논문 2015-10-28

다중 코어 기반의 실시간 가상화 시스템을 위한 이종 운영체제 통합 성능 분석 방법에 관한 연구

(Heterogeneous Operating Systems Integrated Trace Method for Real-Time Virtualization Environment)

경 주 현, 한 인 규, 임 성 수*
(Joohyun Kyong, In-Kyu Han, Sung-Soo Lim)

Abstract : This paper describes a method that is integrated trace for real-time virtualization environment. This method has solved the problem that the performance trace may not be able to analyze integrated method between heterogeneous operating systems which is consists of real-time operating systems and general-purpose operating system. In order to solve this problem, we have attempted to reuse the performance analysis function in general-purpose operating system, thereby real-time operating systems can be analyzed along with general-operating system. Furthermore, we have implemented a prototype based on ARM Cortex-A15 dual-core processor. By using this integrated trace method, real-time system developers can be improved productivity and reliability of results on real-time virtualization environment.

Keywords : Integrated trace, Real-time virtualization, Hypervisor, Heterogeneous operating system, Workload consolidation

I. 서 론

임베디드 하드웨어 성능이 향상됨과 동시에 자동차, 공장 자동화 로봇, 의료, 항만 시스템 등 임베디드 시스템에 높은 수준의 사용자 인터페이스와 안정화된 실시간 소프트웨어가 동시에 요구되고 있다[1]. 따라서 이러한 요구사항을 만족시키기 위한 실시간 임베디드 가상화 기술에 대한 관심이 증가되고 있다.

최근 실시간 임베디드 가상화 기술을 적용하여, 범용 운영체제와 여러 실시간 운영체제를 하나의 시스템에 통합하는 방법이 주된 연구 관심사이다. 이처럼 여러 운영체제를 하나의 시스템에 통합하는 방법은 기존 검증된 소프트웨어를 수정 없이 하나의 시스템에 동작시킬 수 있으므로, 하드웨어적인 비용 절감을 가져온다. 뿐만 아니라, 고 신뢰성이 필요한 실시간 소프트웨어의 오류를 스스로 판단하고, 이를 복구하는 등의 일을 수행할 수 있는 장점을 지닌다. 또한 안드로이드와 같은 고품질의 사용자 인터페이스 구성이 가능한 운영체제를 각종 실시간 운영체제와 동시에 동작시킴으로서, 기존 실시간 운영체제들이 가지고 있는 사용자 인터페이스의 열약한 문제를 해결할 수 있는 장점이 있다.

이처럼 실시간 임베디드 가상화 기술은 서버에서 사용되는 서버 가상화 기술과 다르다. 보통 서버 가상화 기술에는 게스트 운영체제가 리눅스와 같은 범용 운영체제들로 구성되지만, 실시간 임베디드 가상화 기술은 리눅스와 같은 범용 운영체제 뿐만 아니라, 실시간성을 만족시키기 위해 다양한 종류의 실시간 운영체제들로 구성된다. 필요에 따라, 운영체제 없이 동작하는 펌웨어 수준의 게스트 소프트웨어로 구성되기도 한다. 하지만, 이처럼 이종 운영

*Corresponding Author (sslim@kookmin.ac.kr)
Received: 12 Feb. 2015, Revised: 31 Mar. 2015, Accepted: 8 June 2015.
J. Kyong, I.-K. Han, S.-S. Lim: Kookmin Univeristy
* 이 논문은 2015년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.B0101-15-0644, 매니코어 기반 초고성능 스케이러블 OS 기초연구).

ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2015.10.4.233

체제들로 구성된 실시간 임베디드 가상화 환경에서 하이퍼바이저와 게스트들이 함께 이루어지 성능을 분석 할 수 있는 방법이 없는 것이 문제이다.

