모바일 응용프로그램의 성능예측이 모바일 오프로딩에 미치는 영향에 대한 연구

권용인, 양승준, 권동현, 이하윤, 백윤흥
서울대학교 전기컴퓨터공학부
e-mail : yikwon@optimizer.snu.ac.kr

A Study on effect the performance prediction of mobile application on mobile offloading

Yongin Kwon, Seungjun Yang, Donghyun Kwon, Hayoon Yi, Yunheung Paek
Dept. of Electrical and Computer Engineering, Seoul National University

요 약

스마트폰의 보편화와 함께 스마트폰 응용프로그램의 요구성능이 점점 더 높아 지고 있다. 이를 충족시키기 위해서 사용자들은 스마트폰 교체 시기가 점점 더 짧아지지만, 체감 베터리 성능은 좀처럼 나아지기 힘들다. 이러한 문제를 해결하기 위해 모바일 오프로딩 기술이 개발되고 있지만, 모바일 오프로딩 기술은 응용프로그램의 성능을 하락시킬 수도 있는 양날의 검이다. 따라서 항상 최적의 오프로딩 성능을 얻기 위해서는 정확하고 효율적인 응용프로그램 성능 예측 기술의 적용이 우선시 되어야한다.

1. 서론

최근 약 550,000 개의 새로운 안드로이드 기기가 전 세계적으로 활성화될 정도로, 스마트폰의 보편화가 증대되고 있다. 이러한 기기들은 GPS, WiFi, 카메라, 기가바이트 저장공간 그리고 기가헤르츠 단위의 연산처리속도를 지닌다. 그 결과 응용프로그램 개발자들은 게임이나 네비게이션, 비디오 편집기, 증강현실, 언어인식프로그램 등 더욱더 복잡한 응용프로그램을 개발하고 있다. 그러한 프로그램들은 더욱더 큰 연산처리속도와 전력소모를 요하지만, 사용자들은 이에 맞게 수시로 스마트폰을 업그레이드 하여야 하고, 한편으로는 높은 전력 소모로 인한 체감 배터리 용량도 줄어들게 된다.

최근 이러한 문제를 해결하기 위해 모바일 오프로딩 기술이 개발되고 있다. 모바일 오프로딩 기술은 스마트폰 상에서 응용프로그램이 실행 중에 서버로 프로그램 코드와 데이터를 전송하여 다시 수행하고, 수행한 결과를 스마트폰으로 재 전송하는 기술이다. 이 기술로 인해 스마트폰은 서버의 고속의 연산능력 및 빠른 통신속도, 풍부한 메모리 용량, 안정적인 전력공급의 이점을 얻게된다.

서버의 성능이 스마트폰에 비해 월등이 뛰어나기 때문에, 스마트폰에서 수행중이던 응용프로그램을 서버로 이전하면 더 빠른 시간내에 수행이 끝나고 이로 인해 전력 소모가 줄어야 하는 것이 이상적이지만 실제로는 그렇지 않다. 응용프로그램을 서버로 이전하기 위해서는 스마트폰에서 그동안 수행중이던 데이터를 모아서 서버에 전송을 해 주어야 하는데, 이 전송에 필요한 연산시간과 통신시간이 총 수행시간에 포함되기 때문이다. 스마트폰 대신 서버에서 프로그램을 수행 함으로써 얻는 시간적 전력적 이득을 전송하는 데이터를 모으고 보내는데 드는 시간과 전력이 상쇄시키거나 오히려 손해를 보게 할 수도 있다. 따라서 스마트폰에서 프로그램 수행중 특정 코드의 위치에서 서버로 오프로딩을 할지 말지를 정확하게 결정하는 것이 모바일 오프로딩 성능에 큰 영향을 미치게 된다. 모바일 오프로딩으로 이득을 볼 지를 정확하게 계산하기 위해서는 특정 위치의 코드를 오프로딩했을 때 서버에서 수행시간과 데이터 전송에 필요한 시간 및 전력소모량을 예측 해야 하고, 또한 만약 오프로딩을 하지 않았을 경우 필요한 스마트폰에서의 수행시간 및 전력소모도 예측 해야 한다. 하지만 위의 사항들을 예측하기 위해서는 이를 위한 추가적인 연산시간 및 전력소모가 필요하고, 정확한 예측결과를 얻기 위해서는 이 양이 더욱 커져서 스마트폰에서 수행하기에는 부담이 커질 수 있다.

