모바일 앱의 성능향상을 위한 동적 오프로딩 기법

박수석, 라현정, 김수동
숭실대학교 컴퓨터학부
e-mail : parksusuk@naver.com, {hjla80, sdkim777}@gmail.com

Dynamic Offloading Scheme for Improving Performance of Mobile Applications

Su Seok Park, Hyun Jung La, Soo Dong Kim
Dept. of Computer Science, Soong-Sil University

요 약

모바일 컴퓨팅의 보편화에 따라 엔터프라이즈 모바일 컴퓨팅을 위한 복잡한 모바일 어플리케이션의 사용이 점점 요구된다. 그러나, 모바일 디바이스는 제한된 자원을 가지므로 기능도가 복잡한 어플리케이션을 실행하는 것은 어렵다. 본 연구에서는 안드로이드 모바일 디바이스를 이용하여 현재 자원 상태에 따라 일부 기능을 외부 서버로 오프로드 하여 복잡한 어플리케이션 실행을 가능하게 하는 실용적이고 구현 가능한 프레임워크를 제안한다.

1. 서론

모바일 디바이스가 보편화됨에 따라, 모바일 디바이스는 다양한 소프트웨어 어플리케이션을 설치·운영하는 모바일 컴퓨팅 기능을 제공하고 있다. [1]. 모바일 디바이스는 높은 이동성, 휴대성, 풍부한 네트워크 연결성, 컨텍스트 감지 기능 등의 다양한 장점을 가지고 있지만, 휴대를 목적으로 하여 크기가 작기 때문에 전통적인 컴퓨터에 비해 CPU, 메모리, 저장공간 등의 컴퓨팅 자원이 부족하다. 그 결과 자원 소비가 많고 높은 복잡도를 가진 어플리케이션은 모바일 디바이스에서 설치, 운영되기 어렵다. 동적 오프로딩은 실시간에 어플리케이션의 기능 일부를 다른 컴퓨터로 오프로드 하고, 해당 컴퓨터에서 그 기능을 실행하게 하는 기법이다 [2]. 동적 오프로딩 기법을 모바일 컴퓨팅 환경에 적용하면, 제한된 자원으로 다소 복잡한 기능을 수행하는 어플리케이션을 실행할 수 있다 [3]. 그러나, 기존 동적 오프로딩에 대한 연구는 개념적인 수준에 머물러 있어, 모바일 환경 특히 안드로이드 기반 모바일 어플리케이션에 적용하는 것이 어렵다. 그러므로, 본 연구에서는 안드로이드 모바일 디바이스에서 실행되는 어플리케이션의 일부 복잡한 기능을 실시간에 서버로 오프로드하는 프레임워크를 제안한다. 3 장에서는 동적 오프로딩 기법의 주요 개념을 설명하고, 4 장에서는 동적 오프로딩 프레임워크의 아키텍처 및 주요 설계 모델을 제안한다. 그리고, 5 장과 6 장에서는 구현 소스 코드 및 실험 결과를 보여준다.

2. 관련연구

Yang 의 연구에서는 실시간에 오프로딩을 지원하는 시스템 아키텍처를 제안하였다[3]. 아키텍처는 모니터링 엔진과 오프로딩 결정 엔진으로 구성되며, 오프로딩 요청을 받았을 때, 시스템은 적절한 서비스를 찾아서 그것들을 결합시킨다. Li 의 연구에서는 모바일 디바이스에서 실행되는 기능을 모바일 서비스로 배포하는 미들웨어인 Mobile Cloud Framework (MCF)를 제안하였다 [4]. MCF 는 리소스를 관리하는 서비스, 서비스를 관리하는 서비스, 모바일 서비스를 구동시키는 서비스로 구성된다. 또한 서비스를 재구성하는 핵심 기법을 제시하였다. Ye 의 연구는 서비스 품질과 모바일 디바이스 전력 소모량을 관리하기 위해 오프로딩을 지원하는 프레임워크를 제안하였다 [5]. 프레임워크는 Qos 프로파일과 사용자의 프로파일일 이용하여 오프로딩 여부를 결정하고, 서비스 마이그레이션 기술을 이용하여 모바일 환경을 새롭게 구성한다. Huerta-Canepa 의 연구는 제한된 자원의 모바일 디바이스를 위한 가상 클라우드 컴퓨팅 환경을 제안하였다[6]. 이 환경은 오프로딩을 위해 여러 노드들이 자원을 공유할 수 있도록 설계되었다.