본 연구는 이러한 문제점을 해결하기 위하여, 실시간 가상화 환경에서 통합적인 분석 방법을 제시하였다. 이는 성능 분석 기능을 하이퍼바이저 코드에 구현하지 않고, 기존 범용 운영체제에서 제공하는 공개소스 기반의 성능 분석 기법과 툴들을 재사용하는 방법을 사용하였다. 이러한 방법은 성능 분석 기능이 없거나, 고가의 라이선스 비용을 지불해야 사용 할 수 있는 실시간 운영체제의 성능 분석을 공개소스 기반의 툴로 수행 할 수 있는 장점을 가진다. 또한, 성능 분석 기능을 하이퍼바이저 코드에 구현하지 않고, 기존 범용 운영체제의 코드를 재사용함에 따라 하이퍼바이저의 TCB(Trusted Computing Base)을 최소한으로 유지시킬 수 있다. 동시에, 게스트도 역시 고객을 위해 특별한 코드를 삽입하지 않는 동적인 코드 삽입 방법을 사용함에 따라, 성능 측정을 위해 실시간 운영체제의 코드를 수정할 필요가 없는 장점을 가진다.

이처럼 본 연구는 공개 소프트웨어로 구성되어 있는 범용 운영체제의 성능 분석 기능을 재활용하도록 구현하였다. 이를 위해, 하이퍼바이저가 최소한의 서비스를 제공하도록, 동작 중인 게스트를 분석 할 수 있는 기능을 추가하였다. 비록 본 연구는 실시간 하이퍼바이저를 위한 내부 구조에 대한 연구는 아니지만, 본 연구에서 제안하는 방법은 향후 실시간 하이퍼바이저를 위한 스케줄러 개발, 또는 다른 내부 알고리즘 개발 결과물 검증과 개발 생산성을 위해 유용하게 이용될 수 있는 방법이다.

II. 관련 연구

가상화된 환경에서의 통합 성능 분석 방법에 대한 연구는 유닉스 계열의 게스트 운영체제를 사용하는 환경에서 진행되고 있다[2-4].

이러한 방법들은 그림 1에서 보이는 바와 같이, 여러 게스트 운영체제가 동작하는 환경에서 게스트 간의 문맥교환으로 인해 발생하는 기존 성능 분석 방법의 문제점을 해결하였다. 예를 들어 그림 2와 같이 각각 게스트 운영체제가 성능 분석을 한 후 로깅된 파일을 시간의 순서대로 통합하는 방법[2]이 있다. 또한 비슷한 방법으로 KVM위에 성능 분석도구인 LTTng을 사용하여 각각 게스트의 로깅된 파일을 시간 순으로 정렬하여 통합하는 방법[3]을

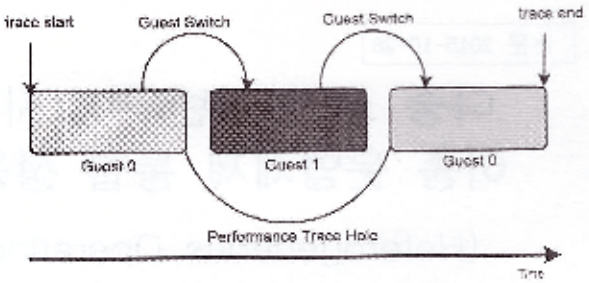


그림 1. 가상화 환경에서 성능 분석
Fig. 1 Performance analysis in virtualization environment

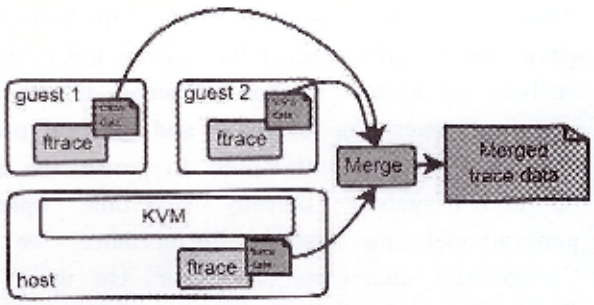


그림 2. ftrace를 이용한 통합 트레이스 기법
Fig. 2 Integrated trace method using the ftrace

사용하고 있다. 하지만 이러한 방법들은 문제가 있다. 반드시 게스트 운영체제들과 호스트 운영체제가 같은 분석 기법을 사용해야 한다. 따라서 이 방법은 이종 운영체제로 구성된 실시간 임베디드 가상화 환경에서는 적합하지 않다.

