안드로이드 시스템에서의 zRam 적용을 통한 성능분석

임상섭[O*], 이치영[**], 유혁[**]
*고려대학교 컴퓨터정보통신대학원 소프트웨어공학과
**고려대학교 컴퓨터 전파통신 공학과
sangsub@korea.ac.kr, cylee@os.korea.ac.kr, hxy@korea.ac.kr

Performance study through zRam enabled in Android system

Sangsub Lim[O*], Chiyoung Lee[**], Chuck Yoo[**]
*Dept. of Software Engineering, Graduate School of Computer Information & Communication, Korea University
**Dept. of Computer and Radio Communications Engineering, Korea University

요 약

안드로이드 시스템은 임베디드 특성에 따라 리눅스에서 기본적으로 제공하는 스왑(Swap) 기능을 지원하지 않고 있다. 임베디드 시스템에서는 NAND 플래시 메모리를 외부 저장장치로 사용함에 따라 속도적인 측면과 마모도 평준화의 문제로 스왑영역을 설정하는데 문제가 있다. 본 논문에서는 zRam을 사용하여 외부 저장 매체가 없이 RAM의 일정 부분을 압축영역으로 설정하여 스왑과 같이 추가적인 가용 메모리를 확보할 수 있는 것을 소개하며 Single 및 Multi Task 환경에서 기존 none-Swap 상태와의 비교를 통한 성능 분석 결과를 보인다.

1. 서 론

안드로이드(Android) 플랫폼은 세계적 검색엔진 업체 구글(Google)사와 세계 각국의 이동통신 관련회사인 OHA(Open Handset Alliance)이 연합하여 2007년 11월에 공개되었다. 2012년 스마트폰 연간 점유율 통계결과 68.4%를 차지하며 전년도에 비해 20%가량 상승하며 현재 가장 대중화된 스마트폰 OS 이다[1].

스마트 폰 운영체제는 과거의 피쳐 폰과 달리 웹 브라우저를 통한 인터넷이나 온라인 게임 등과 같이 heavy content들이 실행되고 멀티태스킹(Multi-tasking) 역시 가능 하게 한다. 이에 따라, 제한된 용량의 메모리를 가진 임베디드 시스템에서는 메인 메모리보다 큰 메모리 영역을 제공해야 하는 경우가 많은데, 이를 위해 가상 메모리 기법이 사용된다. 안드로이드 시스템은 기본적으로 어플리케이션이 취소 키 등으로 백그라운드(Background)로 상태가 전이하여도 해당 프로세스를 제거하지 않는다. 이는 사용자에 의해 다시 실행 될 때의 로딩속도 향상을 위하여 이러한 방향성을 갖는다. 따라서 백그라운드 어플리케이션들이 메모리를 점유하기 때문에 메모리 부족 문제를 야기한다.

본 연구는 이러한 메모리 부족 문제를 해결하기 위한 안드로이드 Framework Layer에서의 메모리 관리 방법들에 대해서 살펴보고 zRam[2]에 대한 설명과 적용을 통한 확장된 물리 메모리에서의 다수의 어플리케이션 동작 시의 성능분석 내용과 이에 대한 결론 및 향후 연구 주제에 대한 방향을 제시한다.

2. 안드로이드 메모리 관리 기법

메모리 관리 측면에서 안드로이드 플랫폼은 크게 3부분으로 나뉘며 각 Layer들에서의 메모리 관리 방법은 서로 상이하며, 관리 주체는 <표 1>과 같다

Layer	Manager
Application	Garbage Collector (GC)
Frameworks	Low memory Killer (LMK) & Activity Manager
Linux Kernel	Out of memory (OOM) Killer

표 1. Android Layer별 메모리 관리 주체

Garbage Collector는 Android 부팅과정에서 Init 프로세스에 의해 zygote 프로세스가 생성되고, GC Thread를 생성한다. 이후, 새로운 프로세스 생성이 요청될 때 마다 fork를 통해 GC thread를 상속받아 각 프로세스 별로 메모리 관리가 가능하게 된다. [3]

Linux Kernel의 기본 메모리 정책은 OOM Killer가 있다. 이는 메모리가 부족한 상황에서 최대한의 가용성을 확보하기 위해 가장 많은 메모리를 할당한 프로세스를 제거하도록 되어 있다. 따라서 현재 프로세스가 포그라운드(Foreground)에서 실행 중인지에 대한 상태 체크가 없으므로 실행중인 어플리케이션이 가장 Heavy할 경우 process kill을 당하게 되는 문제가 있다.

