모바일 클라우드 환경에서 레거시 어플리케이션을 위한 오프로딩 프레임워크

김순곤[1], Abdullah Yousafzai[2], 고광만[2]
[1]중부대학교 컴퓨터게임학과,
[2]상지대학교 컴퓨터정보공학부
sgkim@joongbu.ac.kr, [2]kkman@sangji.ac.kr

Offloading Framework for Legacy Application in Mobile Cloud Environments

Soon-Gohn Kim[1], Abdullah Yousafzai[2], Kwang-Man Ko[2]
[1]Dept. of Computer Game Engineering, Joongbu University
[2]School of Computer and Information Engineering, Sangji University

요 약

최근까지 모바일 디바이스와 고성능 클라우드 서버는 동일한 DVM 실행시간 환경에서 오프로딩을 통해 모바일 디바이스의 어플리케이션에 대해 실행속도 개선하려는 연구가 진행되고 있다. 본 논문에서는 안드로이드 실행시간 환경이 네이티브 어플리케이션을 지원하는 ART로 완전하게 전환되는 상황에서 DVM에서 실행되고 있는 모바일 레거시 어플리케이션에 대해 모바일 디바이스의 복잡한 계산 부담을 줄여 실행속도를 향상시고, 이를 통해 배터리 소모를 감소시키는 프로세스 단위 오프로딩 프레임워크에 대한 설계 내용을 제시한다.

1. 서론

안드로이드 운영체제는 어플리케이션 실행시간 환경인 달빅가상기계(Dalvik VM)의 실행시간 속도 향상 등을 목적으로 컴파일러 기술을 통해 바이트코드로부터 ARM-x 기반 네이티브 코드를 생성하는 새로운 실행시간 환경 ART(Android RunTime)를 Android 2.2 "Froyo"부터 시작하여 최근에는 Ahead-Of-Time(AOT) 네이티브 코드 생성 기술을 적용하여, DVM을 완전히 배제한 ART를 Android 5.0 "Lollipop"에 구축하여 서비스하고 있다. 이러한 ART의 적용은 어플리케이션의 전체적인 실행 효율성 향상, 모바일 디바이스의 전력소비 감소에 따른 배터리 소모 부담 감소 등의 다양한 효과를 유발시키고 있다[1].

ART의 새로운 대체는 바이트코드에 대해 네이티브 코드 생성에 필요한 추가적인 컴파일 시간과 네이티브 코드를 저장하기 위한 메모리 오버헤드가 발생되고 있다. 또한, 기존 DVM에서 안정적으로 실행되었던 기존 모바일 어플리케이션(legacy application)이 새로운 ART에서도 안정적으로 실행될 수 있도록 추가적인 지원이 진행되어야 한다. 특히, 모바일 디바이스에 적합하도록 ARM-x 계열의 네이티브 코드를 생성하여 실행하는 레거시 어플리케이션은 더 이상 풍부한 컴퓨팅 자원과 고성능 처리능력을 가진 x86 기반의 클라우드 서버에서 오프로딩을 통해 실행시간 및 에너지 소모감소의 효과를 얻을 수 없다. 즉, 기존에는 모바일 디바이스와 고성능 서버가 동일한 DVM을 제공하여 실행시간 환경이 동일하였지만, 네이티브 코드 생성을 통한 ART의 채택은 모바일 디바이스와 서버가 서로 다른 실행시간 환경을 유지하므로 오프로딩을 통한 이득을 얻을 수 없다[2].

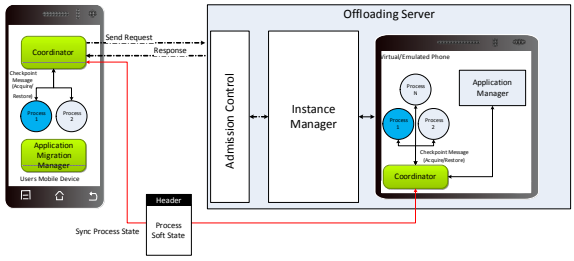
2. 관련연구

현재까지 오프로딩 연구는 가상기계 복제 이전[3], 코드이전 스레드 상태 이전 메카니즘으로 크게 구분되고 있다. 가장 활발하게 적용되고 있는 코드 이전 기술은 계산이 복잡하거나 에너지 소모가 많은 모바일 디바이스의 어플리케이션의 코드일부를 원격지 고성능 서버로 이전하여 실행하는 것으로 전통적인 기존 MAUI[4]를 기반으로 하고 있다. 스레드 상태 이전 기술은 힙 내용, 스택 레지스터 값과 같은 저수준의 스테드 상태를 원격지 서버를 이전하여 실행하는 것으로 CloneCloud[5]와 COMET[6] 연구가 대표적이다. COMET은 안드로이드 DVM의 메모리 모델을 어플리케이션 수준에서 모바일 디바이스와 클라우드 서버가 가상기계 동기화가 가능하도록 하여 스레드 상태이전이 가능하도록 하였지만, 최근에 안드로이드가 DVM을 ART로 전환하면서 스레드 이전을 통한 오프로딩의 효과를 더 이상 갖을 수 없는 상황이 되었다.