또한 기존 성능 분석 도구를 사용하지 않고, 새로운 분석도구를 제안하여 Xen 상의 게스트와 하이퍼바이저간의 스케줄링 상태를 분석하는 방법[4]이 있다. 그러나 이 방법은 모든 게스트 운영체제들을 수정해야하고, 측정 결과를 분석하기 위해, 기존에 개발된 고품질의 GUI를 사용하지 못한다는 문제점이 있다.

III. 본론

1. 이종 운영체제 통합 성능 분석을 위한 구조도

본 논문에서는, 이종 운영체제 환경에서 기존에 개발되어 온 고품질 분석도구를 사용하여, 통합 성능 분석을 위한 방법을 제안한다. 이를 위한 시스템 구조는 그림 3과 같다.

먼저 물리적인 하드웨어 위에 실시간 하이퍼바

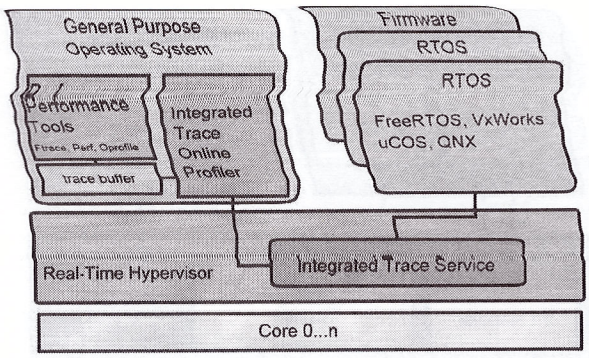


그림 3. 이종 운영체제 통합 성능 분석을 위한 구조도

Fig. 3 Structure of the integrated trace method

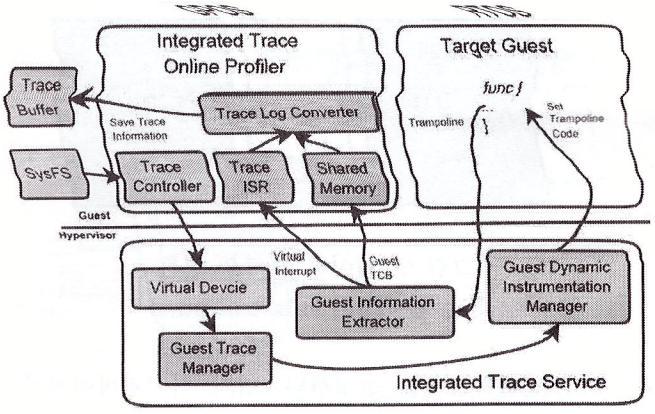


그림 4. 분석도구 내부 구조도

Fig. 4 Internal structure of the integrated trace

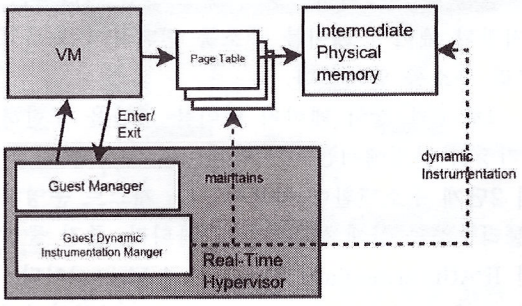


그림 5. 게스트 운영체제 주소 접근 방법

Fig. 5 Method of access to guest address

이저가 존재한다. 실시간 하이퍼바이저 내부에는 실시간 운영체제들의 성능을 분석하기 위한 통합 트레이스 서비스(Integrated Trace Service)가 존재한다. 하이퍼바이저 위에는 범용운영체제(General Purpose Operating System)가 있다. 범용 운영체제의 내부 모듈로는 각 실시간 운영체제의 분석 결과를 수집하기 위한, 통합 성능 분석 온라인 프로파일러(Integrated Trace Online Profiler)가 있다. 여기서 수집된 정보는 실시간 운영체제에서 수행한 성능 측정 결과이다. 통합 성능 분석 온라인 프로파일러는 수집된 성능 측정 결과를, 마치 범용 운영체제의 코드에서 수행한 정보인 것처럼 수정한다. 이렇게 수정된 정보는 공개 소스 성능 분석도구의 트레이스 버퍼에 저장된다. 결과적으로 본 연구에서 제안하는 구조는 로깅된 포맷이 같으므로, 범용 운영체제의 성능 분석 도구를 수정 없이 바로 사용할 수 있게 되는 장점을 가진다.

