모바일 기기를 위한 효과적인 Process Offloading 아키텍처에 관한 연구

양승준, 조영필, 권용인, 권동현, 이하윤, 백윤흥
서울대학교 공과대학 전기컴퓨터공학부
e-mail : sjyang, ypcho, yikwon, dhkwon, hyyi@sor.snu.ac.kr, ypaek@snu.ac.kr

A Study on Effective Process Offloading Architecture for Mobile Device

Seungjun Yang, Yeongpil Cho, Yongin Kwon, Donghyun Kwon, Hyyoon Yi, Yunheung Paek
Dept. of Electrical Engineering, Seoul National University

요 약

Process Offloading 기술은 동작 중인 어플리케이션의 일부를 보다 강력한 성능을 가진 다른 기기로 옮겨 실행하는 기술로써, 스마트폰과 같은 모바일 기기에 적용될 경우 어플리케이션의 수행 속도를 향상시키고 배터리 소모 및 발열을 크게 줄일 수 있다. 본 논문에서는 기존의 연구를 조사 및 고찰하여 보다 효과적인 Process Offloading 아키텍처를 제시하고자 한다.

1. 서론

스마트폰의 등장으로 인해 모바일 기기 시장은 급격한 패러다임의 변화를 맞게 되었다. 음악 감상이나 동영상 시청, 사진 촬영과 같은 독립적인 기능들이 스마트폰을 중심으로 통합되는 동시에 iOS 나 안드로이드와 같은 스마트폰을 위한 운영 체제가 등장하게 되면서, 개발자들이 이러한 기능들을 효과적으로 이용하여 기존에 비해 훨씬 다양하고 복잡한 어플리케이션을 제작할 수 있게 된 것이다. 사용자들의 눈높이 또한 점점 더 높아지면서 보다 더 빠르고 복잡한 어플리케이션에 대한 요구가 점점 증가하게 되었다.

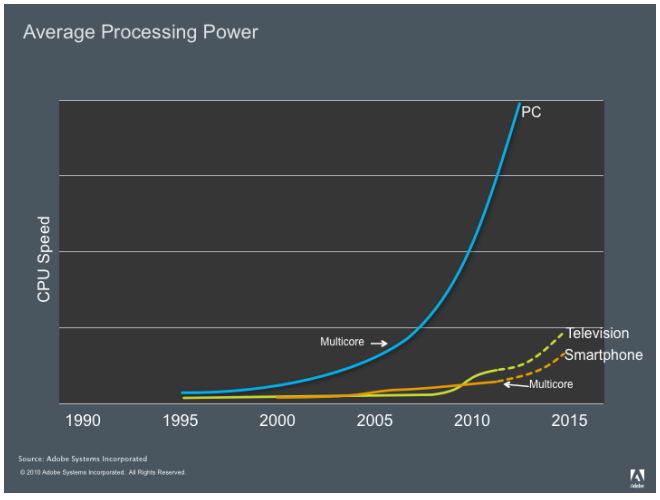

그림 1 시간에 따른 기기 별 연산 처리 능력 비교[1]

이처럼 모바일 어플리케이션의 복잡도가 급격하게 증가하는 반면 스마트폰을 비롯한 모바일 기기의 성능 향상은 이를 충분히 뒷받침하지 못하고 있다. 모바일 기기라는 태생적 한계로 인해 스마트폰은 제한된 크기와 발열 구조를 가질 수밖에 없기 때문이다. 이로 인해 모바일 기기에서는 PC 에서 일어났던 것과 같은 비약적인 성능 향상을 기대하기 힘들다. 위 그림에서는 이러한 스마트폰 성능 향상의 한계를 잘 보여주고 있다. 그림 1 은 각 기기별 연산 처리 능력의 차이를 시간에 따라 나타낸 그래프이다. 이 그래프를 보면 스마트폰의 경우 시간에 따른 연산 처리 능력의 향상 정도가 PC 에 비해 현격하게 떨어지는 것을 볼 수 있다. 특히 PC 에서는 멀티코어 기술이 적용된 이후 성능이 급격하게 향상된 반면 스마트폰에서는 멀티코어 기술이 적용된 이후에도 PC 에서만큼의 성능 향상이 이루어지지 않은 것을 확인할 수 있는데, 이는 앞서 설명한 것처럼 스마트폰의 구조상 여러 개의 코어를 사용하는 고성능 CPU 를 사용할 경우 발생할 수 있는 여러 가지 문제점들을 해결하는 것이 어렵기 때문이라고 볼 수 있다. 실제로 현재 출시되고 있는 대부분의 스마트폰에서 3D 그래픽 처리가 빈번한 고성능의 어플리케이션을 실행할 경우 극심한 배터리 소모 및 발열로 인해 장시간 구동시키기에는 무리가 따르는 것이 현실이다. 이러한 문제점은 향후 기존에 개발된 PC 용 어플리케이션 혹은 이에 준하는 고성능 모바일 어플리케이션을 스마트폰에서 구동시키고자 할 때에 큰 걸림돌이 될 수 있다.