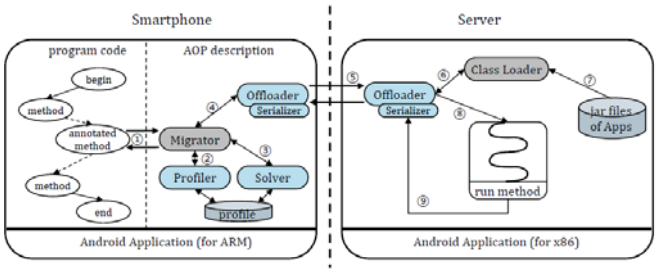
기존 연구[1,2,3]에서는 이러한 부담감을 해소하기 위해 Historical 예측 기법을 사용한다. Historical 예측 기법이란, 응용프로그램의 이전수행결과를 가지고 미래 수행에서의 예측 값으로 사용하는 것이다. Historical 예측 기법의 장점은 응용프로그램 성능 모니터링 만으로 미래의 성능 예측이 가능해, 추가적인 오버헤드가 거의 필요 없다는 점이다. 프로그램이 입력값에 민감하지 않게 동작하거나, 민감하더라도 입력값이 연속성이 있는 형태로 입력된다면 Historical

예측기법은 비교적 정확하게 작동한다. 하지만, 응용프로그램이 입력값에 민감하고, 입력값이 들쑥날쑥하다면 Historical 예측기법으로는 제대로 응용프로그램의 성능을 예측할 수가 없다.

이 연구에서 우리는 응용프로그램의 성능예측의 방법과 정확도에 따라 오프로딩 성능에 어떠한 영향을 미치는 지 알아보고, 성능예측의 오버헤드를 줄여야 할 필요성에 대해 실험을 통해 증명하였다.

2. 모바일 오프로딩

이 장에서는 모바일 오프로딩의 기본적인 구조를 설명하고 실험을 위해 설계한 간단한 모바일 오프로딩 아키텍처에 대해 살펴보겠다.



