부족하기 때문에, 어플리케이션에서 검색, 얼굴인식, 3D 렌더링 등 많은

※ 본 연구는 교육과학기술부/한국과학재단 우수연구센터 육성사업(No.2012-0000470), 2012년도 정부(교육과학기술부)의 재원으로 한국과학재단의 국가지정연구실사업(No.0421-2012-0047), 한국연구재단의 기초연구사업(No.2010-0024529) IDEC, ISRC, 중소기업청에서 지원하는 2012년도 산학연 공동기술개발사업(C0019562)의 지원을 받아 수행된 것임.
† 준 회 원 : 서울대학교 전기정보공학부 박사과정
†† 정 회 원 : 순천대학교 전자공학과 조교수
††† 종신회원 : 서울대학교 전기정보공학부 교수
논문접수 : 2013년 2월 13일
수 정 일 : 1차 2013년 4월 1일
심사완료 : 2013년 4월 12일
* Corresponding Author : Doosan Cho(dscho@sunchon.ac.kr)

리소스를 요구하는 작업을 수행할 경우 이를 서버로 오프로딩 함으로써 어플리케이션의 성능 요구를 충족한다. 오프로딩 모델은 클라이언트에서 서버의 리소스를 활용한다는 점에서 전통적인 서버-클라이언트 모델과 일면 유사하지만, 두 모델은 클라이언트와 서버간 연결의 항상성 부분에서 차이를 보인다. 즉, 네트워크의 단절이나 서버의 리소스 부족과 같은 내-외부적 문제로 인해 클라이언트가 서버에 접속할 수 없게 되었을 때, 서버-클라이언트 모델의 경우 어플리케이션이 의도한 작업을 할 수 없게 되지만, 오프로딩 모델의 경우 어플리케이션이 로컬로 동작함으로써 작업을 중단하지 않을 수 있다. 이러한 차이점으로 인해 오프로딩 모델은 서버-클라이언트 모델에 비해 불안정한 네트워크에 대해 강인한 모습을 보이게 된다.

K. Kumar, et al. [2]에 따르면, 오프로딩 연구의 흐름은 1990년도 후반 오프로딩의 가능성을 확인하는 것에서 시작하여, 2000년도 이후 오프로딩 여부를 결정하는 알고리즘의 고도화, 그리고 현재는 오프로딩을 위한 인프라스트럭처 구현에 집중되고 있다. 이러한 오프로딩 연구는 크게 가상머신-수준 (Virtual Machine-수준) 과 어플리케이션-수준 (Application-수준) 오프로딩으로 구분할 수 있다. 가상머신-수준의 경우 Cloudlet [3]과 같이 OS를 이루는 가상머신 전체를 오프로딩 하는 것으로 최소 수백Mega Byte의 state가 전송되기 때문에 스마트폰의 경우처럼 네트워크 대역폭이 제한되는 경우에는 실용적이지 않다. 어플리케이션 -수준의 경우는 오프로딩하는 스택 프레임의 깊이에 따라 구현 방법이 달라지는데, 스택 프레임의 전체, 부분, 혹은 최상단만을 오프로딩 하는 경우로 나눌 수 있다. 특히 전체 프레임을 오프로딩 하는 경우는 해당 스택과 대응하는 스레드 자체를 오프로딩하는 것으로 볼 수 있기 때문에 오프로딩의 단위(granularity)는 스레드가 되며, CloneCloud [4]가 이에 해당된다. 또한 CloneCloud를 기반으로 state의 전송 오버헤드를 줄이는데 초점을 둔 기존 연구 [10]도 이 갈래에 포함된다. 반면 스택의 최상단 프레임만 오프로딩 하는 경우는 이번에 실행할 함수를 오프로딩 하는 것으로, 오프로딩의 단위는 함수가 되며, MAUI [5]가 이에 해당된다. 본 논문에서 소개하는 SorMob 또한 이에 해당된다.