Process Offloading 기술은 수행 중인 어플리케이션의 일부를 보다 강력한 성능을 가진 다른 기기로 옮겨 실행하는 기술로써, 스마트폰과 같은 모바일 기기에 적용될 경우 어플리케이션의 수행 속도를 향상시키고 배터리 소모 및 발열을 크게 줄일 수 있다는 점에서 앞서 설명한 문제점을 해결할 수 있는 하나의 돌파구

가 될 수 있다. 모바일 기기 및 실행 흐름 분기의 대상이 되는 기기의 성능과 어플리케이션을 구성하는 각각의 함수에 대한 정보를 분석하여 "offloading"의 후보가 되는 함수들을 결정하고 나면 사용자의 통신 상태에 따라 후보 함수 중에서 다른 기기로 옮길 함수들을 동적으로 선택하는 것이다. 이런 방식을 통해 원래 모바일 기기에서 전부 실행되어야 했을 어플리케이션의 일부를 다른 기기에서 실행함으로써 모바일 기기의 배터리 소모 및 발열을 줄이고 전체적인 수행 성능을 향상시킬 수 있다. 특히 최근 들어 각광받고 있는 클라우드 서비스와 연동하게 되면 사용자의 스마트폰과 주변의 접근 가능한 클라우드 서버를 연결시킨 후 어플리케이션의 일부를 분산시킴으로써 사용자에게 어플리케이션을 위한 최적화된 컴퓨팅 환경을 seamless 하게 제공할 수 있게 된다. 이러한 장점으로 인해 Process Offloading 기술은 차세대 모바일 컴퓨팅 환경을 위한 최적화 기법으로 각광받고 있으며, 인텔 및 마이크로소프트와 같은 시장 선도 기업에서도 활발히 연구되고 있다.

본 논문에서는 기존에 연구되었던 Process Offloading 기술을 고찰하여 장단점을 분석한 후, 이를 바탕으로 보다 더 효과적인 Process Offloading 아키텍처를 제안하고자 한다. 먼저 2 장에서는 기존 연구를 소개하고 이들의 장단점을 분석한다. 3 장에서는 기존 연구의 단점을 개선한 Process Offloading 아키텍처에 대해 설명한다. 4 장에서는 향후 연구 방향을 설명하고, 마지막 5 장에서는 전체 내용을 요약하고 결론을 제시한다.

2. 기존 연구 소개 및 장단점 분석

Process Offloading 기술은 어플리케이션의 흐름을 옮기는 작업이 실행 환경의 어느 곳에서 일어나느냐에 따라 다음과 같이 구분된다[2].

- 어플리케이션 레벨
- 가상 머신 레벨

첫 번째 방식에서는 Offloading 에 관한 처리를 어플리케이션, 즉 소스 코드 레벨에서 담당한다. 상대적으로 구현이 쉽고 간단하여 Offloading 을 위해 필요한 오버헤드(overhead)가 적다는 장점이 있으나 소스 코드를 수정해야 하기 때문에 개발자의 부담이 가중된다는 단점이 있다. 두 번째 방식에서는 Offloading 에 관한 모든 처리를 어플리케이션을 동작시키는 가상 머신에서 처리하게 된다. 이러한 방식에서는 소스 코드를 수정할 필요가 없이 모든 작업을 가상 머신에서 처리하므로 개발자의 부담이 없다는 장점이 있으나 구현이 복잡하고 오버헤드가 크다는 단점이 있다. 이 장에서는 각 방식의 대표적인 연구를 소개하고 이들의 장단점을 분석한다.