위의 연구들은 모바일 환경에서 동적 오프로딩을 지원하는 전반적인 아키텍처 및 프레임워크, 주요 컴포넌트 등을 제안하였지만, 이를 실현하기 위한 구체적인 설계 모델 및 구현 기법은 제안하지 않았다.

3. 동적 오프로딩

동적 오프로딩이란 특정 노드의 한정된 자원을 극복하기 위한 해결책 중 하나이다. 실행시간에 특정 노드에서 처리해야 할 작업을 자원이 풍부한 다른 노드로 전송하여 대신 처리하게 하고, 그 결과값을 받아 작업을 진행한다. 그림 1 은 동적 오프로딩이 일어나는 흐름을 보여준다.

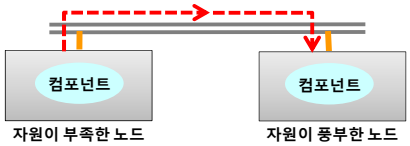

그림 1. 동적 오프로딩 구조

동적 오프로딩을 사용하여 얻을 수 있는 이점은 다음과 같다. 첫째, 제한된 자원을 가지는 노드에 대해서, 복잡도가 높은 어플리케이션을 실행시킬 수 있다. 즉, 특정 노드의 자원으로 실행하기 어려운 실행단위를 자원이 풍부한 다른 노드로 전송하여 해당 작업을 처리할 수 있다. 둘째, 어플리케이션의 실행시간을 단축할 수 있다. 복잡한 기능단위를 풍부한 자원을 가진 특정 노드에서 실행시켜 전반적인 어플리케이션 실행속도 향상을 기대할 수 있다.

4. 동적 오프로딩 아키텍처 및 설계 모델

본 장에서는 동적 오프로딩 기법을 구현한 프레임워크에 대한 아키텍처 및 설계 모델을 보여준다.

4.1. 아키텍처

동적 오프로딩 프레임워크는 그림 2 과 같이 Agent.Mobile, Offloadable 모바일 어플리케이션, Agent.DB, Offloading Manager, Server.DB 로 구성된다.

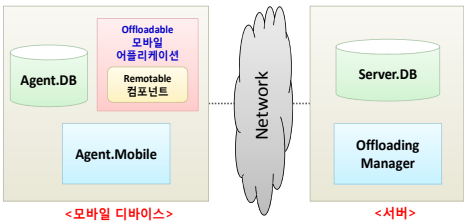

그림 2. 동적 오프로딩 프레임워크 아키텍처

Agent.Mobile 은 각 모바일 디바이스에 설치되어, 동적 오프로딩 관련 작업인 모바일 디바이스의 현재 자원 상태 측정, 오프로딩 여부 결정, Remotable 컴포넌트 전송 등을 수행한다.

Offloadable 모바일 어플리케이션은 이 프레임워크에서 오프로딩이 가능하도록 설계된 어플리케이션으로, Remotable 컴포넌트, Non-Remotable 컴포넌트, Manifest 파일로 구성된다. Remotable 컴포넌트는 서버로 오프로드가 가능한 기능을 가지고 있으며, 순수한 자바로 구현된다. Non-Remotable 컴포넌트는 안드로이드 컴포넌트만[7]으로 구현되어 외부 서버로 오프로드 될 수 없다. Manifest 파일은 동적 오프로딩 수행에 필요한 Offloadable 어플리케이션과 Remotable 정보를 XML 형태로 기술한다.

Agent.DB 는 Offloadable 모바일 어플리케이션, Remotable 컴포넌트 정보, 해당 모바일 디바이스에서 일어난 Offloading 로그 정보를 저장한다.

Offloading Manager 는 서버 노드에 설치되어, 오프로드된 Remotable 컴포넌트와 데이터를 받아 서버에 설치 및 실행하여, 결과를 해당 Agent.Mobile 로 전송하는 역할을 수행한다.