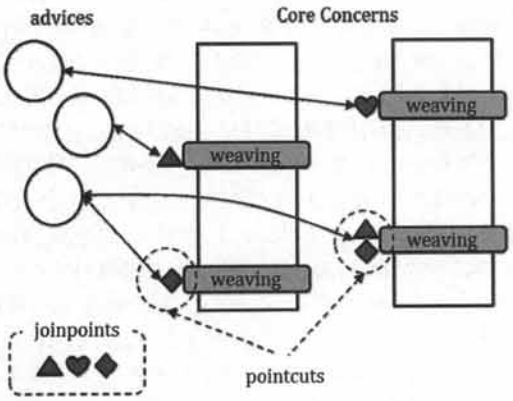

Fig. 1. AOP components

SorMob이 이처럼 스택의 최상단 프레임만 오프로딩 하는 가장 큰 이유는, SorMob이 안드로이드를 대상으로 하기 때문이다. 안드로이드는 자바를 기반으로 하기 때문에 어플리케이션 개발자가 소스코드 레벨에서 스택에 접근할 수 있는 방법이 원칙적으로 없기 때문에 CloneCloud과 같이 전체 프레임을 오프로딩 하는 경우에는 달빅 가상머신(Dalvik Virtual Machine)을 수정함으로써 이러한 문제를 극복했다. 하지만 오프로딩을 하기 위해서 해당 스마트폰에 탑재된 안드로이드 자체를 수정해야 한다는 것은 실용성에 큰 단점이 된다. 반면 R. Ma, et al. [6]는 달빅 가상머신 수정 없이 어플리케이션 레벨에서 스택의 부분 프레임을 오프로딩 하는 방법을 제안한다. 그러나 이를 위해선 어플리케이션 개발자가 스택 프레임을 수집하고 복구하기 위한 코드를 일일이 추가해야 한다는 문제가 있다.

SorMob은 MAUI와 같은 기존의 함수 단위오프로딩 프레임워크와 비교했을 때 어플리케이션 개발자 친화적이라는 장점이 있다. 지난 수년간 오프로딩 연구가 지속되어 왔음에도 불구하고 오프로딩 모델이 널리 사용되지 않고 있는 가장 큰 이유 중 하나는 기존 오프로딩 프레임워크가 사용하기 어렵거나 혹은 별도로 고안된 가상머신을 기반으로 하는 등 지나치게 커스터마이징 되어있기 때문이다. 반면 SorMob은 관점지향프로그래밍 (Aspect Oriented Programming, AOP)을 바탕으로 기존에 작성된 안드로이드 어플리케이션도 소스코드만 있다면, SorMob 라이브러리의 추가와 간단한 annotation의 추가만으로 오프로딩이 가능하도록 한다. 본 논문에서는 AOP를 바탕으로 구현된 SorMob의 구조 및 성능을 소개하고자 한다. 이를 위해 2장에서는 AOP에 대해 알아보고, 3장에서는 SorMob의 전체적인 구조 및 각 컴포넌트에 대해 알아본다. 4장에서는 SorMob의 성능에 대해 살펴보고 5장에서 마무리를 하겠다.

2. Aspect Oriented Programming

G. Kiczales, et al. [7] 에 의해 소개된 AOP는 최근 많은 관심을 받고 있다. AOP는 그 자체로 독자적인 프레임워크를 구축한다기 보다는 기존의 객체지향프로그래밍 (Object Oriented Programming, OOP) 를 바탕으로 작성된 프로그램에 적용하는 형식을 취한다. AOP의 개념은 어떠한 동작을 관점(aspect)이라 불리는 단위 모듈에 적용하는 것을 골자로 한다. 이때 동작은 충고(advice)라 불리며 관점을 표현하기 위해 결합점(joinpoint)와 교차점(pointcut)이 사용되고 충고가 결합점과 교차점에 의해 지정된 지점에 삽입되는 것을 직조(weaving)혹은 횡단(crosscutting)이라 한다. 일반적으로 충고는 실행 코드로 표현되고 결합점은 메소드의 실행 전 또는 후가 된다. 교차점은 결합점의 조합으로 표현되며 충고가 교차점의 위치에 적용되는 직조는 컴파일타임, 로딩타임, 런타임에 수행된다. 이를 예를 들어 소개하면, DataBase 클래스가 있고, 데이터 입출력을 위한 두 메소드 getData, putData가 있다고 하자. 개발자는 각 DataBase의

