제33회 한국정보처리학회 춘계학술발표대회 논문집 제17권 제1호 (2010. 4)

연성 실시간 Xen 하이퍼바이저 설계

허경우, 김진, 고영웅
한림대학교 컴퓨터공학과
e-mail: rapperkw, jinkim, yuko@hallym.ac.kr

Design of Soft Real-time Xen Hypervisor

Kyung Woo Hur, Jin Kim, Young Woong Ko
Dept of Computer Engineering, Hallym University

요 약

본 논문에서는 Xen 가상 머신에 연성 실시간 기능을 제공하기 위한 방안을 제시한다. 현재 실시간 하이퍼바이저(hypervisor)는 대부분 경성 실시간(hard real-time) 보장을 위하여 자원을 파티션(partition)하고 고정된 태스크에 대한 수행을 지원한다. 본 연구에서는 오픈 환경을 위한 조합적 실시간 프레임워크에 기반한 계층형 실시간 스케줄링 시스템을 제안하고 있다. 제안하는 시스템은 오픈 환경에서 동적으로 태스크 및 컴포넌트의 유입이 가능하게 설계되었다. 제안하는 시스템에 대한 사전 연구 결과로써 Xen 플랫폼의 오버헤드 분석 결과를 제시하고 있다.

1. 서론

컴퓨터 기술과 하드웨어 가격의 하락으로 복잡하고 고성능의 하드웨어 자원을 이용한 시스템이 대중화 되고 있다. 최근 널리 사용되고 있는 시스템의 경우 대부분 멀티코어 기술이 적용된 프로세서를 사용하고 있으며, 크기가 작고 하드웨어 사양이 높아져 복잡한 연산 처리를 하는 소프트웨어를 지원하고 있다. 이와 같은 하드웨어의 급속한 변화는 가상화(Virtualization) 기술[1]의 기반이 되고 있다. 가상화는 고성능의 단일 하드웨어 플랫폼에 다수의 시스템이 동시에 운용이 될 수 있는 기반을 제공하는 기술이다.

본 연구에서는 가상화 오픈 프로젝트인 Xen[2] 에서 게스트 운영체제들이 실시간 처리를 지원할 수 있도록 하기 위한 플랫폼을 제안하고 있다. 기존의 실시간 가상화에 대한 연구는 대부분 경성 실시간 시스템을 대상으로 하고 있기 때문에 하드웨어 자원을 고정적으로 파티셔닝(partitioning)하는 방법을 도입한다. 또한 실시간 커널과 비실시간 커널이 각각의 고정된 하드웨어 자원에 할당되어 수행됨으로 새로운 태스크의 유입이나 게스트 운영체제의 마이그레이션 등을 지원하기 어려운 문제점이 존재한다.

예를 들어, 현재 널리 사용되고 있는 실시간적인 처리를 위한 가상화 기술로는 KUKA RTOSWin[3], RTS Hypervisor, Trango, acontis technologies의 AT-RTOSVisor, Vmware MVP(Mobile Virtualization Platform) 등이 대표적이다. KUKA RTOSWin은 GUI를 처리하는 운영체제로 Windows XP/Vista를 지원하며 실시간 적인 처리를 위해서는 Windows CE, VxWorks, On Time RTOS-32, RT/Linux 그리고 QNX를 지원한다. 각 운영체제는 VmfWin(Virtual Machine Framework for Windows)라는 가상 머신 위에서 수행이 되며, 실시간 운영체제와 Windows 운영체제 간의 KUKA VMF Binary Module을 통해서 처리가 된다. 즉, 실시간 운영체제는 주어진 태스크 집합에 대해서 운영을 하고, GUI 처리를 해야 하는 데이터는 VMF를 통해서 Windows 운영체제로 보내주는 방법을 사용하는 것이다. 본 접근 방식은 가상 머신에서 동작되는 실시간 운영체제들이 고정적으로 분할된 프로세서 자원을 사용하기 때문에 시스템 세팅이 동적으로 바뀔 수 있는 상황에서 적용하기 힘든 단점이 있다. 즉, 운영체제마다 코어가 매핑되어 수행되고 인터럽트 처리와 같은 부분만 우선순위가 높은 실시간 운영체제로 전달되어 분석/처리가 되며, 운영체제간의 데이터 교환은 VMF에서 제공하는 통신 채널을 이용하거나 공유 메모리를 사용해서 데이터를 전달하기도 한다. 따라서 이와 같은 방식은 실제적인 산업현장에서 사용하는 고정된 실시간 태스크를 가지고 있는 시스템에서는 큰 무리 없이 사용할 수 있으나 범용적인 모델로 사용하기에는 무리가 있다. 열거된 다른 실시간 처리를 위한 가상화 기술도 이와 유사한 방법으로 수행된다.