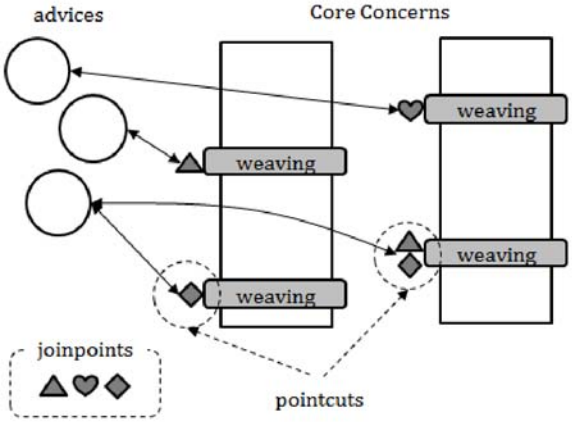
(그림 1) 모바일 오프로딩 아키텍처

그림 1 은 전형적인 모바일 오프로딩 아키텍처이다. 핵심적인 구성요소로는 Migrator, Profiler, Solver, Offloader 가 있다. 오프로딩이 가능한 응용프로그램이 스마트폰에서 수행중에 서버로 이전 가능한 표식을 만나면 일단 수행중인 프로그램을 멈추고 Migrator 를 호출한다. Migrator 는 Profiler 에게 수행에 대한 모든 정보를 기록하게 함과 동시에 Solver 를 호출하여 기록된 정보를 참조하여 오프로딩을 할 지 말지를 결정하도록 한다. 만약 오프로딩을 하지 않도록 결정되면 Migrator 가 멈춰있던 프로그램을 다시 스마트폰에서 수행하도록 하고 서비스를 종료한다. 반면에 Solver 가 오프로딩을 하도록 결정하면 Migrator 는 Offloader 를 호출하여 수행중이던 프로그램의 정보를 모아 서버에 보내도록 지시한다. 스마트폰의 Offloader 에 의해 모아진 정보는 서버측에도 있는 Offloader 가 받아서 스마트폰에서 수행중이던 응용프로그램의 상태를 똑같이 복원한다. 복원이 완료되면 서버에서 응용프로그램의 멈췄던 지점부터 다시 수행하게 되고, 수행이 끝나면 서버측의 Offloader 가 정보를 다시 모아 스마트폰의 Offloader 로 보내준다. 이렇게 받은 정보는 Migrator 에 의해 스마트폰에서 복원되고, 다시 스마트폰에서의 수행을 재개시킨다.

```
public class MainActivity {
   GAME g  = new GAME();

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

public class GAME implements Serializable {
   Object [] enemies;

   @Migratable
   public Object doGame(String cmd) {
      ....
   }
   ....
}
```

(그림 2) 오프로딩 용 응용프로그램 예제

GPS 나 터치입력 등의 기능 처럼, 스마트폰의 모든 코드가 서버에서 수행될 수는 없기 때문에 서버에서 수행할 수 있는 코드 영역을 구분해줄 필요가 있다. 프로그램분석 기술을 통하여 자동으로 영역을 나눠주는 기술도 있지만, 우리는 응용프로그램 개발단계에서 프로그램 코드 내부에 오프로딩이 가능한 코드라는 표식을 남길 수 있도록 하였다. 그림 2 는 오프로딩이 가능한 코드의 한 예이다. DoGame 메소드가 @Magratable 이라는 표식으로 마킹 함으로써 이 메소드는 서버에서 수행될 수 있다는 의미를 지니게 된다.



(그림 3) AOP 컨셉

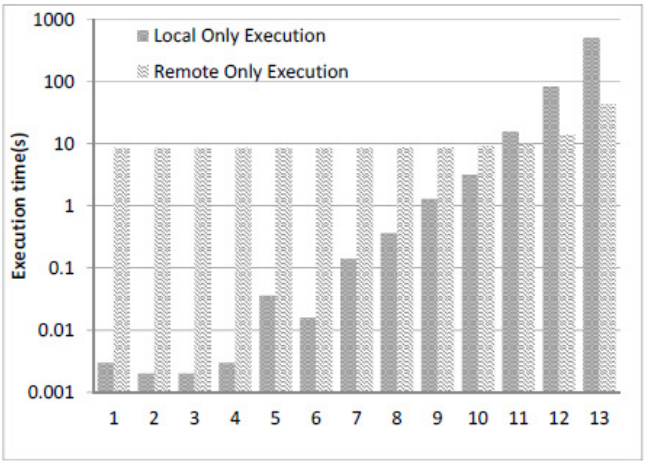
위의 오프로딩 가능 표식을 가능케 하고, 오프로딩 아키텍처를 쉽고 유연하게 구현하기 위해서 우리는 Aspect-Oriented Programming(AOP) 기법을 사용하였다. AOP 프로그램 기법은 메인 프로그램의 비즈니스 로직으로부터 2 차적 또는 보조 기능들을 고립시키는 프로그램 패러다임이다. AOP 는 프로그램을 명확한 부분으로 나눈다. AOP 의 기본요소는 Advices 이다. Advices 는 코트의 일부분으로, 프로그램이 수행될 때