2.1. MAUI[3]

MAUI 는 마이크로소프트의 .NET Common Language Runtime (CLR)을 기반으로 동작하는 Process Offloading 기술이다. 다음 그림은 MAUI 아키텍처의 전반적인 구조이다.

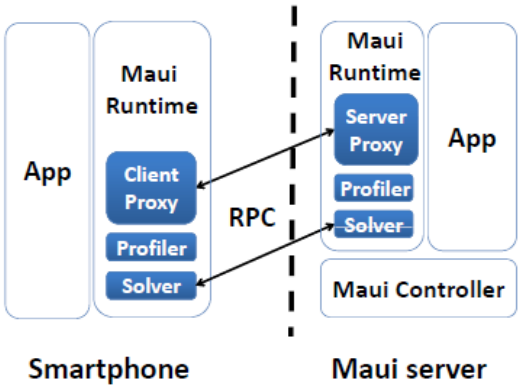

그림 2 MAUI 아키텍처의 구조

MAUI 는 어플리케이션을 구성하는 각 함수의 일부분을 서버에서 실행함으로써 모바일 기기의 성능을 높이고 배터리 소모를 줄인다. 어플리케이션 개발자가 서버에서 실행할 수 있는 ("remoteable") 함수를 미리 지정해 놓으면 MAUI 는 이들 함수가 서버에서 실행되는 데에 필요한 정보를 주고 받을 수 있도록 소스 코드를 변경한다. 이렇게 변경된 함수들은 어플리케이션 실행 도중 모바일 기기와 서버 간의 통신 상태에 따라 어느 쪽에서 실행될지 결정되며, 이렇게 서버에서 실행될 함수가 결정되면 모바일 기기에서는 RPC 통신을 이용하여 필요한 정보를 넘기고 서버는 이를 받아 해당 함수를 실행시킨다.

MAUI 는 서버에서 실행될 함수를 어플리케이션 실행 도중의 통신 상태에 따라 동적으로 결정함으로써 offloading 의 효율을 높이는 동시에 각 함수가 필요로 하는 정보만 서버로 전송함으로써 offloading 에 필요한 오버헤드를 최소화한다는 것이 장점이다.. 하지만 개발자가 서버에서 실행 가능한 함수를 일일이 지정해야 하고, 해당 함수들이 서버에서 실행될 수 있도록 소스 코드를 수정해야 하기 때문에 개발자의 개발 부담을 가중시킬 뿐만 아니라 이미 완성되어 바이너리 형태로 배포되는 어플리케이션에는 적용시킬 수 없다는 단점을 가지고 있다.

2.2. CloneCloud[4]

CloneCloud 는 인텔 리서치 그룹에서 2008 년부터 연구되어 온 Process Offloading 기술이며 구글의 안드이드 OS 를 기반으로 동작한다.

