클라우드를 이용한 모바일 오프로딩 시스템 연구*

박세훈, 최찬호, 엄현상, 염헌영
서울대학교 컴퓨터공학과
e-mail : {tshpark, chchoi, hseom, yeom}@dcslab.snu.ac.kr

A Study on Mobile Offloading System in Cloud Computing

Sehoon Park, Chanho Choi, Heonsang Eom, Heon Y. Yeom
Dept. of Computer Engineering, Seoul National University

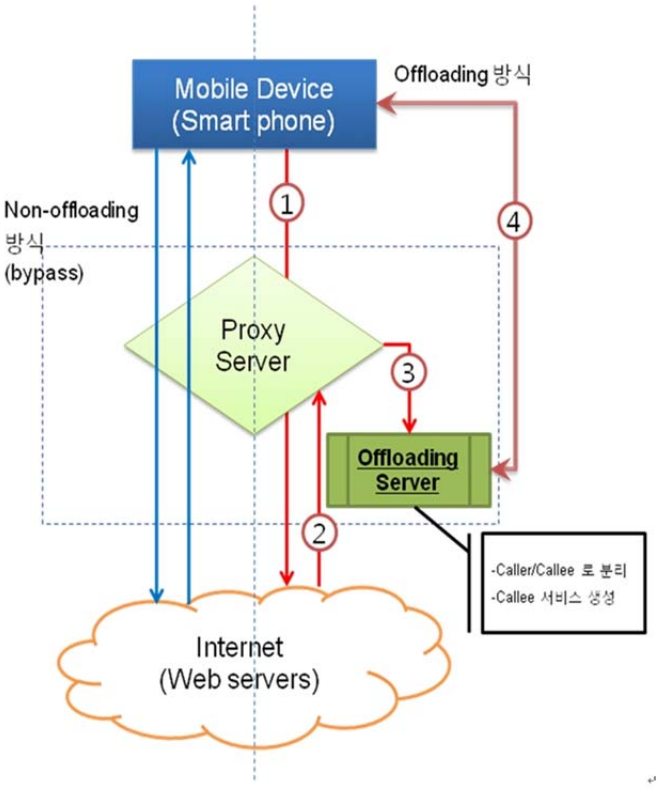
요 약

모바일 디바이스의 사용이 폭발적으로 증가하고 있고, 특히 스마트폰을 위한 많은 어플리케이션이 소개 되고 있지만, 여전히 스마트폰이 가지는 낮은 하드웨어 성능과 제한적인 배터리용량은 PC 환경을 대체하기가 어렵다. 본 논문에서는 클라우드가 가지는 서버 환경을 통해서 모바일 디바이스의 특정 태스크를 서버 사이드로 오프로딩 하여 실행하고, 단말은 그 결과만 전달받는 방식의 모바일 오프로딩 시스템을 제안한다. 실험을 통해서 모바일 오프로딩의 방법이 어플리케이션의 응답성을 높일 수 있음을 확인하였다. 또한 제시된 오프로딩 기법을 통해서 단말의 CPU utilization 을 줄이고, 따라서 단말의 소모전력을 최대 70%까지 줄일 수 있었다.