2. 분석도구 내부 구조

앞 절에서 설명한 분석도구의 내부 구조는 그림 4와 같다. 내부 구조는 크게 사용자 입력을 받아 일을 수행하는 부분과 점프코드 삽입을 통한 게스트를 로깅하는 부분 그리고 저장된 정보를 추출하여 분석도구 버퍼에 저장하는 부분으로 구성된다.

2.1. 사용자 인터페이스 처리

먼저 사용자 입력을 받아 일을 수행하는 부분이 있다. 사용자로부터 커널의 사용자 인터페이스인 sysfs 인터페이스를 통해 설정, 시작, 종료 등에 대한 입력을 받는다. 이러한 입력 정보는 트레이스 통제자(Trace Controller)를 통해 하이퍼바이저 인터페이스인 가상 디바이스(Virtual Device) 인터페이스를 거쳐 하이퍼바이저로 명령을 전달하게 된다. 전달되는 정보로는 로깅을 위한 게스트 코드의 주소, 로깅을 수행할 게스트의 아이디, 시작, 종료에 대한 시간 정보가 있다.

2.2. 게스트 로깅

분석도구의 내부구조 중 다른 하나는 점프코드 삽입을 통한 게스트를 로깅하는 부분이다. 관련된 모듈로는 그림 4의 게스트 트레이스 관리자(Guest Trace Manager)와 동적 코드 삽입 관리자(Guest Dynamic Instrumentation Manager)모듈이 존재한다. 사용자로부터 분석 코드 지점을 입력 받으면, 게스트 메모리 영역에 접근하여 점프코드를 삽입한다. 실제 구현은 게스트와 하이퍼바이저 서로간의 수행 권한이 다르므로, 하이퍼콜 명령어를 삽입하여 구현하였다.

이 때 수행되는 게스트와 하이퍼바이저간의 메모리는 다르게 관리가 된다. 따라서 모든 게스트의 주소에 접근하기 위해서는 주소변환이 필요하다. 그림 5는 동적 코드 삽입 관리자가 어떻게 게스트의 주

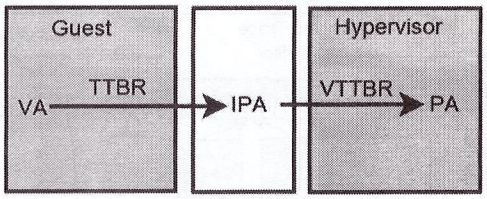


그림 6. 페이지 테이블 변환
Fig. 6 Page table transition

소를 변환하는지를 설명한다. 먼저 하이퍼바이저가 게스트 운영체제에 접근하기 위해서는 게스트의 페이지 테이블 정보를 알고 있어야 한다.

이 페이지 테이블 주소는 게스트 간의 문맥교환이 발생할 때 저장된다. 동적 코드 삽입 관리자는 이 때 저장된 페이지 테이블 주소를 변환하여 물리적인 메모리 주소를 알아낸다.

그림 6과 같이 페이지 테이블 변환을 구현하였다. 가상화 환경에서는 하이퍼바이저 계층이 있기 때문에 2단계 주소변환이 이루어진다. 게스트 운영체제의 물리주소는 실제 물리주소가 아니라, 중간 물리주소인 IPA(Intermediate Physical Address)이다. 따라서 실제 물리 주소를 얻기 위해서는 하이퍼바이저 계층에서 다시 변환해야 한다. 이 때 참조하는 레지스터는 가상 메모리 주소변환 테이블을 가리키는 레지스터인 VTTBR(Virtualization Translation Table Base Register)이다. 이와 같이 2단계 변환을 수행하여 물리주소를 얻는 방법을 구현하였다.