사용 패턴을 분석하기 위한 관점으로 메소드의 실행 전(결합점)과 실행 후(결합점)에 분석 코드(충고)를 삽입하고자 한다. 다만, 개발자는 모든 메소드에 분석 코드를 삽입하지 않고, getData의 실행 전후(교차점)와, putData의 실행 전(교차점)에만 분석 코드를 삽입하고자 한다. 개발자가 이러한 AOP를 위한 설명(description) 작성을 완료한 후, 설명에 맞춰 코드를 직조하는 시점을 컴파일 타임으로 설정하면, DataBase 클래스의 코드가 컴파일 될 때 pointcut에 따라 getData와 putData 메소드의 전후에 분석코드가 자동으로 삽입되게 된다.

이러한 AOP를 개념적으로 보지 않고 기능적으로 보면 결국 코드 삽입(code injection)이라고 할 수 있다. 개발자는 이를 활용해 프로그램의 testability를 향상 시키거나 프로그램의 동작을 감시하는 등의 작업을 할 수 있다. 뿐만 아니라 이를 오프로딩에 활용할 경우 여타 오프로딩 프레임워크와 달리 오프로딩에 대한 접근성을 향상시켜 개발자의 오프로딩 활용빈도를 향상시킬 수 있다. 이는 AOP의 명료한 개념과 손쉬운 구현에 기인한다. 자바에서 AOP를 구현하기 위한 도구로는 직접적인 AOP 도구는 아니지만, 코드 삽입을 통해 AOP를 간접적으로 구현할 수 있는 자바 프록시(proxy)를 비롯해 바이트 코드 조작(bytecode instrument)도구인 ASM, Javassist를 사용하는 방법들이 있다. 반면에 직접적인 AOP 도구인 Spring AOP나 AspectJ [8]를 사용할 수도 있는데, Spring AOP는 Spring 프레임워크를 기반으로 한 어플리케이션에서만 동작하기 때문에 SorMob에서는 범용성을 위해 AspectJ를 사용했다. 그 이유는 AspectJ가 eclipse 프로젝트로써 안드로이드 빌드환경에서 잘 동작하고, 라이브러리 형식이기 때문에 기 작성된 안드로이드 어플리케이션에 간단히 적용할 수 있다는 장점을 가지고 있기 때문이다. 다만, 안드로이드가 자바를 메인 언어로 사용할지라도, 자바의 바이트코드를 바로 사용하지 않고 이를 패키징한 형태인 달빅 바이트코드를 사용하기 때문에, 안드로이드 환경에서 AspectJ의 모든 기능을 사용할 수는 없다. 대표적으로 AspectJ는 안드로이드 상에서 컴파일타임 직조만 사용가능하며 로딩타임과 런타임 직조는 자바와 안드로이드가 사용하는 바이트코드 구조의 불일치로 인해 사용이 불가능하다.