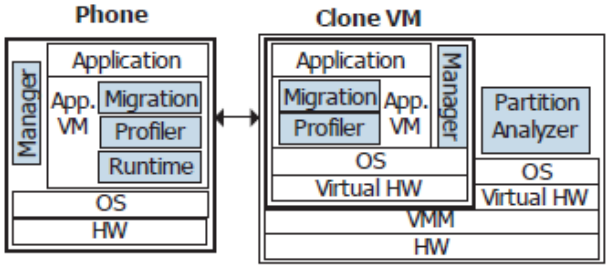

그림 3 CloneCloud 아키텍처

CloneCloud 의 대략적인 동작 원리는 다음과 같다. 먼저 어플리케이션이 실행되기 전에 바이너리 분석 도구를 이용하여 어플리케이션 바이트코드(bytecode)를 분석한 후 offloading 의 후보가 될 수 있는 함수들을 추려낸다. 이 때 각 후보 함수의 시작 부분 및 종료 부분에 분기(migrate)와 합기(re-integrate)를 나타내는 명령어를 삽입한다. 후보 함수들을 판별하고 나면 사전에 측정한 모바일 기기 및 서버의 성능에 대한 프로파일 정보, 모바일 기기와 서버 간의 통신 연결 상태와 분기 시 참조해야 할 전제 조건 등을 고려하여 offloading 될 함수들을 결정한다. 이렇게 결정된 '통신 상태에 따른 각 후보 함수 별 분기 여부'를 파티션 시나리오(partition scenario)라 부르며 모바일 기기 내부의 데이터베이스에 저장되었다가 어플리케이션이 시작될 때 offloading 관리자에 의해 호출된다. 어플리케이션이 실행되고 나면 달빅 가상 머신(Dalvik VM)에서 바이트코드를 실행하게 되는데, 실행 도중 분기 명령어를 만나게 되면 어플리케이션의 실행을 멈추고 서버에 모바일 기기의 클론(clone)을 만든 후 실행 흐름을 옮긴다. 이 때 서버에 생성된 클론과 모바일 기기의 상태를 동일하게 만들어 주기 위해 달빅 가상 머신의 힙(heap) 오브젝트와 함께 해당 함수가 실행되던 스레드의 스택 자료 구조 또한 서버로 전송한다. 이러한 과정을 거쳐 생성된 클론 어플리케이션을 실행하는 도중 합기 명령어를 만나면 분기 때와 동일한 방법으로 실행 흐름을 모바일 기기로 돌려준다.

CloneCloud 기술은 안드로이드 OS 의 달빅 가상 머신을 수정하는 방식으로 offloading 을 구현하였기 때문에 어플리케이션을 수정할 필요가 없어 개발자의 작업 편의가 매우 높다는 장점이 있다. 하지만 어플리케이션 시작 전에 통신 상태에 따라 분기할 함수들을 미리 결정한 후, 어플리케이션이 시작될 때의 통신 상태에 따라 미리 정해진 분기 시나리오대로만 동작한다는 점에서 어플리케이션 동작 도중 통신 상태가 바뀌었을 경우 분기 효율이 떨어진다는 것이 단점이다. 분기를 위해 달빅 가상 머신의 힙 오브젝트, 스택 자료구조와 같은 내부 상태를 클라우드 서버로 전송해야 한다는 점과 분기 여부를 결정하는 데 있어서 모바일 기기와 클라우드 서버 각각의 성능, 각 기기 간 통신 상태만을 고려하기 때문에 모바일 기기의 CPU 사용량이나 어플리케이션의 메모리 점유 등을 실시간으로 반영하지 못한다는 점 또한 단점으로 꼽힌다.

3. 제안된 Process Offloading 아키텍처

앞서 설명한 기존 연구들은 장점 및 단점을 동시에 가지고 있다. MAUI 의 경우 소스 코드를 수정해야 한다는 것이 가장 큰 단점이며, CloneCloud 의 경우 어플리케이션 수행 도중 바뀌는 통신 환경을 반영하지 못하며 offloading 을 위한 오버헤드가 크다는 단점을 가지고 있다. 본 장에서는 이러한 단점들을 개선하는 동시에 Process Offloading 의 효율을 높일 수 있는 아키텍처를 제안하고자 한다.

본 연구에서 제안하는 아키텍처는 기본적으로 가상 머신을 수정하여 offloading 을 수행하는 방식을 따름으로써 기존 어플리케이션의 바이너리를 수정하지 않고도 offloading 이 가능하도록 하였다. 아키텍처를 구성하고 있는 중요 모듈에 대한 설명은 다음과 같다.

- Solver

어플리케이션을 구성하는 함수들 중 어떤 함수들이 실제로 서버로 offloading 될 것인지 결정하는 solver 는 Process Offloading 기술의 성능을 결정하는 핵심 중의 핵심이다. 본 연구에서 제안하는 solver 는 정적 방식과 동적 방식의 두 가지 방법으로 offloading 될 함수를 결정한다. 먼저 정적 방식에서는 스마트폰 및 서버의 연산 처리 능력, 어플리케이션이 처음 시작될 때의 통신 상태만을 고려하여 어떤 함수가 서버로 offloading 되어야 스마트폰의 성능 및 배터리 소모량이 향상될지를 계산한다. 이렇게 계산된 각 함수 별 offloading 여부를 분기 시나리오라고 하며, 만들어진 통신 상태 별 시나리오는 시나리오 DB 에 저장되었다가 어플리케이션이 시작될 때 사용된다.