이러한 리눅스 커널의 OOM Killer의 문제를 해결하기 위해, Arve Hjønnevåg는 LMK를 제안하였다. LMK는 각 프로세스의 상태를 나타내는 adjust 값과, 이에 매핑되는 메모리 하한선인 minfree 값을 시스템 내에서 지정하여, 가용 메모리가 minfree값 이하가 되면, 해당

adjust값에 해당되는 어플리케이션 들을 제거하게 된다. 그 외에 메모리가 부족한 상황이 아니더라도 framework layer 에서는 ActivityManger에 의해서 백그라운드 상태로 존재하는 프로세스(시스템 프로세스 제외)가 제한되어, 정해진 수치(기본값 12) 이상이 되면 adjust값이 큰 순서에 따라 process kill을 수행한다.

3. zRam

zRam은 기존까지 "compcache"라고 불렸으며, 리눅스 커널 2.6.33버전부터 staging단계로 포함된 모듈이다.이는 외부 저장장치를 사용하지 않고, Block device 형태로 주 기억장치인 RAM상의 일정 영역 지정을 통해 메모리가 부족 할 경우 메모리를 압축하여 보존하는 방식으로 스왑과 유사한 기능을 하는 가상메모리 기법 이다.

임베디드 환경에서는 보조기억 장치를 NAND 플래시를 사용함에 따라 PC상에서 사용되는 스왑방식을 지원하는데 있어 제약사항이 존재한다. NAND 플래시는 EEPROM처럼 이미 쓰여진 공간에 덮어쓰기를 할 수 없다. 따라서 이를 위해서는 삭제 연산이 선행되어야 한다. 일반적인 NAND 플래시의 읽기 연산의 수행시간은 20us, 쓰기 연산의 수행시간 은 200us, 삭제 연산의 수행시간은 1.5ms로써 페이징 처리가 빈번해 질수록 수행시간이 길어질 수 있는 문제점을 가지고 있고, 각 연산 수행시간의 비 대칭성 이라는 특성이 존재한다[4]. 그리고 각 블록당 제한된 숫자의 삭제 연산횟수가 정해져 있기 때문에 플래시 메모리를 위한 마모도 평준화(Wear leveling)가 고려 되어야 한다. 이러한 이유로 Android system에서는 스왑을 지원하지 않는다.

본 연구에서는 zRam을 통하여 스왑을 활성화 하기 위해 compcache-0.6.2[5] 버전을 포팅하여 진행하고 4장에서 이에 대한 성능 분석을 한다.

4. 실험 결과 및 성능 평가

4.1 평가 환경

본 논문의 실험에서는 안드로이드 레퍼런스 모델인 삼성 nexus S로 하였으며 기기의 사양은 다음과 같다.

Processor	1GHz CortexA8 (Hummingbird)
RAM/NAND	512MB / 16384 MB
OS Version	Linux-2.6.35.7 with Android 2.3.7

표 2. 평가 환경

성능평가는 싱글 태스크(Single Task)와 멀티 태스크(Multi Task)로 크게 두 분류로 나눴으며 세부적으로 zRam을 적용하였을 경우 와 NAND 플래시에 스왑을 적용하였을 경우, 그리고 LMK만 사용하는 안드로이드 기본 상태의 경우로 분류 하여 메모리 변화 추이 및 로딩 시간을 측정하였다.

4.2 싱글 태스크

부팅 이후, 전체 가용메모리는 353,280KB이며 zRam의 크기는 전체 가용 램의 15%인 52,984KB로 지정하였다. 그리고 스왑은 전체 RAM 사이즈의 두 배인 1GB로 할당하였다. <그림 1>은 하나의 어플리케이션에서 OOM Exception이 발생되기 전까지 Heap 영역에 메모리 할당을 계속한 결과이며, zRam이나 스왑을 통해서 싱글 태스크에서 늘어난 크기만큼의 추가적인 가용 메모리를 확보할 수 있음을 보여준다.