프로그램 상의 특정 위치에 추가될 수 있다. 이 특정 위치는 Jointpoints(JP)라고 불린다. JP 를 Advices 에 맞추는 일을 Pointcuts 라고 부른다. 또한 Advices 를 소스코드에 추가하는 작업을 Weaving 또는 Cross-cutting 이라고 부르고, 이 작업은 컴파일중이나 로딩, 프로그램 수행중에 이루워진다. 그림 3 에 AOP 기법의 컨셉을 그림으로 나타내었다. AOP 는 오프로딩 아키텍쳐를 응용프로그램과 완전히 독립되도록 만들어준다. 안드로이드 환경 상에서 오프로딩이 가능하기 위해서는, 안드로이드의 Dalvik 가상머신을 수정하거나, 응용프로그램들을 일일이 수정하는 방법 밖에 없었다. 하지만 AOP 를 이용하면 그림 2 처럼 간단한 표식 마킹만으로도 오프로딩이 가능하게 해준다.

3. 응용프로그램 성능 예측이 모바일 오프로딩 성능에 미치는 영향

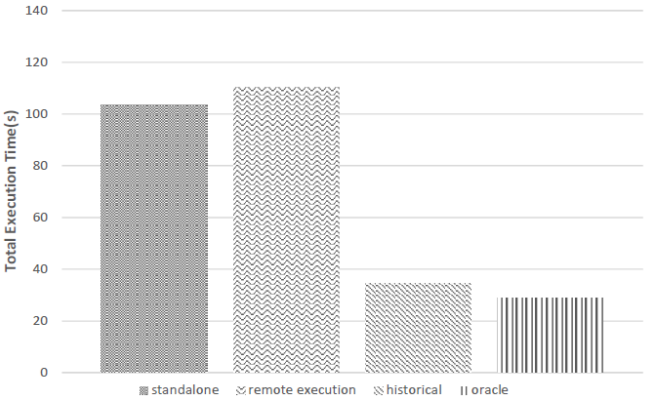
이 장에서는 2 장에서 구현은 간단한 오프로딩 아키텍쳐를 가지고 실험을 통하여 응용프로그램 성능 예측과 모바일 오프로딩 성능 사이의 관계에 대해 논의 하도록 하겠다.

실험을 위해 스마트폰은 1.2 Ghz CPU 와 1GB RAM 을 지니고, 안드로이드 4.0.3 버전의 운영체제가 설치되어있는 삼성 갤럭시 넥서스를 사용하였다. 또한 서버는 3.1 GHz quad-core CPU 와 8 GB RAM 을 지녔고, Linux 환경에서 안드로이드 가상머신을 동작시켰다.