1. 서론

스마트폰의 탄생은 데스크톱 중심의 개인 컴퓨터 환경을 여러 가지로 바꾸어 놓고 있으며, 스마트폰의 보급률은 2015 년에 전세계 인구의 기준으로 50%까지 올라갈 것이라고 한다. 이미 2011 년에 스마트폰의 판매량은 데스크톱 PC 의 판매량을 뛰어 넘었다. 대표적인 사이트인 애플 스토어와 구글 플레이에 등록되는 스마트폰의 앱의 개수는 하루 최대 1,500 개에 달하고 있으며, 종래 데스크톱 에서 사용되던 많은 어플리케이션 및 웹 프로그램이 점점 스마트폰 중심으로 옮겨지고 있는 추세이다. 하지만 여전히 데스크톱 PC 에 비해 낮은 스마트폰의 하드웨어 성능과 배터리를 사용하는 제한된 전력공급은 모바일 디바이스가 가지는 가장 큰 제약사항이다. 본 논문에서는 클라우드 상의 오프로딩 서버들을 통해서 단말의 성능을 분산 처리하는 모바일 오프로딩에 대한 연구를 제안하고자 한다. 복잡한 계산을 요구하는 어플리케이션이나 [1, 2] 이미지 프로세싱 및 비디오 디코딩 같이 워크로드가 높은 어플리케이션들의 태스크를 클라우드 상의 서버들에게 오프로딩 하여 처리하고 모바일에서는 처리 결과만을 전달받고 UI (User Interface) 처리하는 방식을 말한다. 이러한 오프로딩 기법은 이미 알려진 것이 [3, 4] 있지만 본 논문은 기존 연구와는 조금 다르게 다양한 플랫폼을 가지고 있는 스마트폰의 OS 들에게 범용적으로 적용될 수 있는 웹 환경의 어플리케이션을 그 대상으로 하고 있다. 오프로딩의 적용여부를 선택적으로 판단하기 위해서 코드에 오프로딩을 위한 특수한 주석을 삽입 함으로써 오프로딩 감지의 오버헤드를 줄이고, 쉽게 구현할 수 있도록 하였다. 본 논문에 제안된 오프로딩 방식을 통해서 우리는 실험에 사용된 어플리케이션의 응답시간을 획기적으로 개선할 수 있었으며, 모바일 단말의 CPU 사용량을 낮추고, 그로 인한 모바일 디바이스의 배터리 전력소모를 최대 74%까지 낮출 수 있었다.

2. 모바일 오프로딩 시스템



(그림 1) 모바일 오프로딩 아키텍처

일반적으로 모바일에서 웹 어플리케이션 및 웹 리

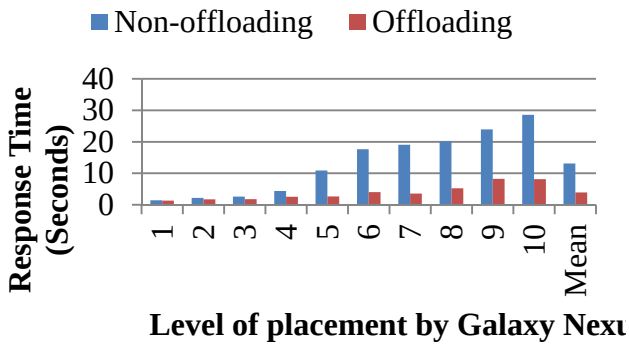
소스를 실행하기 위해서는 리소스가 위치한 외부 웹 서버에 접속하여 관련 코드를 단말에 내려 받은 이후, 단말에서 해당 리소스를 Parsing 하여 어플리케이션을 실행한다. 하지만, 워크로드가 높은 작업이나, CPU intensive 한 작업이 많은 경우에 모바일 환경의 특성상 다른 서비스에 영향을 미치고 과도한 전력소모를 동반할 수 있다. 본 논문에서는 모바일 환경에서의 오프로딩을 위해서 RMI (Remote Method Invocation) 방식을 이용한 클라이언트의 stub 과 서버사이드의 skeleton 으로 나누어 진다. 그림 1 에서 표시된 것처럼 본 모바일 디바이스와 웹 서버 사이에 built-in Proxy 서버가 위치하고 있으며, Proxy 서버는 리소스가 오프로딩이 필요하다고 판단되면 해당 어플리케이션을 위한 독립적인 skeleton 코드를 생성하고, Nodejs [4] 를 이용하여 해당 서비스를 실행한다. 오프로딩이 필요 없는 경우는 그림 1 의 왼쪽 두 개의 화살표처럼 프록시에서는 웹 요청과 응답을 단순히 bypass 하게 되고, 오프로딩이 감지된 경우 (오른쪽) Proxy 서버는 오프로딩 서비스를 생성하고 이후 모바일 단말은 오프로딩 서버와 직접적으로(④) 통신하게 된다. 이러한 경우 모바일 클라이언트는 더 이상 해당 리소스 및 어플리케이션을 직접 실행할 필요 없이 stub 으로 구현된 함수만 호출하고, 실제 작업은 클라우드 상의 특정 built-in 프록시에 의해 생성된 오프로딩 서버에서 실행이 진행된다.

오프로딩의 판단 기준은 다양한 모바일 디바이스에 범용으로 사용될 수 있기 위해서 웹 리소스를 중점으로 구현하였고, 웹 리소스의 상단에 특정한 주석을 삽입함으로써 해당 리소스 및 어플리케이션이 오프로딩을 지원하는 웹 어플리케이션임을 판단할 수 있도록 하였다. 다양한 스마트폰 운영체제가 사용되고 있는 현재의 시점에서 공통적으로 적용이 가능하고, 플랫폼에 상관없는 웹 환경에서의 오프로딩은 그 효율성이 매우 높다고 할 수 있다.