이처럼 정적 방식을 통해 계산된 시나리오는 어플리케이션 수행 도중 변하는 상황을 반영하지 못하기 때문에 자칫 비효율적일 수 있다는 단점이 있다. 따라서 본 연구에서는 solver 에 동적 방식을 추가하여 어플리케이션 수행 도중 바뀌는 여러 가지 변수를 확인하고 이를 반영하여 정적 방식으로 결정된 분기 시나리오를 변경할 수 있도록 한다.

- Migrator

Migrator 는 가상 머신의 Heap 오브젝트나 Stack 과 같이 offloading 을 위해 반드시 필요한 가상 머신의 상태 정보를 수집하여 전송하거나, 전달 받은 정보를 이용하여 서버에 만들어진 클론 어플리케이션의 상태를 offloading 가능하도록 만들어주는 역할을 담당한다. 어플리케이션 내의 인터프리터 Thread 와 연동하여 어플리케이션 수행 도중 분기 명령어를 만나면 solver 로부터 실제 offloading 여부를 전달받는 동시에 관련 함수를 호출하는 역할도 수행한다. 수집한 가상 머신의 상태 정보를 offloader 로 전달하여 서버로 전송할 수 있도록 하는 것도 migrator 의 역할이다. 본 연구에서는 앞으로 연구될 최적화 기법을 적용하기에 용이한 구조를 가지도록 migrator 를 설계하였다.

- Offloader

Offloader 는 Process Offloading 기술을 구성하는 각 모듈을 관리하는 역할을 담당하는 모듈이다. 연결 가능한 offloading 서버를 찾고 모바일 기기와 서버를 연결하여 offloading 이 가능하도록 수행 환경을 조성하는 역할 또한 담당한다.

이처럼 본 연구에서는 기존 연구의 단점을 개선하는 동시에 향후 연구될 최적화 기법을 쉽게 적용할 수 있도록 Process Offloading 아키텍처를 설계하였다. 다음 그림은 제안된 아키텍처의 구조를 간략하게 나타낸 것이다.

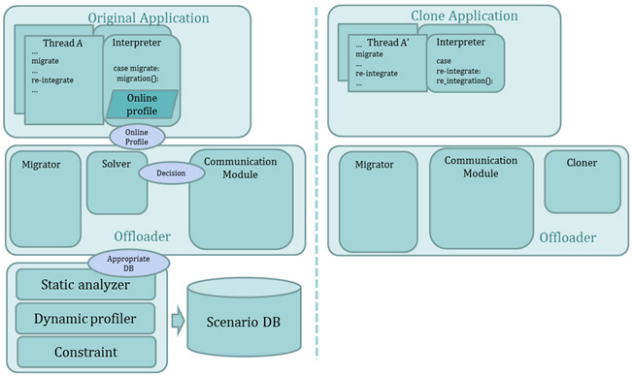

그림 4 제안된 아키텍처의 구조

4. 향후 연구 방향

보다 더 효과적으로 Process Offloading 을 수행하기 위해서는 기존 연구에서 연구되지 않았던 새로운 최적화 기법이 필요하다. 이 장에서는 이러한 최적화를 위해 필요한 향후 연구 방향을 설명한다.

○ 컴파일러 테크닉을 이용한 최적화 기법

기존의 연구들은 성능 향상의 걸림돌이 되는 난제들을 많이 가지고 있다. 예를 들어 CloneCloud 기술의 경우, offloading 을 위해 가상 머신의 상태를 서버로 전송할 때 유효한 Heap 오브젝트들을 전부 전송해야 한다는 단점이 있다. 또한 서버의 성능 측정을 위해 주어진 어플리케이션을 사전에 실행하여 프로파일링 하는데, 서버의 경우 스마트폰과 달리 어플리케이션의 입력을 재현하는 데에 한계가 있어 제대로 된 결과를 얻을 수 없기 때문에 결과적으로 solver 의 성능이 떨어진다는 것 또한 단점이다.