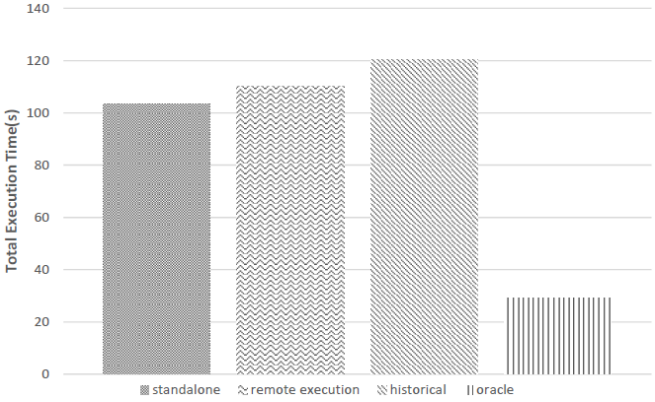
(그림 4) N-Queen 알고리즘 수행시간

먼저 스마트폰 상에서 N-Queen 알고리즘을 입력값을 1 부터 13 까지 차례대로 주면서 수행시간을 측정하였다. 또한 N-Queen 알고리즘을 무조건 서버로 오프로딩을 하도록 설정하여 수행시간을 재 측정 하였다. 그림 4 는 각각의 실험결과를 보여준다. 실험 결과에 따르면 입력값이 11 미만일 경우에는 스마트폰에서 수행하는 것이 더 빠르지만 11 이상일 때는 스마트폰에서 수행하는 것 보다 서버로 오프로딩하여 수행하는 것이 훨씬 더 빠른 것을 알 수 있다. 이와 같이 입력값과 응용프로그램의 성능을 분석하여 프로그램 개발자가 서버로의 오프로딩 조건을 입력할 수 있다면 효율적인 오프로딩이 가능하겠지만 이에는 여러 문제가 따른다. 첫째, 모든 응용 프로그램의 성능이 N-Queen 알고리즘과 같이 단순한 입력 값 하나에 따라 좌우되지 않는다. 둘째, 스마트폰와 서버의 성능에 따라 응용프로그램의 성능이 달라지기 때문에 개발자가 임의로 정하기는 힘들다. 셋째, 오프로딩을 할지를 결정하는 요소가 수행시간 외에도 에너지소모, 통신속도 등 여러 요소가 많기 때문에 변수가 많다. 넷째, 그럼에도 불구하고 개발자가 수동으로 코드를 수정해야 하는 것은 많은 부담감이 따른다. 따라서 오프로딩 아키텍쳐 내에는 프로그램 성능을 자동으로 예측하여 오프로딩을 할 지를 결정하는 Solver 가 이를 참조하여 문제를 풀 수 있도록 해야한다.



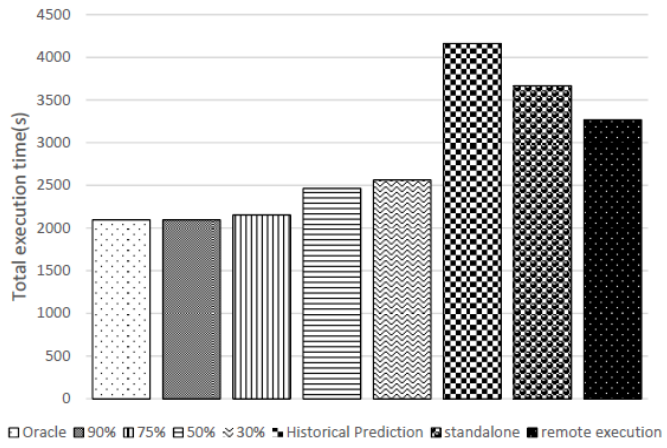
(그림 5) 정렬된 입력값에 대한 N-Queen 수행시간

그림 5 는 N-Queen 알고리즘에 1 부터 13 까지 입력값을 차례대로 주었을 때 스마트폰 단독으로 수행했을 때와 항상 서버로 오프로딩을 하였을 때, Historical 예측기법을 사용할 때, 마지막으로 이상적인 경우에 대해 각각 실험하였다. 실험 결과에 따르면 이 응용프로그램이 오프로딩을 함으로서 얻을수 있는 최소 수행시간은 Oracle 과 같다. Historical 예측 기법을 사용하면 Oracle 만큼은 아니지만 매우 근접한 결과를 얻을 수 있다. 반면에 스마트폰에서 단독으로 수행하였을 때와, 무조건 서버로 오프로딩을 하였을 때는 이들보다 약 세배 이상의 수행시간을 보인다. 따라서 Historical 예측 기법의 성능이 뛰어나다고 할 수 있다.