Server.DB 는 서버 측과 통신하는 Agent.Mobile 정보, Offloadable 모바일 어플리케이션 정보, 전송된 Remotable 컴포넌트 정보, 전체 프레임워크에서 발생한 Offloading 로그 정보를 저장한다.

4.2. 설계 모델

Agent.Mobile 은 MVC 패턴을 적용하여 안드로이드 환경에서 실행 되도록 설계되었다. 그림 3 은 Agent.Mobile 의 구조를 보여주는 클래스 다이어그램이다.

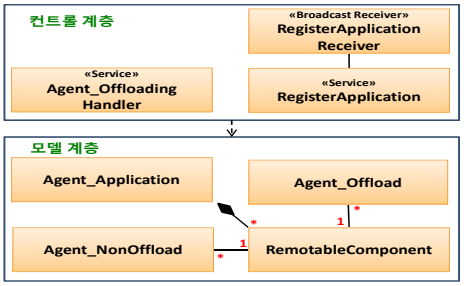

그림 3 . Agent.Mobile 클래스 다이어그램

컨트롤 계층은 3 개의 클래스로 구성된다. **Agent_OffloadingHandler** 는 오프로드 기능을 담당하는 컨트롤러로 안드로이드의 Service 를 이용하도록 설계된다. **RegisterApplicationReceiver** 는 어플리케이션 설치시 발생하는 브로드캐스트를 받기 위한 클래스이다. **RegisterApplication** 은 RegisterApplicationReceiver 에 의해 호출되며, 오프로딩이 가능한 어플리케이션의 정보를 Agent.DB 에 저장하기 위한 클래스이다. 안드로이드의 *Service* 를 사용한다.

모델 계층은 어플리케이션 정보(Agent_Application), 어플리케이션에 포함된 오프로딩 가능한 클래스 정보(RemotableComponent), 오프로딩 과거기록(Agent_Offload), 모바일 디바이스에서 실행된 과거기록(Agent_NonOffload)을 관리하는 클래스로 구성된다.

Offloading Manager 는 JVM 에서 실행되며, 서버에서 오프로드를 관리하도록 그림 4 와 같이 설계되었다.

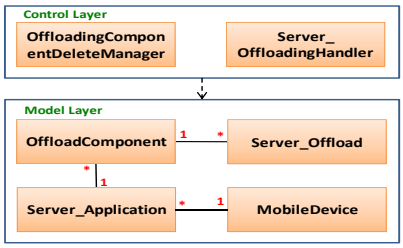

그림 4. Offloading Manager 클래스 다이어그램

컨트롤 계층은 Agent.Mobile 에서 전송한 Remotable 컴포넌트를 받아 실행하고, 그 결과값을 해당 Agent.Mobile 에 전송하는 컨트롤러인 **Server_OffloadingHandler** 와 더이상 사용하지 않는 오프로드된 컴포넌트를 삭제하는 **Server_OffloadingCoomponentDeleteManager** 로 구성된다. **모델 계층**은 서버로 오프로드된 어플리케이션과 컴포넌트 정보 (Server_Application, OffloadComponent), 어플리케이션이 설치되었던 모바일 디바이스 정보(MobileDevice), 서버로 전송된 오프로드 기록(Server_Offload)을 관리하는 클래스로 구성

된다.

그림 5 는 실시간에 자원을 측정하여 Agent.Mobile 이 오프로드 여부를 결정한 후, Remotable 컴포넌트를 Offloading Manager 로 전송 과정을 보여주는 시퀀스 다이어그램이다. 이 과정은 사용자의 직접적인 입력 없이 이루어진다.