다음으로 동적 코드 삽입 관리자는 추출된 물리적인 메모리 위치를 활용하여 하이퍼콜 명령어를 삽입하는 일을 수행한다. 결국 분석하고자 하는 실시간 운영체제가 하이퍼콜 명령어를 만나면, 하이퍼바이저의 트랩 핸들러가 수행한 코드 위치를 저장한다.

2.3. 정보 추출 및 로그 버퍼에 저장

마지막으로 저장된 정보를 추출하여 범용 운영체제의 분석도구 버퍼에 저장하는 부분이 있다. 게스트 운영체제가 점프코드를 만나면, 게스트 정보 추출자(Guest Information Extractor) 모듈을 호출한다. 이때 추출되는 정보는 게스트의 TCB(Task Control Block) 정보이다. 이 TCB 정보는 게스트 메모리에 접근하여 추출된다. 추출되는 정보는 다음과 같다. 현재 어떠한 태스크가 수행중인지에 대한 정보와 해당 코어의 스택에 쌓여 있는 정보 그리고 현재 태스크 상태이다. 위 정보가 수집되면, 게스트 정보 추출자는 수집된 정보를 범용 운영체제의 공유메모리에

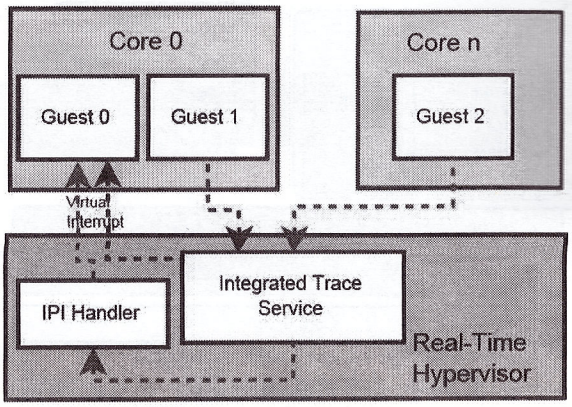


그림 7. 다중 코어에서 가상 인터럽트 전달
Fig. 7 Interrupt injection between multi-cores

저장한다. 그리고 가상 인터럽트(Virtual Interrupt)를 통해 범용 운영체제에게 위 정보가 도착했음을 알려준다.

실제 구현은 그림 7과 같다. 같은 코어에서 동작 중인 게스트일 경우에는 파란색 점선과 같이 코어간 통신이 없이 전달되도록 구현하였다. 만약 다른 코어에 동작 중인 게스트일 경우에는 코어간 IPI(Inter Processor Interrupt)를 이용하여 가상 인터럽트를 게스트에게 전달하도록 구현하였다.

이처럼 수집된 태스크 정보와 코드 수행위치에 대한 정보는 트레이스 인터럽트 서비스 루틴(Trace ISR)과 로그 변환기(Log Converter)를 통해 전달된다. 다음으로 실시간 운영체제의 태스크 코드 정보, 태스크 ID등을 마치 리눅스의 태스크인 것처럼 수정한다. 마지막으로 수정된 정보를 커널의 성능 분석도구가 가지고 있는 트레이스 버퍼에 저장하는 일을 수행한다. 이러한 일을 수행하면, 마치 실시간 운영체제에서 수행한 태스크 정보들이 마치 리눅스의 태스크 정보인 것처럼 성능 분석도구 버퍼에 저장된다. 결국 이러한 과정을 통해 기존 성능 분석에 사용되는 호스트 GUI의 수정 없이, 통합 성능분석이 가능하다.

3. 프로토타입 시스템에 적용

3.1. 프로토타입 환경

본 연구에서 제안하는 방법을 그림 8와 같은 프로토타입 환경에 적용 하였다. 하드웨어는 Cortext-A15 기반의 Arndale Board[5]를 사용하였고, 타겟 하이퍼바이저는 공개소스로 구현된 실시간 하이퍼바이저인 K-Hypervisor[6]를 사용하였다.