3. 실험환경 및 구현결과

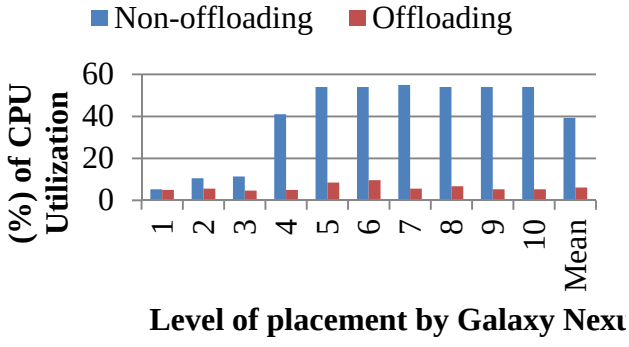
본 논문에서는 Computation 로드가 높은 오목게임을 [3] 통해서 모바일 디바이스에서 직접 실행하는 방식과 오프로딩을 통한 클라우드 방식의 서버에서 실행하는 방식을 서로 비교해 보았다. 오목 게임은 게임이 진행될수록 수가 놓아지는 경우의 수가 2 배 이상 증가하게 되며, 최적의 결과를 위한 위치를 선정하기 위해서는 몇 수 앞의 결과를 미리 Profiling 이 필요한 CPU intensive 어플리케이션이라고 할 수 있다. 본 게임은 자바스크립트로 만들어진 웹 어플리케이션이며, 모바일 디바이스의 운영체제(OS)에 상관없이 웹 브라우저를 통해서 실행될 수 있는 범용 어플리케이션이다. 오프로딩 서비스를 위해서 사용된 서버의 Specification 은 쉽게 찾을 수 있는 Intel Core2 Quad 2.83GHz/8G Memory 의 데스크톱 서버이며, 모바일 디바이스는 구글의 레퍼런스 폰인 갤럭시 넥서스 (Exynos 1.43GHz/1G memory)를 사용하였다.

어플리케이션의 성능은 주로 어플리케이션의 response time 과 단위 시간에 수행할 수 있는 처리량으로 구별할 수 있는데, 여기서는 오프로딩의 경우에 얼마나 빠른 어플리케이션의 응답을 보여주는 지에 대해서 실험하였다.



(그림 2) 오프로딩 방식과 비오프로딩 방식에서의 프로그램 응답시간 비교

그림 2 는 오프로딩 방식과 비 오프로딩 방식에서 오목 프로그램이 진행됨에 따른 단말의 응답시간을 측정한 것이다. X 축은 게임이 진행되는 수를 나타내며 게임 레벨의 수가 증가할수록 비 오프로딩 방식에서는 응답시간이 급격히 증가하는 것을 볼 수 있다. 이는 그만큼 많은 경우의 수가 발생하게 되고, 각 경우의 수에 대해서 계산의 Workload 가 급격히 증가하기 때문이다. 반면에 오프로딩 방식을 사용한 결과는 비 오프로딩 방식과 비교해서 매우 낮은 것을 알 수 있다. 오프로딩의 방식에서는 클라이언트의 좌표를 서버에게 넘기고 서버는 계산을 완료한 다음 그 결과 좌표를 모바일로 전달하게 된다. 테스트된 게임의 결과를 보면 오프로딩의 방식은 비 오프로딩 방식에 비해 단말의 성능을 평균 4 배이상 개선됨을 알 수 있다.

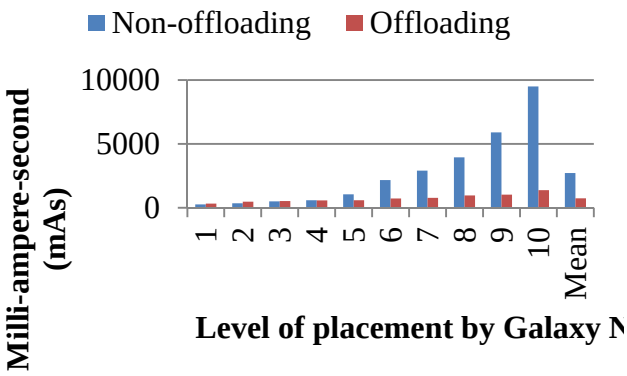


(그림 3) 오프로딩 방식과 비 오프로딩 방식에서의 CPU Utilization 비교

그림 3 에서는 그림 1 과 같은 실험을 진행할 때에 있어 각각의 경우에 대한 CPU utilization 을 Quick System tool [7] 로써 측정한 표이다. 게임의 수가 4 번째 이상부터 계산의 경우의 수가 증가하고 있고, 따라서 모바일 단말의(갤럭시 넥서스) CPU utilization 이