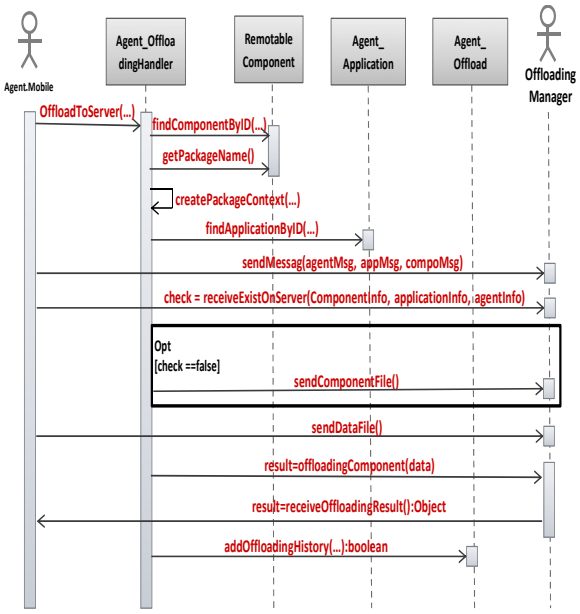

그림 5. 오프로드 과정을 다루는 시퀀스 다이어그램

오프로드가 결정되면 해당 Remotable 컴포넌트 정보, 어플리케이션 정보를 DB 로부터 가져와 Offloading Manager 로 보낸다. 이는 서버에 Remotable 컴포넌트가 존재하는지 확인하기 위함이다. 서버에 동일한 Remotable 컴포넌트가 있다면 컴포넌트의 데이터 파일만 보낸다. 서버로부터 온 결과값을 전달하고 오프로딩 과거 기록을 저장한다.

5. 구현

5.1. 모바일 에이전트 (Agent.Mobile) 구현

본 절에서 안드로이드 2.3 API 를 사용하여 모바일 에이전트(Agent.Mobile)의 주요 기능인 “모바일 어플리케이션 등록”과 “Remotable 컴포넌트 전송”에 대한 구현 결과를 보여준다.

동적 오프로딩이 가능한 어플리케이션을 설치할 경우, 설치된 어플리케이션 정보는 에이전트에 등록된다. 목록 1 은 이에 대한 소스코드 일부를 보여준다.

목록 1. 오프로딩이 가능한 어플리케이션 등록

```
1.  public class RegisterApplicationReceiver extends BroadcastReceiver{
2.      public void onReceive(Context context, Intent intent) {
3.          //’PACKAGE_ADDED’ Broadcasting인 경우
4.          if(intent.getAction().equals("android.intent.action.
    PACKAGE_ADDED")){
5.              //Intent에서 Package name 추출
6.              String packageName = intent.getDataString().substring(8);
7.              Intent registerAppInfo_Intent =
                new Intent(context,RegisterApplication.class);
8.              registerAppInfo_Intent.putExtra("packageName",
    packageName);
9.              //DB 저장 위해 서비스를 호출
10.             context.startService(registerAppInfo_Intent);
11.         }
12.….}
```

4 번째 줄에서 모바일 디바이스에 새로운 어플리케이션 설치할 때 발생하는 ‘PACKAGE_ADDED’ 를 감지한다. 그리고, 6 번째 줄에서 패키지 이름을 추출하여, 10 번째 줄에서 Agent.DB 에 어플리케이션과 관련된 Remotable 컴포넌트 정보를 저장하기 위해 RegisterApplication 서비스를 호출한다.

목록 2 는 서버로 Remotable 컴포넌트를 전송하는 소스코드이며, 이는 그림 5 의 순서와 동일하다.

목록 2. 오프로딩이 가능한 어플리케이션 등록

```
1.  public String offloadToServer(int serverID, int componentID,
        Bundle data) throws ClassNotFoundException{
2.      …
3.      ObjectOutputStream oOut = new ObjectOutputStream(out);
4.      ComponentMessage compoMsg
            =this.sendMessage(oOut,in,componentID);//메시지 전송
5.      // 동일한 컴포넌트가 있는지 서버 측에 확인
6.      Boolean componentExistCheck=dIn.readBoolean();
7.      //서버에 컴포넌트가 없으면, 해당 컴포넌트 파일 전송
8.      if(componentExistCheck == false) {
9.          this.sendComponentFile(manager,compoMsg,out,in);
10.     }
11.     this.sendDataFile(manager, compoMsg, out, in);//관련데이터전송
12.     …
13.     Object obj = oIn.readObject(); // 서버에서 실행된 결과 받음.
14.….}
```

Agent.Mobile 은 사용자가 오프로딩과 관련된 여러 옵션들을 설정하고, 오프로딩 결과를 보여주는 화면을 제공한다. 오프로딩 관련 옵션은 오프로딩 결정 여부에 사용되는 모바일 디바이스 자원 임계치, 어플리케이션 별 오프로드 가능 여부 결정이 있다. 그림 6 은 오프로딩 기록을 보여주는 Mobile.Agent 의 사용자 인터페이스의 구현 화면이다.