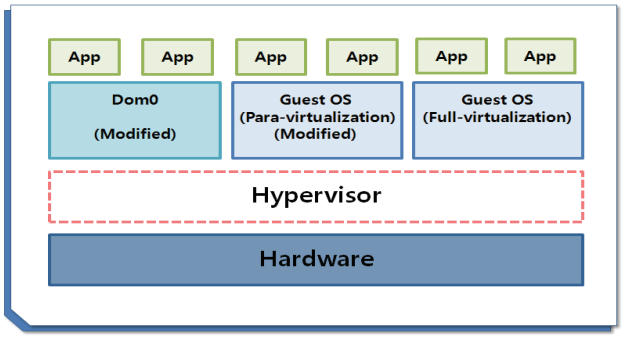
본 연구에서는 기존 Xen 커널에 계층적 스케줄링 개념을 적용한 연성 실시간 가상 머신 플랫폼을 제안한다. 오픈 환경에서 동적으로 컴포넌트가 유입이 되고, 태스크가 동적으로 추가/삭제 될 수 있는 방식을 지원한다. 제안하는 방식은 이론적인 배경으로 주기적 자원 모델을 기반으

1) 본 연구는 중소기업청에서 지원하는 2009년도 산학연 공동기술개발사업 (No. 00039553)과 한국 연구재단(KRF)의 지원(No.2009-0076520)을 받은 결과물임을 밝힙니다.

로 하는 컴포넌트 인터페이스를 사용하는 것이 특징이며, 런타임에 컴포넌트 인터페이스가 동적으로 조절이 될 수 있도록 한다.

2. 시스템 설계 및 오버헤드 측정

가상화는 운영체제의 수정 유무에 따라 전가상화(Full Virtualization)와 반가상화(Para-Virtualization)로 나눌 수 있다. 반가상화는 하이퍼바이저 위에서 수행되는 게스트 운영체제의 소스코드를 수정해서 수행해야 하고, 블록 디바이스나 네트워크의 입출력을 Dom0 이라는 특별한 게스트 운영체제로 리다이렉션(redirection)하여 처리한다는 부분에서 마이크로 커널 개념과 비슷해 보인다. 하지만 전가상화는 기존의 플랫폼 수정 없이 하이퍼바이저에 의해서 에뮬레이션(emulation) 되어 처리가 되기 때문에 매우 편리하게 사용 할 수 있다. 그리고 에뮬레이션에 의한 성능 감소를 극복하기 위해서 하드웨어적인 지원이 필수이며, Intel VT 또는 AMD-V 와 같은 프로세서 수준의 가상화 기술 지원으로 급격한 성능 향상이 이루어지고 있다. 본 연구에서 사용하는 Xen은 전가상화와 반가상화를 모두 지원하며 Xen 아키텍처는 다음 그림1 과 같이 구성되어 있다.

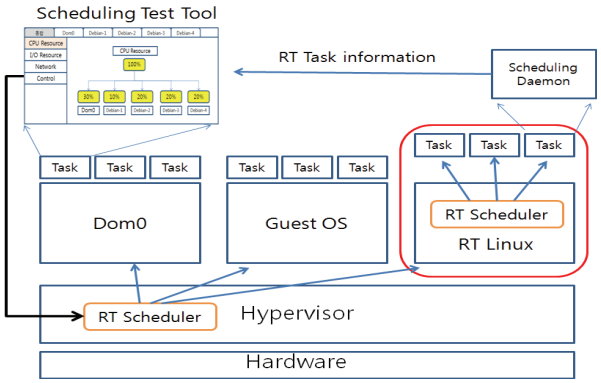


(그림 1) Xen Architecture

시스템이 부팅되는 순간 부트 로더가 Xen 하이퍼바이저를 호출하고 하이퍼바이저는 부팅에 관련된 장치 및 스케줄러 초기화와 Dom0을 모듈 형태로 생성한다. 이 때 게스트 운영체제는 Xen의 하이퍼바이저 위에서 동작하며 하이퍼바이저에 포함되어있는 Credit[4] 스케줄러에 의해서 스케줄링 된다. 스케줄러는 게스트 운영체제를 동작시키기 위한 VCPU(Virtual CPU)를 스케줄링 하며 VCPU에 30ms 마다 실행을 보장한다. 이는 Round Robin 스케줄러와 유사하게 공평성을 주었으며 급하게 처리해야 하는 작업을 위해 BOOST 라는 우선순위를 주어 응답성을 향상시켰다.