3. SorMob

SorMob은 어플리케이션 개발자 친화적인 오프로딩 프레임워크를 추구한다. SorMob은 라이브러리 형태로 배포되기 때문에 새로 작성하는 프로그램에 쉽게 적용 가능하고, 기존에 작성된 프로그램이라 하더라도 소스코드만 있다면 손쉽게 적용할 수 있다는 점이 특징이다. SorMob이 이러한 장점을 가지는 이유는 AOP를 기반으로 개발되었기 때문이다. 달빅 가상머신의 수정 없이 어플리케이션에 오프로딩을 적용하기 위해선 필연적으로 어플리케이션이 수정되어야 한다. 이는 다시 말해 오프로딩을 위한 코드들이 어플리케이션에 삽입되어야 한다는 뜻이 된다. 그러나 이러한 작업은 어플리케이션 개발자에게 매우 불편한 작업이며, SorMob은 AOP를 통해 이를 자동화 할 수 있었다.

3.1 Components and processing flow

Fig. 2는 SorMob의 컴포넌트와 그것들의 동작 흐름을 보여준다. 각각의 컴포넌트들은 Fig. 2에 표시된 번호 순서대로 상호작용 및 동작한다. 자세한 동작 순서는 다음과 같다.

(1) 프로그램이 실행되는 도중 사전에 annotate된 메소드가 발견되면 Migrator로 제어권이 넘어간다. 이렇게 동작 하는 것은 SorMob에서 오프로딩 하려는 목표 메소드를 사전에 지정된 annotation을 통해 구별하기 때문인데, 보다 자세한 사항은 3.2절에 나타나 있다.
(2) Migrator는 Profiler를 통해 해당 목표 메소드에 관련된 프로파일 정보를 업데이트 한다.
(3) Migrator는 Solver를 통해 오프로딩 여부를 결정하며 이때 Solver는 프로파일 정보를 참조한다

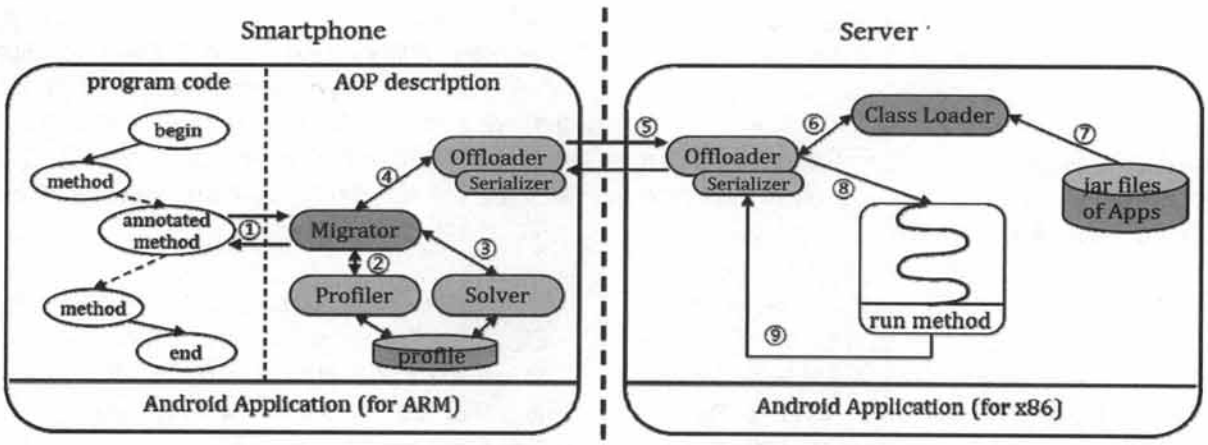

Fig. 2. SorMob framework and processing flow

(4) 오프로딩을 하기로 결정되면, Migrator는 목표 메소드를 오프로딩 하기 위한 state를 수집하여 Offloader로 전달한다.
(5) Offloader로 전달된 state들은 Serializer를 통해 직렬화 되어 서버로 전달된다.
(6) 서버측 Offloader는 전달된 데이터는 역직렬화한 후, ClassLoader를 통해 목표 메소드를 로드한다.
(7) 대부분의 경우 목표 메소드는 아직 로드되어 있지 않기 때문에 ClassLoader는 목표 메소드의 정보를 목표 메소드를 포함한 jar파일에서 로드하게 된다.
(8) 로드된 목표 메소드는 클라이언트로부터 전달된 state를 기반으로 실행된다.
(9) 목표 메소드의 수행이 끝나면, 해당 결과를 클라이언트로 전달해야 한다. 이때 수행 결과를 포함하는 모든 state를 전송하지 않고 클라이언트로부터 전달된 state 중 변경된 state만을 수집하여 전송함으로써 네트워크 부하를 최소화 한다.