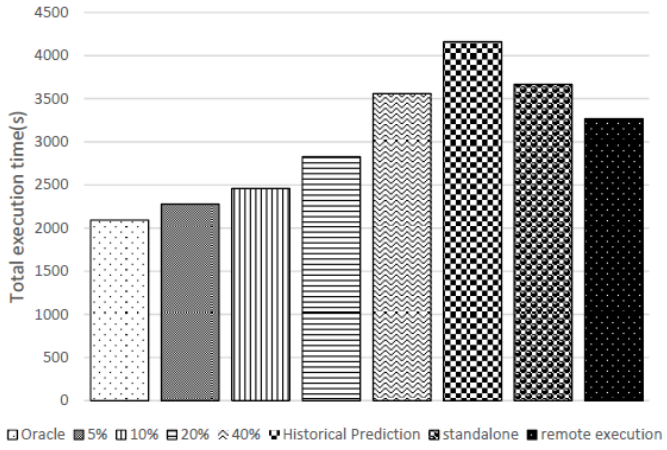
(그림 6) 무작위 입력값에 대한 N-Queen 수행시간

반면에 그림 6 에서는 N-Queen 알고리즘에 1 부터 13 까지의 입력을 무작위로 주었을 때 그림 5 와 같이 각각 네가지 경우의 수행시간에 대한 실험을 하였다. 입력값을 차례대로 준 그림 5 와 비교해서 스마트폰 단독수행 했을 때와 항상 서버로 오프로딩을 하는경우, 그리고 Oracle 의 경우는 실험결과가 같게 나왔다. 하지만 Historical 예측 기법을 사용 했을 경우에는 그림 5 와는 달리 나머지 세가지 경우 모두에 비해 높게 나왔다. 이는 잘못된 응용프로그램 성능 예측 결과가 오히려 큰 손해를 입힐 수 있다는 점을 보여준다.



(그림 7) 성능예측 정확도 별 N-Queen 수행시간

위에서 우리는 Historical 예측기법 만으로는 오프로딩 성능 저하를 올 수 있다는 것을 알았다. Oracle 에 근접한 오프로딩 성능을 얻기 위해서는 응용프로그램의 성능 예측을 수행 중에 정확히 할 수 있어야 한다. 성능 예측의 정확도가 오프로딩 성능에 어떠한 영향을 미치는 지 알아보기 위해 실험을 해 보았다. 그림 7 을 각 성능 예측 정확도 별로 N-Queen 알고리즘 입력값 중 8 과 13 사이의 값을 무작위로 100 개를 선택하여 총 수행시간을 측정해 보았다. 그 결과 성능예측 정확도가 75% 이상일 경우에는 Oracle 에 근접하는 것을 확인 하였다.



(그림 8) 성능예측 오버헤드 별 N-Queen 수행시간

보편적으로 응용프로그램의 성능예측의 정확도와 성능예측을 위한 오버헤드 시간은 Trade-off 을 가진다. 따라서 높은 성능 정확도를 얻기 위해서는 그만큼 시간이 많이 필요할 수 있다. 그림 8 은 90%의 성능 예측 정확도 일 때, 각 오버헤드(스마트폰에서 단독 수행시의 수행시간에 대한 비율)에 따라 오프로딩의 성능에 어떠한 영향을 미치는 지를 보여준다.

4. 결론

모바일 오프로딩 기술은 스마트폰의 부족한 연산능력 및 메모리 용량을 보완하고 베터리 소모를 줄여 가능한 오래 쓰게 해준다. 하지만 Solver 의 오프로딩 결정에 따라 오히려 성능에 악영향을 미칠 수도 있다. 특히 Historical 예측 기법은 입력값 패턴에 따라 성능 예측이 제대로 되지 않는 경우가 있다. 따라서 응용 프로그램 수행중 성능 예측을 정확하게 할 필요성이 있고, 이와 더불어 성능 예측을 위한 오버헤드를 줄인다면 최적에 가까운 오프로딩 성능을 얻을 수 있을 것이다.

참고문헌

[1] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, “MAUI: making martphones last longer with code offload,” in Proc. ACM MobiSys, 2010.
[2] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, “ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading,” in Proc. IEEE INFOCOM, 2012.
[3] D. Kovachev, T. Yu, and R. Klamma, “Adaptive computation offloading from mobile devices into the cloud,” in Proc. IEEE ISPA, 2012.

Acknowledgement

본 연구는 교육과학기술부/한국과학재단 우수연구센터 육성사업(과제번호 2012-0000470), 중소기업청에서 지원하는 2012 년도 산학연공동기술개발사업(No. C0019562), 미래창조과학부 및 한국산업기술평가관리원의 산업융합원천기술개발사업(정보통신) [No. 10047212, 1kB 이하 암호문 간의 연산을 지원하는 동형 암호 원천 기술 개발 및 응용 연구] 및 IDEC 의 지원을 받아 수행하였습니다.