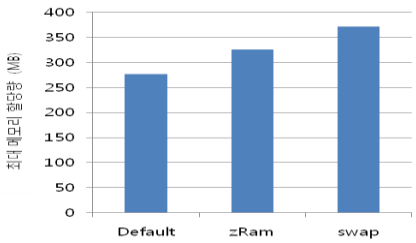

그림 1. Single Task 최대 메모리 할당량

4.3 멀티 태스크

멀티 태스크 실험은 싱글 태스크와 같은 환경에서 진행하였으며, 동일한 조건으로 테스트하기 위해 Android-SDK에서 제공하는 Monkey Test 툴을 이용하여 15개의 어플리케이션을 선택하여 실제로 사용자가 사용하는 것처럼 하기 위하여 스크립트를 작성하였다. 그리고 메모리의 극빈상황에서 LMK 동작에 따른 변화 추이 만을 확인하기 위하여, 백그라운드 어플리케이션의 개수(Hidden Apps)를 30개로 수정하였다.

NAND 플래시에 1GB로 스왑영역을 적용하였을 때, 어플리케이션이 차례로 수행되는 과정에서 스왑-아웃(Swap-out)이 지속적으로 요청 됨에 따라, ANR(Application Not Responding)이 발생하여 테스트가 불가능 하였다. 이는 비어있는 스왑 인덱스(Swap-index) 검색시간이 커짐에 따라 실제로 사용 하기에 반응성이 현격히 떨어짐을 확인하였다. 따라서 멀티 태스크 환경 에서는 zRam의 적용과 비 적용상태(LMK만 적용)에서의 비교 측정을 하였다.

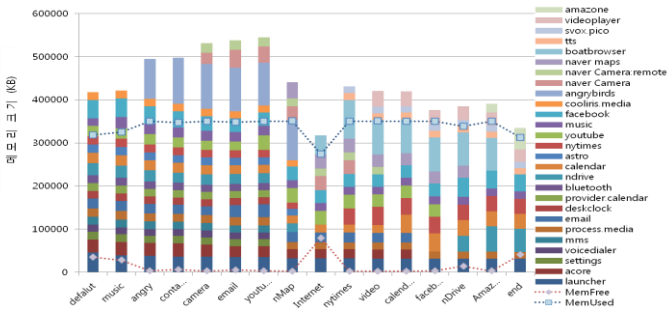

그림 2. 어플리케이션 실행 별 메모리 변화

<그림 2>는 zRam이 비활성화 되었을 때의 메모리 변화량과 각 프로세스들의 RSS(Resident Set Size)

측정값을 나타낸다. LMK에 의해서 어플리케이션이 종료되면 일시적으로 Memory Free영역이 늘어나는 것을 알 수 있었으며, 인터넷이나 위치정보 관련 어플리케이션과 같이 메모리 소모가 큰 어플리케이션이 실행되면 부족한 메모리 확보를 위해 LMK가 동작하는 것을 알 수 있다. 그리고 순차적으로 어플리케이션들이 실행되며 status에 따라 adjust 값이 변화되고, LMK 발생시 minfree를 기준으로 adjust 값과 각 프로세스의 RSS 크기를 비교하여 제거 할 프로세스 선정하고 종료처리 됨을 확인 하였다.

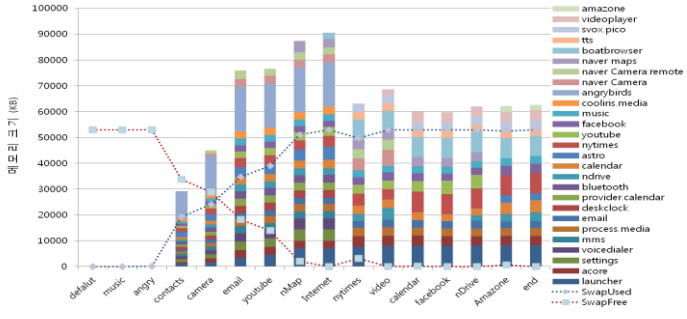

그림 3. zRam 적용을 통한 스왑 메모리 변화

<그림 3>은 zRam을 적용 후 각 프로세스들의 스왑 할당량 및 SwapUsed/SwapFree 수치를 나타낸다. MemFree의 가용량이 부족해 질 때부터 스왑-아웃이 발생하여 SwapUsed 수치가 증가되는 것을 확인하였으며, 반대로 LMK가 동작함에 따라 SwapFree 크기가 늘어가는 것을 알 수 있다.