급격히 상승하는 하고 있다. 하나의 어플리케이션에서 무리하게 CPU 를 쓰게 되면, 모바일 디바이스가 가지는 기본적인 기능에 제약이 될 수 도 있으며 이러한 사용은 단말의 제한된 배터리 성능을 크게 저하시키는 결과를 초래하게 된다. 이에 반하여, 오프로딩의 경우는 게임의 수에 상관없이 단말의 성능에는 크게 영향을 미치지 않음을 알 수 있다. 모바일은 간단히 요구되는 좌표 값만을 넘겨주고 서버가 대신 일을 진행하기 때문에 CPU 의 활동량을 줄일 수 있고, 따라서 해당 시점에 다른 일을 할 수 있는 장점을 가진다.

모바일 환경에서 요즘 가장 중요한 이슈는 바로 전력사용을 줄이는 일이다. 모바일 단말의 특성상 제한된 전력을 이용해야 하기 때문에 전력소모가 높은 high CPU 를 사용하거나 메모리 접근이 빈번히 일어나는 어플리케이션을 사용하는 것은 다소 부담스럽다. 특정 어플리케이션 때문에 모바일 기기 및 스마트폰이 가지는 기장 기본적인 기능인 전화와 메일 체크 등을 할 수 없다면 그것은 분명 큰 문제일 것이다. 하지만 오프로딩 기법을 이용하면 그림 4 에서 보여지는 것처럼 CPU intensive 한 어플리케이션의 사용에도 단말의 전력 소모량을 크게 줄일 수 있다.



(그림 4) 오프로딩 방식과 비오프로딩 방식에서의 전력소모 비교

갤럭시 넥서스 단말은 오픈 플랫폼의 형태로써 제조사에서 제공하는 하드웨어 유닛의 각각의 평균적인 전력소모량을 알 수 가 있다. 이에 BatteryStats [8] 라는 숨겨진 운영체제의 API 를 이용하고 그림 5 에서 나타난 것처럼 전체 소비 전력 중에서 CPU 및 Network 의 사용량을 시뮬레이션 함으로써 게임이 진행되는 도중의 전력소모량을 계산 할 수 있었다. 그림 5 에 따라, 어플리케이션 A 가 소모하는 전류량의 해당 어플리케이션이 사용하는 CPU 및 Wakelock 그리고 Wi-Fi/3G 가 사용하는 시간에 따라, 제조사가 제공하는 전력효율을 계산하여, 전체적으로 단말이 소모하는 전력을 서로 비교하는 것이 가능하다. 그림 4 에서 보여주는 것처럼 비 오프로딩 방식에서는 게임의 레벨 수가 증가함에 따라 요구되는 전력이 급격히 증가하지만, 오프로딩의 방식에서 실질적인 계산은 서버가 담당하기 때문에 좌표 값을 주고 받은 부분의 전력만 필요하게 된다. 따라서, 오프로딩 방식에서의 전력량은 게임의 레벨 수와 크게 차이가 없음을 알 수 있으며, 그래프에 따르면 비 오프로딩 방식에 비해서 평균 3 배이상의 전력량을 감소 시킬 수 있었다.

- Total Power consumption of Application A = (CPU + Wakelock + Traffic + Wi-Fi/3G + sensors) power consumption.

where

CPU Power consumption =

∑ (CPU active time*Average current for CPU)

(Average current is a constant value provided by the manufacturer of the hardware)

(그림 5) 오프로딩 방식과 비 오프로딩 방식에서의 전력소모 비교

결과들을 종합해 볼 때 실험에 사용된 게임 웹 어플리케이션의 레벨이 증가할수록 컴퓨터가 계산해야 하는 수가 급격히 늘어나고 있고 이에 따라 CPU utilization 과 전력소모 또한 증가하게 된다. 따라서, 이러한 high computational workload 를 annotation 기법으로 미리 파악하여 클라우드 상의 서버들에게 오프로딩 함으로써 단말의 성능을 높이고 아울러 전력소모를 최소화 할 수 있다.