게스트 운영체제로는 안드로이드 4.4(kitkat) 버

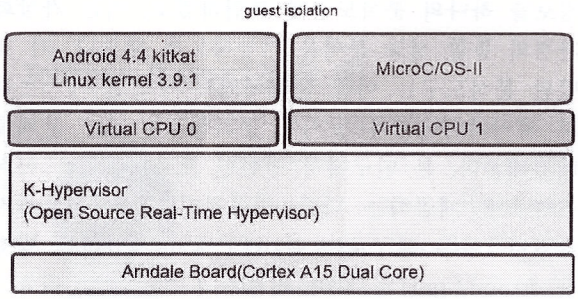


그림 8. 프로토타입 시스템 구성도
Fig. 8 Prototype system structure

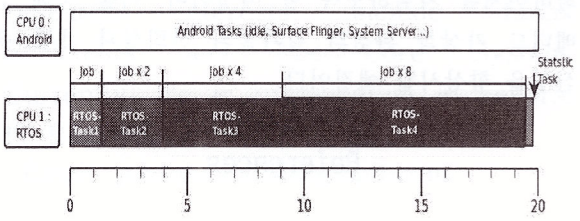


그림 9. CPU 코어별 태스크 워크로드
Fig. 9 Task workload for each cpu

전과 리눅스 커널 3.9.1을 사용하였고, 두 번째 게스트 운영체제에는 실시간 운영체제 중 하나인 MicroC/OS-II를 이용하였다.

본 연구의 프로토타입에 대한 실험을 위해, 게스트 안드로이드는 CPU 0번에 고정 할당하였다. 또한 실시간 운영체제는 CPU 1번에 고정 할당하여 동작시켰다. 안드로이드는 화면 대기상태로 두었고, 실시간 운영체제의 워크로드는 그림 9와 같이 구성하였다. 일정 기간 동안 일을 수행하는 RTOS-Task1과 점점 2배의 일을 더 수행하는 RTOS-Task2, RTOS-Task3, RTOS-Task4 태스크들로 구성하여, 통합 시스템 성능분석을 수행하였다.

4.2. 프로토타입에 적용

본 연구에서 사용한 오프라인 분석 툴은 안드로이드 시스템에서 전력과 성능을 같이 분석할 수 있는 자동화 분석툴[7]을 이용하였다. 통합 성능분석을 수행한 후 기존 리눅스 성능 분석 GUI를 수정 없이 이용하여, 오프라인 분석을 하였다.

그림 10은 통합 트레이스 결과를 보여준다. 먼저 툴의 왼쪽에는 CPU 명과 태스크 명에 대한 리스트를 보여주고, 오른쪽은 수행시간을 시각화하여 보여준다. 여기서 파란색 점선 박스로 되어 있는 부분은 범용 운영체제인 안드로이드의 성능분석 결과이

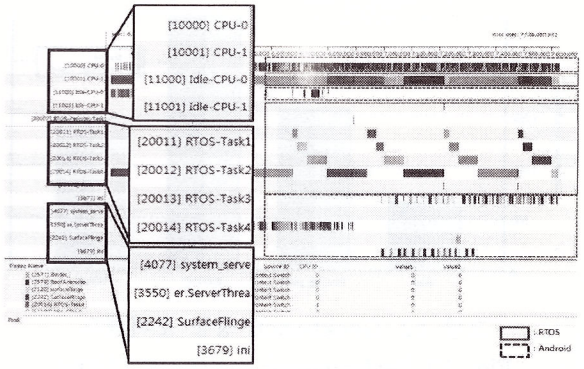


그림 10. 통합 트레이스 호스트 GUI 결과
Fig. 10 Result in the integrated trace

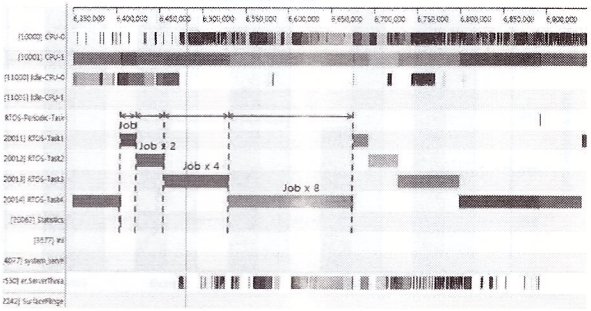


그림 11. 통합 트레이스 결과 확대 화면
Fig. 11 Zoom in the integrated trace result

고, 빨간색 실선 박스로 되어 있는 부분은 실시간 운영체제의 성능분석 결과이다.