이러한 동작 과정에 사용되는 컴포넌트들은 다음의 역할을 수행한다.

Migrator AOP의 충고 코드 이다. Profiler와 Solver을 통해 목표 메소드의 오프로딩 여부를 결정하며, Offloader를 통해 오프로딩을 수행한다. 네트워크 단절 혹은 예외(exception) 발생 등으로 인해 오프로딩이 중단될 경우 목표 메소드를 로컬로 수행한다.

Profiler, profile 오프로딩 여부를 결정하기 위한 프로파일 데이터를 수집하고 보관한다. Profiler는 Migrator의 도움을 받아 목표 메소드에 대한 정보를 실시간으로 수집하며 오프로딩을 결정하기 위한 중요 요소의 하나인 Round Trip Time (RTT) 를 비롯한 네트워크의 상태는 목표 메소드의 호출과 관계없이 주기적으로 확인한다. Profile에 수집되는 데이터에는 목표 메소드들의 실행시간, 빈도, 오프로딩 시 필요한 state의 크기, RTT, 네트워크 대역폭이 있다.

Solver profile 데이터를 바탕으로 목표 메소드를 오프로딩 했을 때, 네트워크 오버헤드를 고려하여 이득을 취할 수 있을지 판단한다.

Offloader, Serializer 클라이언트의 Offloader는 목표 메소드를 서버에서 실행하기 위한 state를 Serilizaer를 이용해 직렬화 하여 서버로 전송한다. 전송하는 state는 목표 메소드의 클래스 및 오브젝트 정보, 아규먼트의 클래스 및 오브젝트 정보, 그리고 목표 오브젝트의 이름이다. 기본적으로 SorMob은 Serializaer로써 자바에 포함된 Object-InputStream, ObjectOutputStream을 사용한다. 하지만 이들 클래스는 직렬화를 하기 위해 목표 클래스에 Fig. 3의 GAME 클래스와 같이 Seriali-zable 인터페이스를 구현(implement)해야 하는 번거로움이 있다. 그렇기 때문에

```
public class MainActivity {

  public void onClick(View v) {
    GAME g = new GAME();
    g.doGame("attack");
    ....
  }
}


public class GAME implements Serializable {
  Object [] enemies;

  @Migratable  <------------------  offloading할
  public Object doGame(String cmd) {  method에 tagging
    ....
  }
}
```

Fig. 3. A usage example of SorMob

SorMob은 해당 Serializer 외에도 오픈소스 자바 Serializer인 Kryo [9]을 사용하여 보다 손쉬운 직렬화 기능을 지공한다. 서버 Offloader는 직렬화된 state를 Seriali-zer를 사용해 해석하게 된다. 목표 메소드의 실행이 끝나면, 서버 Offloader는 클라이언트의 Off-loader와 같이 state를 수집하여 전송하는데, 이때 전송 오버헤드를 줄이기 위해 전체 state가 아닌 변경된 state만을 수집하여 클라이언트로 전송한다. SorMob에서는 목표 메소드의 실행 전 state 와 목표 메소드의 실행 후 state를 직렬화한 결과물인 두 바이트 배열의 차이점을 Approximate String Matching 알고리즘을 통해 구하였고, 이를 patch 데이터로 사용하였다. 클라이언트의 Off-loader는 patcher를 사용해 해당 patch 데이터를 현재 어플리케이션에 적용한다.