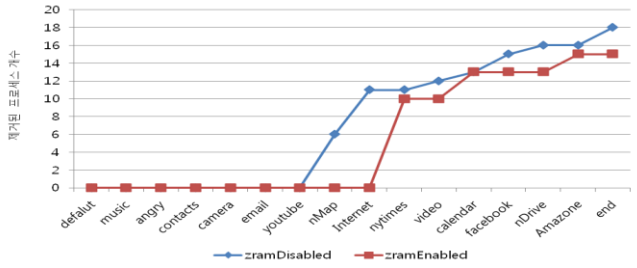

그림 4. zRam 적용에 따른 제거된 프로세스 수 비교

<그림 4>는 LMK에 의해 제거된 프로세서의 누적 개수를 나타낸다. zRam을 적용하였을 시 메모리 확장이 이뤄진 만큼 LMK가 늦게 호출되었으며, 비 적용한 것에 비하여 4회 덜 호출 되었음을 알 수 있다. 그리고 제거된 프로세스들의 개수에서도 3개의 프로세스가 백그라운드에 살아 있는 것을 보여준다. 최근의 High-end급 스마트 폰은 2G의 RAM이 탑재되어 출시됨에 따라 본 연구 결과보다 더 많은 어플리케이션이 존재할 수 있을 거라고 예상된다.

<그림 5>는 어플리케이션이 실행요청을 받은 뒤, ActivityManager를 통해 화면에 출력될 때까지의 시간을 측정한 것을 나타낸다. zRam이 활성화 되었을 시에 LMK가 메모리 확보를 위하여 프로세서를 제거하는 작업이 진행 될 시 시간소요가 많아지며, 측정 후반부로 갈수록 로딩 시간이 더 소요된다는 단점을 갖지만, ANR 발생과 같은 반응성 측면에서 큰 문제가 발견되지는 않았다.

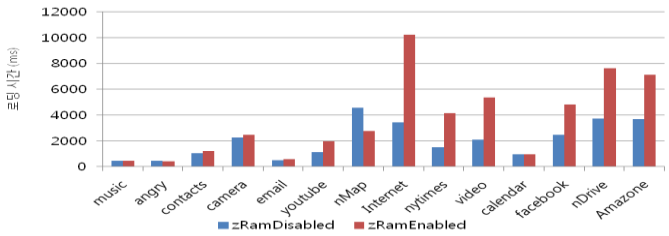

그림 5. zRam 적용에 따른 로딩시간 비교

4. 결론 및 향후 연구

본 논문에서는 zRam 적용을 통해서 안드로이드 시스템의 가용메모리 증대하는 방법에 대해서 소개하고 있다. zRam을 적용함에 따라 페이지 압축 소요시간과 RAM의 일정부분을 할당해야 하는 문제점을 가지고 있지만 추가적인 가용메모리를 확보할 수 있었으며, 기존의 NAND에 스왑을 추가하는 것에 비하여 높은 반응성을 갖는 것을 확인하였다. LMK는 RSS 값을 통해서 재거 될 프로세스(victim)를 선정한다. RSS의 수치는 여러 프로세스가 공통적으로 사용하는 라이브러리 등의 메모리 소모량도 포함한 크기이기 때문에 프로세스 고유의 크기라고 할 수 없다. 따라서 PSS(Proportional Set Size)의 비교를 통해서 개선될 것이라고 예상한다. PSS는 프로세스의 고유 메모리 사용량과 하나의 프로세스가 차지하는 공유 메모리 비율을 나타낸다. 따라서 RSS방식에 비하여 한번의 프로세스 제거할 시에 많은 메모리가 회수 될 수 있을 것이라 예상한다. 그 외에도 저장되는 Data들을 선별하여 가변 데이터가 아닌 상수 데이터들을 외부 저장장치에 backing_store로 보관하여 NAND 플래시의 rewrite시의 문제점과 Wear leveling 를 극복하면서 메모리 가용량을 효율적으로 분배될 수 있을 거라 생각되며, 이에 대한 추가적인 연구를 진행할 계획이다.

참 고 문 헌

[1] Scott Bicheno, “Global Smartphone OS Market share by Region: Q4 2012” http://www.strategyanalytics.com/default.aspx?mod=reportabstractviewer&a0=8222

[2] http://wiki.gentoo.org/wiki/Zram#Enabling_zram

[3] P. Dubroy, "Memory Management for Android Apps," Google I/O Development Conference, 2011.

[4] Seungjae Baek, Jongmoo Choi, "Linux Kernel Internal" , Sep, 2008, Kyohak Inc.

[5] http://code.google.com/p/compcache/