4. 제약사항 및 향후 연구

본 논문에서 제안하는 모바일 오프로딩 시스템은 여전히 몇 가지 문제점을 가지고 있으며 향후 아래와 같은 문제점들을 극복하는 것이 중요하다. 첫 번째, 웹 리소스에 주석을 삽입해서 오프로딩 여부를 결정하는 방식은 기존의 프로그램들의 수정을 가져올 수 있는 단점이 존재한다. 따라서, 웹 리소스를 자동으로 분석하여 오프로딩의 여부를 예측하는 기법이 필요할 것이다. 또한, 네트워크가 일시적으로 다운되거나 오프로딩 서버가 응답하지 않을 경우 단말과 오프로딩 서버와의 통신이 불가한 경우에, 기존의 비 오프로딩 방식으로 전환되는 서비스도 필요할 것이다. 마지막으로 CPU intensive 한 어플리케이션 외에도 보다 다양한 서비스와 어플리케이션을 모바일 오프로딩의 기법에 활용될 수 있도록 그 적용 범위를 넓혀 가는 것이 매우 중요한 향후 연구가 될 수 있을 것이다. 현재 우리는 웹 리소스를 자동으로 분석하여 오프로딩이 필요한지를 built-in Proxy 서버에서 판단하고, 자동으로 오프로딩 서비스를 생성하는 시스템을 구현 중에 있다.

5. 결론

모바일 디바이스는 데스크톱 중심의 PC 사용을 점점 대체하고 있으며, 특히 스마트폰을 위한 어플리케이션은 매일매일 폭발적으로 증가하고 있는 추세이다. 따라서 본 논문에서는 모바일 디바이스가 가지는 하드웨어 성능상의 제약과, 제한된 배터리 용량을 해결할 수 있는 하나의 기법을 제시하였다. 모바일 오프

로딩 기법은 클라우드상의 서버들을 활용하여 모바일의 태스크를 분산 처리하는 방식이다. 다양한 OS 플랫폼이 가지는 모바일 환경에서 범용적으로 사용될 수 있도록 웹 리소스를 사용하였고, 소스에 특정 주석을 삽입함으로 인해서 선택적으로 오프로딩을 결정할 수 있도록 하였다. CPU intensive 한 퍼즐게임을 사용하여 오프로딩 방식과 단말이 직접 실행하는 방식을 비교 실험 해 보았으며, 결과적으로 오프로딩 방식이 어플리케이션의 응답성도 높여주고, 단말의 워크로드를 줄여줄 뿐만 아니라 전력소모도 획기적으로 낮출 수 있음을 보여주었다.

참고문헌

[1] A. Hill, B. MacIntyre, M. Gandy, B. Davidson, H. Rouzati. Kharma:
An open kml/html architecture for mobile augmented reality applications. ISMAR ’10: proceedings of the international symposium on mixed and augmented reality (2010), pp. 233–234.
[2] Dong-Yup Lee, Rajib Saha, Faraaz Noor Khan Yusufi, Wonjun Park,
Iftekhar A. Karim. Web-based applications for building, managing
and analyzing kinetic models of biological systems. Brief. Bioinform.
[Epub ahead of print, doi:10.1093/bib/bbn039, September 19, 2008].
[3] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, RanveerChandra, and Paramvir Bahl. MAUI: making smartphones last longer with code offload. In MobiSys '10: Proceedings of the 8th international conference on Mobile systems, applications, and services. ACM, 2010.
[4] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: Elastic execution between mobile device and cloud. In Proceedings of the 6th European Conference on Computer Systems (EuroSys 2011), April 2011.
[5] “Gomoku”, http://en.wikipedia.org/wiki/Gomoku/
[6] Node.JS, http://nodejs.org, 2012-04
[7] Quick System Info, a utility application for quick access of the basic system information for Android platform, http://code.google.com/p/qsysinfo, 2012-04-17.
[8] Batterystats API, Refer to Device/frameworks/base/core/java/android/os/BatteryStats.java

* 이 논문은 2010 년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행되었으며(No. 2010-0024969), 이 연구를 위해 연구장비를 지원하고 공간을 제공한 서울대학교 컴퓨터연구소에 감사 드립니다.