ClassLoader 서버에서 목표 메소드를 실행하기 위한 클래스들을 로드하고, 목표 메소드를 로드한다.

Jar files of apps 서버에서 목표 메소드를 실행하기 위해선 해당 메소드를 실행하기 위해 필요한 클래스 들의 바이트코드 정보가 필수적이기 때문에 이들 바이트코드를 서버에 모아둔다.

3.2 Developing

SorMob을 개발에 사용하기 위해선 안드로이드 SDK가 설치된 eclipse가 필요하며, 개발 후 오프로딩의 테스트 및 실제 사용을 위해선 안드로이드가 설치된 서버가 필요하다. 이때 서버는 ARM기반과 x86기반 모두 가능하다. 완성된, 혹은 개발중인 안드로이드 프로젝트에 SorMob을 적용하는 절차는 다음과 같다.

1 - AspectJ eclipse 플러그인 설치
2 - 프로젝트에 SorMob 라이브러리 추가
3 - 오프로딩 목표 메소드에 annotation 추가
4 - 목표 메소드의 클래스를 jar 파일로 export
5 - 추출한 jar 파일을 서버에 저장
6 - 서버에 SorMob의 서버 어플리케이션을 설치

세 번째 절차에서 오프로딩 할 메소드에 annotation을 추가하는 것은 Fig. 3에 나타나 있다. 목표 메소드의 정의(Definition) 위에 annotation '@Migratable'을 붙이면, 해당 메소드가 호출될 때 자동으로 Migrator로 전달됨으로써 오프로딩 프로세스가 실행된다. 네 번째 절차에서 서버에 저장하기 위해 클래스 들의 바이트코드들을 jar 파일로 추출(export)하는데, 이때 해당 클래스는 AspectJ에 의해 직조되기 이전의 코드를 가지고 있어야 한다. 왜냐하면 직조를 통해 바이트코드의 Constant Pool이 변경 될 경우, 서버의 ClassLoader에서 목표 메소드 및 연관 클래스들을 로드 할 수 없게 되기 때문이다.

4. Experiment

SorMob이 기존에 구현한 오프로딩 프레임워크와 비교에 어느 성능차이를 보이는지 비교 해본다. 비교 대상은 기존 연구 [10]를 위해 개발한 프레임워크로써, CloneCloud과 유사하지만 낮은 전송 오버헤드를 가진 프레임워크이다. 비교 실험에는 해당 프레임워크를 직접적으로 사용하는 대신 스택의 최상단 프레임만을 오프로딩하도록 수정한 프레임워크를 사용하였다. 실험환경으로 클라이언트는 듀얼코어 1.2 GHz CPU, 1 GB의 RAM을 가진 스마트폰을 준비하였고, 서버로는 쿼드코어 3.1 GHz CPU, 8 GB의 RAM을 셋팅하였다. 스마트폰의 OS는 안드로이드 4.03이며 서버의 OS는 우분투 11.10에 Virtual Box를 통해 안드로이드 4.03 x86로 설정하였다.