해당 툴을 이용하여, 특정 시점을 확대하면, 그림 11과 같은 결과를 볼 수 있다. 앞에서 설명한 것과 같이, RTOS-Task1, RTOS-Task2, RTOS-Task3과 RTOS-Task4가 주기적으로 2배의 시간동안 일을 수행한다. 이는 앞에서 설명한 실시간 운영체제의 태스크 수행흐름과 동일한 결과를 보여준다. 동시에 안드로이드 태스크인 system_server, SurfaceFlinger 등 CPU0에서 수행하는 안드로이드 태스크도 볼 수 있다.

4.3. 성능 오버헤드

리눅스 게스트의 문맥교환을 대상으로 벤치마크를 사용하여 성능 오버헤드를 측정하였다. 비교 대상으로는 Native 리눅스와 본 논문에서 제안한 통합 트레이스(Integrated Trace) 기능을 추가한 리눅스이다. 동적 코드 삽입은 리눅스의 문맥교환에만 추가하여 그 성능을 분석하였다. 수행한 벤치마크로는 Geekbench[8]와 UnixBench[9]이다.

먼저 Geekbench 실험 결과는 그림 12와 같다.

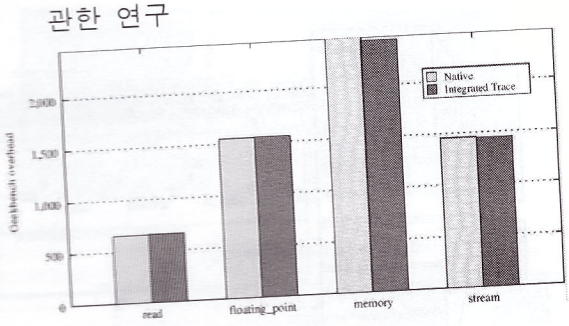


그림 12. Geekbench 성능 오버헤드
Fig. 12 Performance overhead of the Geekbench

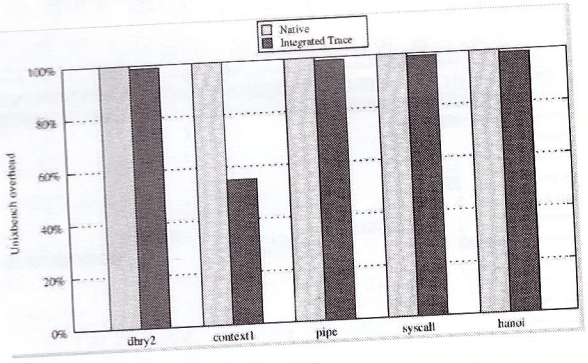


그림 13. UnixBench 성능 오버헤드
Fig. 13 Performance overhead of the UnixBench

실험 결과는 모든 벤치마크에서 약 1%미만의 오버헤드가 발생하였다.

Unixbench의 경우 그림 13과 같이 대부분 약 1% 미만의 오버헤드가 발생하였고, 문맥교환만 수행하는 context1 벤치마크일 경우 45%의 성능 저하가 발생하였다. 이는 다른 일을 수행하지 않고 계속 문맥교환만 호출하기 때문에, 점프코드에 의해 발생하는 오버헤드이다. 따라서 성능측정을 수행하는 함수가 증가 할수록 점프코드와 페이지 변환 등에 의해 성능 저하를 가져온다. 하지만 본 기법은 성능측정 시 11배의 오버헤드를 가지는 리눅스의 ftrace[10]와 같이 디버깅할 때만 활성화 되어 동작하며, 평소 동작중인 시스템의 성능에는 영향을 미치지 않는다.