2.1 실시간 Xen 하이퍼바이저 설계

본 연구에서는 그림2 에서 보이는 계층적 구조의 실시간 가상 머신을 제안한다. 제안하는 시스템은 계층적 스케줄링 개념에 기반을 하고 있는 조합적 실시간 스케줄링을 (Compositional Real-time Scheduling) 이론을 Xen 커널과 게스트 운영체제 적용하고 있다.



(그림 2) 게스트 운영체제와 Dom0 그리고 하이퍼바이저 간의 스케줄링 정보 전달 과정

조합적 실시간 스케줄링 이론에서는 각 컴포넌트가 제공하는 인터페이스에 따라서 일정한 주기와 수행시간이 보장이 되면, 내부의 컴포넌트가 주어진 스케줄러를 사용해서 데드라인을 놓치지 않고 수행이 될 수 있다는 이론을 제시하고 있다[5]. 따라서 게스트 운영체제를 하나의 컴포넌트로 가정하고, 내부에 있는 실시간 태스크 집합과 스케줄러에 대해서 필요한 프로세서 자원을 할당할 수 있으면, 해당 게스트 운영체제는 제한 시간을 놓치지 않고 실시간 태스크를 스케줄링 할 수 있다. 제안하는 시스템에서는 2단계의 계층으로 구성되어 있으며 상위 컴포넌트는 하이퍼바이저에 해당되며, 하위 컴포넌트는 게스트 운영체제에 해당된다. 상위 컴포넌트는 상위 스케줄러에 의해서 게스트 운영체제들을 스케줄링하게 되며, 이때, 게스트 운영체제에게 할당된 주기와 수행시간을 기준으로 RM 또는 EDF 방식으로 스케줄링을 하게 된다. 각 게스트 운영체제는 자신의 차례가 되었을 때, 처리해야할 실시간 작업을 주어진 하위 스케줄러로 처리하게 된다.

태스크가 실시간으로 수행되기 위해서는 상위 스케줄러로 동작되는 하이퍼바이저 뿐만 아니라, 하위 스케줄러가 수행되는 게스트 운영체제도 조합적 실시간 스케줄링 프레임워크를 지원해야 한다. 이를 위해서 본 연구에서는 게스트 운영체제로 사용할 리눅스 커널에 실시간 기능을 강화하는 작업을 수행한다. 리눅스 커널은 모노리틱 커널 구조로 되어 있고, 커널에서 장시간의 지연이 발생할 수 있다. 따라서 우선 순위가 높은 실시간 태스크의 수행을 보장하기 위해서는 커널에서 지연되는 시간을 최소로 하여야 한다. 현재 리눅스 커널에서 preemption 포인트를 곳곳에 추가하여 커널에서 발생되는 지연을 줄이는 연구가 많이 진전되어 있는 상황이다. 이중에서 본 연구는 Linux PREEMPT-RT 패치를 적용하여 커널의 중간 중간에서 preemption이 발생이 될 수 있도록 한다. 또한 현재 리눅스 커널은 실시간 스케줄링 개념이 제대로 지원되고 있지

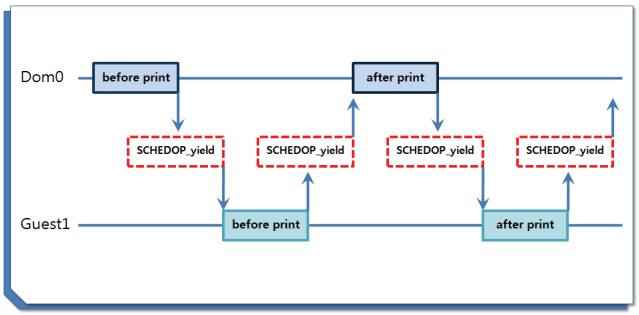
않다. 실시간 스케줄러의 범주로 SCHED_RR, SCHED_FIFO가 지원되기는 하지만 주기적인 태스크 지원을 위한 모델이 아니라, 태스크의 우선순위를 급격히 올려줌으로 특정한 태스크가 우선적으로 실행될 수 있게 해주는 고정 우선 순위 기반 실시간 스케줄러에 해당된다. 본 연구에서는 조합적 실시간 스케줄러로 사용될 수 있는 RM과 EDF 스케줄러를 리눅스 커널에 추가하는 작업을 수행한다. 추가된 스케줄러 기능을 이용할 수 있도록 하기 위해서 실시간 스케줄러를 호출할 수 있는 라이브러리를 작성한다.