실험에는 과학적 벤치마크인 Bubble-sort, N-Queen, Fast Fourier Transform (FFT) 어플리케이션을 사용하였다. SorMob은 메소드 오프로딩이고, 스택의 최상단 프레임만 오프로딩 하는 것도 결과적으로 메소드 오프로딩과 동일함으로 두 오프로딩 프레임워크의 성능은 Fig. 4에 나타나 있는 것과 같이 유사하다. 게다가 Bubble-sort와 N-Queen에서는 SorMob이 최상단 프레임 오프로딩에 비해 평균적으로 27% 향상된 성능을 보여준다. 이러한 성능 차이는 일반적으로 자바 소스코드 레벨에서 오프로딩을 하는 SorMob이 달빅 가상머신 레벨, 즉 Native-C 코드 수준에서 오프로딩을 하는 최상단 프레임 오프로딩에 비해 느릴 것이라는 예상을 벗어난다. 오히려 SorMob의 경우 자바 자체가 Native-C에 비해 느린 것 외에도 AOP에 의한 오버헤드까지 추가된다. 하지만 Fig. 5에서 확인할 수 있는 것처럼 SorMob이 최상단 프레임 오프로딩에 비해 송수신 하는 state의 크기가 작기 때문에 전체적으로 보다 나은 성능을 보이게 된다. 이처럼 오프로딩을 하기 위해 송수신하는 state의 크기가 차이가 나는 것은 state를 수집하는 위치의 차이에 기인한다. 자바 수준에서 동작하는 SorMob과 Native-C 수준에서 동작하는 최상단 프레임 오프로딩 모두 state를 자동으로 수집을 한다. 그런데 최상단 프레임 오프로딩의 경우 달빅-가상머신이 구현된 Native-C 수준에서 state를 수집하기 때문에 SorMob에서는 수집하지 않는 state 또한 수집하게 된다. 대표적인 state로는 달빅-가상머신에서 자바의 클래스를 정의하기 위한 데이터가 있다. 이러한 이유로최상단 프레임 오프로딩은 SorMob에 비해 많은 state를 송수신 하게 된다. 다만, 클라이언트에서 서버로 송신하는 state의 경우 모든 벤치마크에서 SorMob이 더 작다는 것을 확인할 수 있지만, 수신하는 state의 경우에는 Bubble-sort와 FFT에서 SorMob이 오히려 더 크다는 것을 확인할 수 있다. 이러한 차이가 발생하는 것은 서버의 수신 state의 크기를 줄이기 위해 모든 state를 전송하는 대신 변경된 state만을 전송하는 방식에 있어 최상단 프레임 오프로딩에서 사용한 것이 더 효율적이라는 점에서 기인한다. SorMob의 경우에는 3.1절에 나타나 있는 바와 같이 직렬화된 두 state에 Approximate String Matching 알고리즘을 통해 변경된 state를 찾았지만, 최상단 프레임 오프로딩은 각 자바 오브젝트의 멤버 필드들을 순차적으로 비교함으로써 변경된 state를 찾았다. 더 효율적임에도 불구하고 SorMob에서 이러한 방법을 도입하지 않은 것은 자바 수준에서 각 오브젝트의 멤버 필드들을 순차적으로 비교하기 위해 사용하는 자바 리플렉션(Reflection)의 오버헤드가 상당하기 때문이다.

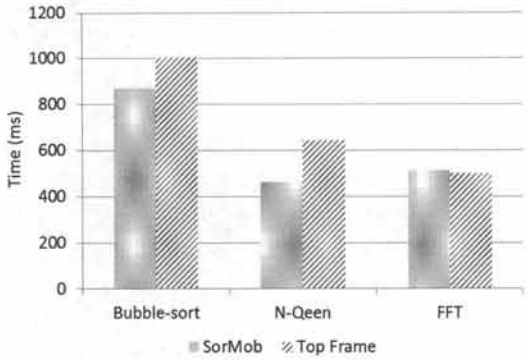

Fig. 4. Offloading time

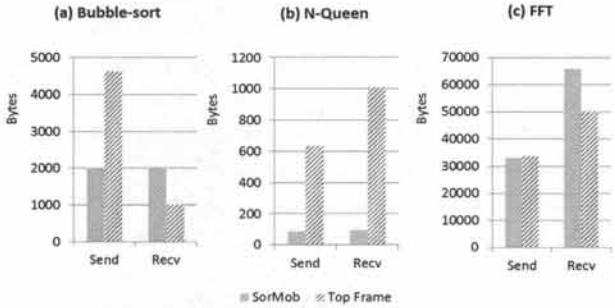

Fig. 5. Size of sent and received state from client to server

5. Conclusion

스마트폰의 부족한 리소스, 특히 연산 리소스의 부족을 보완하기 위한 오프로딩 연구는 많은 진전을 이루었다. 하지만, 오프로딩 자체의 효율성에 집중을 하다 보니 오프로