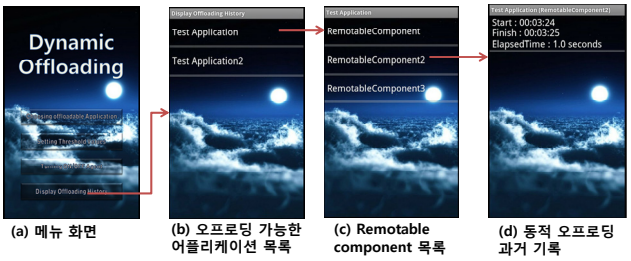

그림 6. Agent.Mobile 구현 화면

5.2. 서버 (OffloadingManager) 구현

서버 측 OffloadingManager 의 주요 기능인 “Remotable 컴포넌트 설치 및 실행”에 대한 구현 결과를 보여준다. 목록 3 은 이에 대한 소스코드이다.

목록 3. Remotable 컴포넌트 설치 및 실행

```
1.  private Object executeComponent(MobileMessage agentMsg,
        ApplicationMessage appMsg, ComponentMessage compoMsg,
        Object data) throws IOException {
        . . .  //Remotable 컴포넌트 파일 객체를 생성한다.
2.      ClassLoader cl = new URLClassLoader(urls);
3.      // 컴포넌트 클래스 로드함.
4.      Class<?> cls =
    Class.forName(compoMsg.packageName+"."+compoMsg.component
    Name, true, cl);
5.      //로드된 클래스에 정의된 생성자 정보 읽음.
6.      Constructor<?> constructor1 = cls.getConstructor(new
    Class[]{});
```

```
7.        //생성자 이용하여 객체 생성
8.        Object object1 = constructor1.newInstance();
9.        // 호출할 메소드 정보 읽음.
10.       Method method = cls.getMethod(compoMsg.methodName, new
    Class[] {Object.class});
11.       // Remotable 컴포넌트 실행
12.       Object resultObj = method.invoke(object1, data);
13.       return resultObj;
14.…}
```

서버에는 모바일 어플리케이션이 사전에 설치되어 있지 않고, Agent.Mobile 에서 특정 기능을 오프로드 할 때에 즉시 설치된다. 그러므로, 실시간에 전송된 클래스 파일을 실행하기 위하여, 2 번째 줄에서 URLClassLoader 를 이용하였다.

그리고, OffloadingManager 는 오프로드된 Remotable 컴포넌트의 정보를 사전에 알지 못하기 때문에, 이 정보를 실시간에 얻어오기 위해 Java Reflection 을 사용하였다. 6 번째 줄에서 'Class.forName'으로 Remotable 컴포넌트 클래스를 로드하고, 10 번째 줄에서는 Remotable 컴포넌트 클래스의 객체를 생성하며, 12 번째 줄에서 Remotable 컴포넌트 객체에서 메소드를 추출한다. 최종적으로 14 번째 줄에서 추출한 함수를 데이터 값과 함께 실행하고 결과값을 얻는다.

6. 동적 오프로딩 기법을 이용한 실험

본 절에서는 동적 오프로딩이 일어났을 때와 모바일 장비에서 컴포넌트가 실행되었을 때 실행시간의 차이를 비교하여 동적 오프로딩의 장점을 증명한다. 실험을 위해 사용한 모바일장비는 삼성전자의 GALAXY TAB 이며, 서버는 Windows7(64bit), CPU(Intel Core2 Quad Q9400, 2.66GHZ), RAM(4G,DDR3)를 사용하였다. 테스트 어플리케이션은 하노이 탑의 판 개수를 입력받아, 판의 이동 횟수를 계산한다.

그림 7 은 판의 개수를 21 부터 시작하여 2 씩 증가함으로써 동적 오프로딩이 수행된 경우와 그렇지 않은 경우의 실행시간을 비교한 그래프이다.