따라서 향후 연구에서는 컴파일러 테크닉을 이용하여 이러한 문제들을 개선하기 위한 최적화 기법을 개발하고자 한다. 예를 들어 WCET(Worst Case Execution Time)과 같은 컴파일러 테크닉을 이용하면 서버 성능을 프로파일링 할 때 생기는 입력 재현 문제를 보완할 수 있어 solver 의 성능을 향상시킬 수 있다. 또한 전통적인 컴파일러 최적화 기법을 이용하여 소스 코드를 분석하고 이를 Heap 공간을 관리하는데 반영하면 가상 머신 상태 전송에 따른 오버헤드를 크게 줄이는 것이 가능할 것이다. 추후 연구를 통해 이러한 최적화 기법을 연구함으로써 Process Offloading 기술의 효율을 크게 향상시킬 수 있을 것으로 기대된다.

○ CPU 사용량과 메모리 점유를 반영하기 위 최적화 기법

지금까지의 Process Offloading 연구에서는 offloading 을 결정하는 데 있어서 스마트폰 및 서버의 성능, 스마트폰의 통신 상태만을 반영하였다. 하지만 실제 스마트폰 동작 환경에서는 이 외에도 고려해야 할 변수가 많다. 예를 들어 3D MMORPG(Massive Multiplayer Online Role Playing Game)과 같은 어플리케이션의 경우, 사용자들의 조작에 따라 각 사용자들에게 보여줘야 하는 오브젝트의 수가 급격히 늘어남으로써 이를 렌더링하기 위한 연산의 양이 갑자기 엄청나게 늘어나는 상황이 발생할 수 있다. 이러한 상황은 결과적으로 CPU 사용량을 급격하게 증가시켜 어플리케이션의 성능을 급격하게 떨어뜨린다. 또한 각각의 어플리케이션마다 사용할 수 있는 메모리의 양을 제한하고 있는 스마트폰의 실행 환경으로 인해 어플리케이션은 항상 메모리 부족에 시달릴 수밖에 없으며 이는 어플리케이션의 오동작으로까지 이어질 수 있다.

이러한 문제점을 해결하기 위해 향후 연구에서는 CPU 의 사용량이나 메모리 점유와 같이 어플리케이션 수행 도중 동적으로 바뀔 수 있는 변수를 고려할 수 있는 Process Offloading 기법을 개발하고자 한다. 이를 통해 보다 다양한 실행 환경을 반영하여 Process Offloading 을 수행함으로써 결과적으로 모바일 기기의 성능과 배터리 수명을 향상시킬 수 있을 것이다.

5. 결론

본 논문에서는 Process Offloading 기술에 대한 기존 연구를 분석하여 장단점을 파악하고 이를 개선할 수 있는 아키텍처를 제안하였다. 제안된 아키텍처는 두 가지 방식의 Process Offloading 기술의 장점을 혼합하여 각각의 방식이 가지는 단점을 개선하고 효율을 높일 수 있도록 설계되었다. 또한 Process Offloading 기술의 효율을 보다 향상시키기 위해 필요한 최적화 기법의 연구 방향을 제시하였다.

Acknowledgement

본 연구는 교육과학기술부/한국과학재단 우수연구센터 육성사업(과제번호 2012-0000470), 2011 년도 정부(교육과학기술부)의 재원으로 한국과학재단의 국가지정연구실사업(No.2011-0018609) 및 IDEC 의 지원을 받아 수행되었습니다.

참고문헌

[1] Adobe Featured Blogs, http://blogs.adobe.com

[2] S.Bouchenak and D.Hagimont, “Zero Overhead Java Thread Migration”, *INRIA*, 2002

[3] E.Cuervo, A.Balasubramanian, D.Cho, A.Wolman, S.Sariou, R.Chandra and Paramvir Bahl, “MAUI: Making Smartphones Last Longer with Code Offload”, *The 8th Annual International Conference on Mobile Systems, Applications and Services (MobiSys)*, 2010

[4] B.Chun, S.Ihm, P.Maniatis, M.Naik, and A.Patti, “CloneCloud: Elastic Execution between Mobile Device and Cloud”, *The European Professional Society on Computer Systems (EuroSys)*, 2011

[5] U.Kremer, J.Hicks, and J.M.Rehg, “Compiler-Directed Remote Task Execution for Power Management”, *Proceedings of The Workshop on Compilers and Operating Systems for Low Power (COLP)*, 2000

[6] 민옥기, 김학영, 남궁한, “클라우드 컴퓨팅 기술 동향”, 전자통신동향분석, 2009