딩 프레임워크 개발자 외에 일반 어플리케이션 개발자의 경우 해당 오프로딩 프레임워크를 사용하는데 많은 애로사항을 겪게 되었다. 이에 본 논문은 이러한 점을 개선하기 위해 AOP 개념을 도입하여 개발자 친화적인 오프로딩 프레임워크인 SorMob을 개발하였다. 어플리케이션 개발자는 기존의 안드로이드 프로젝트나 혹은 새로운 프로젝트에 간단한 작업만으로 SorMob을 사용해 오프로딩 환경을 구축할 수 있었으며 실험을 통해 SorMob은 기존의 오프로딩 프레임워크에 비해 뒤쳐지지 않는 성능을 보임을 확인하였다.

참 고 문 헌

[1] Greenhalgh, Peter. "Big. LITTLE Processing with ARM Cortex™-A15 & Cortex-A7, Improving Energy Efficiency in High-Performance Mobile Platforms." *white paper, ARM September* (2011).

[2] Kumar, Karthik, et al. "A Survey of Computation Offloading for Mobile Systems." *Mobile Networks and Applications* (2012): 1-12.

[3] Ibrahim, Shadi, et al. "CLOUDLET: towards mapreduce implementation on virtual machines." *Proceedings of the 18th ACM international symposium on High perfor-mance distributed computing*. ACM, 2009.

[4] Chun, Byung-Gon, et al. "Clonecloud: elastic execution between mobile device and cloud." *Proceedings of the sixth conference on Computer systems*. 2011.

[5] Cuervo, Eduardo, et al. "MAUI: making smartphones last longer with code offload." *Proceedings of the 8th international conference on Mobile systems, appli-cations, and services*. ACM, 2010.

[6] Ma, Ricky KK, and Cho-Li Wang. "Lightweight Application-Level Task Migration for Mobile Cloud Computing." *Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on*. IEEE, 2012.

[7] Kiczales, Gregor, et al. "Aspect-oriented programming." *ECOOP'97-Object-Oriented Programming* (1997): 220-242.

[8] Laddad, Ramnivas. *AspectJ in action: practical aspect-oriented programming*. Vol.512, Manning, 2003.

[9] http://code.google.com/p/kryo/.

[10] Sengjun, Yang, et at. "Fast Dynamic Execution Offload-ing for Efficient Mobile Cloud Computing" in Pro-ceedings of the Eleventh Annual IEEE International Conference on Pervasive Computing and Com-munication (PerCom)

조 영 필

e-mail : ypcho@sor.snu.ac.kr

2010년 포항공과대학교 전자전기공학부 (학사)

2011년~현 재 서울대학교 전기정보공학부 박사과정

관심분야 : 모바일 클라우드 컴퓨팅, 임베디드 시스템

조 두 산

e-mail : dscho@sunchon.ac.kr

2001년 한국외국어대학교 전자제어공학과 (학사)

2003년 고려대학교 전기전자공학과(석사)

2009년 서울대학교 전기컴퓨터공학부(박사)

2008년~2010년 ㈜ 리코어스 이사

2010년~현 재 순천대학교 전자공학과 조교수

관심분야 : 임베디드 시스템 설계/최적화, 메모리 시스템 최적화, 컴파일러, 병렬분산처리

백 윤 흥

e-mail : ypaek@snu.ac.kr

1988년 서울대학교 컴퓨터공학과(학사)

1990년 서울대학교 컴퓨터공학과(석사)

1997년 UIUC 전산과학(박사)

1997년~1999년 NJIT 조교수

1999년~2003년 KAIST 전자전산학교 부교수

2003년~현 재 서울대학교 전기정보공학부 교수

관심분야 : 임베디드 소프트웨어, 컴파일러, MPSoC, CGRA