3. 레거시 어플리케이션을 위한 오프로딩 프레임워크

모바일 디바이스에서 동작하는 핵심 모듈은 어플리케이션 이전 메니저, 어플리케이션 이전 코디네이터이며, 안드로이드 어플리케이션의 바이트코드를 동적 코드분석을 통해 프로세스 단위 오프로딩 단위를 자동적으로 결정하

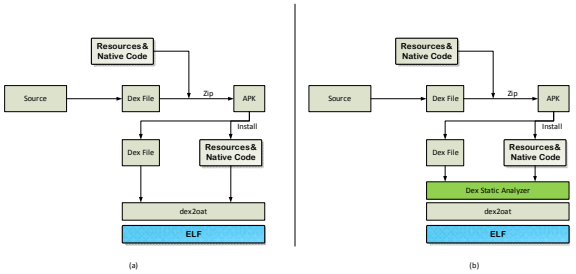
는 분할, 다양한 프로세스가 오프로딩 서버에서 최적화 수행은 위한 프로세스 스케줄링, 오프로딩 요청 및 응답수신, 모바일 어플리케이션을 대한 에너지 소비에 대한 프록파일 정보 추적을 수행한다.



(그림 1) 레거시 어플리케이션을 위한 오프로딩 프레임워크(Bird-eye View)

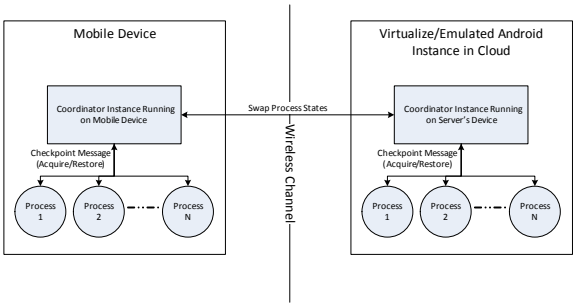
모바일 어플리케이션의 바이트코드를 정적 분석 기법을 통해 분석한 후 오프로딩 시작/반환 지점에 마커(Migration_Marker_Send, Migration_Marker_Receive)를 어플리케이션 개발자의 추가적인 노력없이 자동으로 추가하여 프로세스 단위의 오프로딩 가능하도록 결정한다. 이러한 접근방법은 Clonecloud에서 어플리케이션 개발자의 추가적인 소스변경 부담없이 오프로딩 단위를 결정하는 장점을 기반으로 하고 있다. 따라서 프로세스 단위의 오프로딩을 어플리케이션 개발자가 직접 소스코드에 지정하지 않고 컴파일러 정적 코드분석 기술을 응용하여 탐지한 후 오프로딩에 적합하게 소스코드를 수정하며, 이 과정에서 오프로딩된 단위는 로컬지역 하드웨어 정보를 이용할 수 없고 로컬 디바이스와 서버간에 실시간으로 동기화할 수 없는 상황을 고려하여 프로파일링 정보 결정하는 도전내용을 가지고 있다.

정적 코드 분석기는 실제로 안드로이드 ART에서 제공하는 내용을 (그림 2)와 같이 “Dex Static Analyzer”를 구현하여 통해 Dex 파일 또는 리소스&네이티브 코드를 dex2oat의 실행가능한 형식으로 변환시키다.



(그림 2) 오프로딩 마커 처리를 위한 ART 변형전(a), 변경후(b, Dex2Static Analyzer 추가)

어플리케이션 이전 코디네이터는 가장 최적의 오프로딩 서버를 탐색하고 선택하여 연결을 설정한다. 이 과정에서 선택된 오프로딩 서버는 기본적으로 ID, IP를 이용하여 모바일 디바이스의 코디네이터와 연결을 설정하도록 [그림 10]과 같이 설계하였다. 또한 코디네이터는 내부에 에너지 소비 프로파일링을 위해 오프로딩 서버의 에너지 소비 프로파일링을 위한 연결을 설정하고 프로파일링 정보 및 에너지 소비 측정 결과를 오프로딩 서버로부터 전달 받을 수 있는 핵심모듈을 내장하고 있다.



(그림 3) 모바일 사이드와 오프로딩 서버간 프로세스 오프로딩 설정

4. 결론 및 향후연구 내용

본 논문은 안드로이드 실행시간 환경이 네이티브 어플리케이션을 지원하는 ART로 완전하게 전환되는 상황에서 DVM에서 실행되고 있는 모바일 레거시 어플리케이션에 대해 모바일 디바이스의 복잡한 계산 부담을 줄여 실행속도를 향상시고, 이를 통해 배터리 소모를 감소시키는 프로세스 단위 오프로딩 프레임워크에 대한 설계를 완료하여 구체적인 구현을 진행중이다.

참고문헌

[1] Ben Cheng; Bill Buzbee, ″A JIT Compiler for Android′s Dalvik VM″ Google. Retrieved, 2015.

[2] Sean Buckley, ″′ART′ experiment in Android KitKat improves battery life and speeds up apps″. Engadget Retrieved, 2014.

[3] E. Y. Chen and M. Itoh, “Virtual smartphone over IP,” in World of Wireless Mobile and Multimedia Networks (WoWMoM), 2010.

[4] Eduardo Cuervo, et atl., “MAUI: Making Smartphones Last Longer with Code Offload”, International Conference on Mobile Systems, Applications, and Services(MobiSys ’10), 2010.

[5] Byung-Gon Chun, et al, “CloneCloud: Elastic Execution between Mobile Device and Cloud”, International Conference on Mobile Systems, Applications, and Services(MobiSys ’11), 2011.

[6] M. S. Gordon, D. A. Jamshidi, S. A. Mahlke, Z. M. Mao, and X. Chen, “COMET: Code Offload by Migrating Execution Transparently”, in 10th USENIX Symposium on Operating Systems Design and Implementation, 2012.