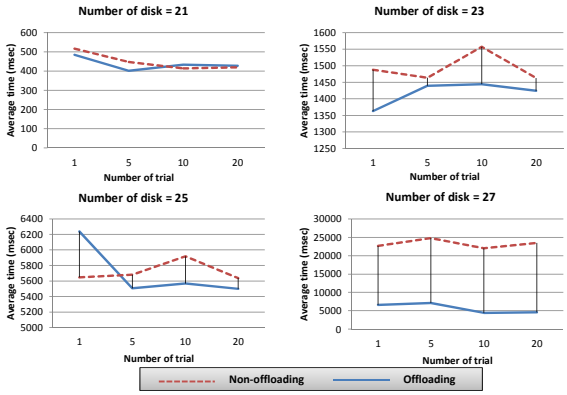

그림 7. 실험 결과

판의 개수가 비교적 적을 때에는 오프로딩이 수행된 경우와 그렇지 않은 경우의 수행시간이 비슷하였지만, 판의 개수가 늘어날수록 오프로딩한 경우의 실행시간이 적게 걸리며, 오프로딩 되지 않은 경우와의 수행 속도차이가 증가하는 것을 확인할 수 있다. 오프로드된 컴포넌트의 복잡도가 증가할 수록 동적 오프로딩이 컴포넌트의 수행속도 향상에 큰 영향을 미치는 것을 확인할 수 있다.

7. 결론

모바일 디바이스는 다양한 어플리케이션 실행의 편리함을 제공하지만, 메모리나 CPU 속도등 컴퓨팅 자원이 부족하다. 또한 엔터프라이즈 모바일 컴퓨팅을 위한 복잡한 모바일 어플리케이션의 사용이 점점 요구되어, 모바일 디바이스의 컴퓨팅 자원 부족문제는 더욱 대두될 것이다. 본 논문에서는 모바일 디바이스의 자원 제약사항을 해결하기 위해 모바일에 동적 오프로딩을 적용하였다. 기존의 개념적인 수준에 머물러 있는 동적 오프로딩에 대한 연구를 안드로이드 모바일 디바이스에서 실행되는 어플리케이션의 일부 복잡한 기능을 실시간에 서버로 오프로드하는 프레임워크의 설계를 보여주었다. 또한 설계에 따라 구현한 동적 오프로딩 프레임워크의 실험 결과를 통해 모바일 동적 오프로딩의 실효성을 보여주었다.

Acknowledgement

이 논문은 정보통신산업진흥원의 SW 공학 요소기술 연구개발사업에 의해 지원되었음을 밝힙니다. 이 논문은 2009 년 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구입니다. (No. 2009-0076392)

참고문헌

[1] König-Ries, B. and Jena, F., "Challenges in Mobile Application Development," *it-Information Technology*, Vol. 52, No. 2, pp. 69-71, 2009.

[2] Malek, S., et al., "An Architecture-driven Software Mobility Framework," *The Journal of Systems and Software*, Vol. 83, pp. 972-989, 2010.

[3] Yang, K., Ou, S., and Chen, H.H., "On Effective Offloading Services for Resource-Constrained Mobile Devices Running Heavier Mobile Internet Applications," *IEEE Communications Magazine*, Vol. 46, No. 1, pp. 56-63, 2008.

[4] Li, X., Zhang, H., and Zhang, Y., "Deploying Mobile Computation in Cloud Service," In Proceedings of 1st International Conference on Cloud Computing (CloudCom 2009), Lecture Notes in Computer Science (LNCS) 5931, pp. 301-311, 2009.

[5] Ye, Y, Jain, N., Xia, L., Joshi, S., Yen, I., Bastani, F., Cureton, K.L., and Bowler, M.K. "A Framework for QoS and Power Management in a Service Cloud environment with Mobile Devices," *In Proceedings of the 5th IEEE International Symposium on Service Oriented System Engineering (SOSE 2010)*, pp. 236-343, 2010.

[6] Huerta-Canepa, G. and Lee, D., "A Virtual Cloud Computing Provider for Mobile Devices," In Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond (MCS 2010), Article No. 6, 2010.

[7] Android Developers, http://developer.android.com/index.html (accessed Sep. 15, 2011)