IV. 결 론

본 연구는 최근 임베디드 이종 운영체제를 사용하는 실시간 가상화 환경을 대상으로, 통합적인 성능 분석 방법을 구현하였다. 이종의 운영체제 환경을 분석하기 위해, 기존 개발되어 온 범용 운영체제의 성능 분석도구를 이용하였다. 여러 운영체제의 정보를 하나의 분석도구에 통합하였고, 이는 가상화 환경의 통합 성능 분석을 가능하도록 해준다. 기 개발된 분석도구는 이종 운영체제로 구성된 게스트들과 하이퍼바이저가 동시에 분석할 수 없는 문제를 해결해준다. 본 연구에서 구현한 프로토타입은 아직 리눅스에 제공하는 다른 범용적인 툴을 사용하지 않았다. 향후 연구로는 본 연구에서 제안하는 방법을 ftrace, perf와 같이 리눅스의 범용적인 리눅스 툴을 지원하도록 할 예정이다. 또한 본 연구에서 활용한 하이퍼바이저는 아직 실시간성을 만족하지 못한 문제점이 있다. 향후 실시간성을 지원하는 하이퍼바이저를 지원하도록 할 예정이며, 이를 통해 임베디드 가상화 환경의 결과물의 신뢰성과 개발 생산성을 향상시킬 예정이다.

References

[1] G. Heiser, "Virtualizing embedded systems: why bother?," Proceedings of The 48th ACM/EDAC/IEEE Design Automation Conference, pp. 901-905, 2011.

[2] Y. Yunomae, "Integrated trace using virtio-trace for a virtualization environment," Proceedings of LinuxCon, 2013.

[3] M. Gebai, M.R. Dagenais, "Virtual machines CPU monitoring with kernel tracing," Proceedings of IEEE 27th Canadian Conference on Electrical and Computer Engineering, pp. 1-6, 2014.

[4] Z. Shao, L. He, Z. Lu, H. Jin, "Vsa: An offline scheduling analyzer for xen virtual machine monitor," Future Generation Computer Systems, Vol. 29, No. 8, pp. 2067-2076, 2013.

[5] http://arndaleboard.org

[6] https://github.com/kesl/khypervisor

[7] H.J. Kim, J.H. Kyong, S.S. Lim, "A systematic power and performance analysis framework for heterogeneous multiprocessor system," IEMEK J. Embed. Sys. Appl., Vol. 9, No. 6, pp. 315-321, 2014 (in Korean).

[8] http://www.primatelabs.com/geekbench

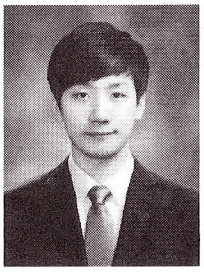
[9] http://code.google.com/p/byte-unixbench

[10] T. Bird, ""Measuring function duration with ftrace,"" Proceedings of The Japan Linux Symposium, 2009.

Joohyun Kyong (경 주 현)

He is currently a Ph.D. candidate in Computer Science at Kookmin University, Seoul, Korea. He received his M.S. degree in Computer Engineering from Hansung University, in 2009. His current research interests include Linux Scalability and Many-Core System. Currently, He is participating in several IT R&D programs in Korea.
Email: joohyun0115@gmail.com

In-Kyu Han (한 인 규)



He is currently a Ph.D. candidate in Computer Science at Kookmin University, Seoul, Korea. He received his M.S. degree in Computer Science from Kookmin University, in 2015. His current research interests include Internet of Things(IoT) and Virtualization System. Currently, He is participating in several IT R&D programs in Korea.
Email: in1004kyu@gmail.com

Sung-Soo Lim (임 성 수)

He received Ph.D. degrees in Electrical and Computer Engineering from Seoul National University, Seoul, Korea, in 2002. He is professor in the school of Computer Science in Kookmin University, Korea. His research interests include Virtualization, Real-Time Systems and Mobile Power Management.
Email: sslim@kookmin.ac.kr