마지막으로 조합적 실시간 스케줄러에 의해서 컴포넌트가 성공적으로 스케줄링되기 위해서는 스케줄링 테스트가 수행되어야 한다. 이를 위해서 Dom0에서 자원 모니터링 및 스케줄링을 테스트해주는 태스크가 수행된다. 동적으로 스케줄링 가능성 테스트를 수행하기 위해서 해당 모듈은 하이퍼바이저에 존재하거나 또는 Dom0에 존재해서 수행하고 최종 결과만을 하이퍼바이저에게 전달하는 두 가지 방식으로 구현될 수 있다. 하이퍼바이저에 스케줄링 가능성 테스트 모듈을 적재 하는 경우는 모든 게스트 운영체제가 하이퍼바이저에게 스케줄링 정보를 전달해야 하기 때문에 하이퍼 콜을 수행할 수 있는 반가상화 게스트 운영체제들만 가능하다. 또한 스케줄링 정보를 보관하는데 필요한 메모리 자원과 계산 시간을 필요로 하기 때문에 하이퍼바이저의 크기가 커지게 되는 단점을 가진다. 그리고 스케줄링 테스트를 하기 위해서 게스트 운영체제의 자원 사용량을 지속적으로 모니터링하고 그 데이터를 유지하는 작업도 하이퍼바이저가 해야 한다는 문제가 있다. Dom0에 스케줄링 테스터를 두고, 스케줄링 가능성 체크에 대한 결과만 하이퍼바이저에 전달하는 방식은 Dom0에서 GUI 방식으로 사용자와 인터액션을 할 수 있기 때문에 사용자의 선호도를 반영하여 주기 및 수행 시간 정보를 변경하기 용이하다. 또한 자원 모니터링 및 스케줄링 정보를 사용자 태스크에서 관리하기 때문에 메모리와 계산 시간에 구애받지 않고 수행할 수 있는 장점이 있다. 본 연구에서는 스케줄링 테스터 툴과 자원 모니터링 툴을 Dom0에서 GUI 프로그램으로 구현하여 관리하는 방법으로 접근한다.

2.2 Xen 오버헤드 측정

본 연구에서는 Xen 에 실시간 스케줄링을 적용하기 위해 현재 Credit 스케줄러가 동작되고 있는 Xen 시스템의 성능을 측정하였다. 우선 가상화 형태에 따른 게스트 운영체제의 성능을 측정하기 위하여 시스템에 Xen을 설치하고 게스트 운영체제를 전가상화와 반가상화의 형태로 설치하였다. 그리고 각 게스트 운영체제에 lmbench[6] 벤치마크 툴을 설치하여 각 운영체제 별 성능을 측정하였다. 또한 실시간 스케줄링에서 고려해야 하는 중요한 변수인 각 게스트 운영체제가 스케줄링 될 때의 오버헤드를 측정하였다. 본 연구에서는 정확한 오버헤드 측정을 위해 CPU의 TSC(Time Stamp Counter) 값을 이용하였다. CPU에는 전원이 인가되면서 부터 CPU의 클럭 틱(tick)을 기록하는 내부 카운터가 있는데 이것을 TSC라고 한다. 스케줄링 되기 전의 TSC 값과 스케줄링 되고 나서의 스케줄링 값을 비교 하면 걸린 시간 및 오버헤드를 측정할 수 있다.

또한 각 게스트 운영체제에서 주어진 시간을 다 소모하지 않고 강제적으로 다른 운영체제로 스케줄링 되게 하기 위하여 하이퍼 콜(Hyper Call)을 이용하였다. 하이퍼 콜은 운영체제의 시스템 콜과 유사한 형태로 게스트 운영체제에서 Xen의 하이퍼바이저로 요청을 보낼 수 있다. 기본적으로 Xen에는 약 50개 정도의 하이퍼 콜이 정의 되어 있으며(include/public/xen.h) 스케줄링에 관련된 하이퍼 콜인 __HYPERVISOR_sched_op에 SCHEDOP_yield 매개변수를 이용하여 게스트 운영체제간의 스위칭을 하였다. 실험 프로그램은 무한 루프로 수행되면서 한번의 TSC 값을 출력을 하고 하이퍼 콜을 호출하여 다른 VCPU에게 제어를 넘기게 된다. 이 때, CPU를 받은 게스트 운영체제에서는 다시 TSC 값을 출력하고 또 다시 하이퍼 콜을 호출하여 CPU 제어를 양보한다. 이것을 반복적으로 수행하여 제어를 양보하기 전의 TSC 값과 제어를 넘겨 받은 운영체제의 TSC 값을 비교하여 오버헤드를 측정하였다. 아래의 그림3 은 본 연구에서 사용한 프로그램의 개념도를 나타내고 있다.



(그림 3) 실험 프로그램 개념도

3. 실험 결과 및 분석

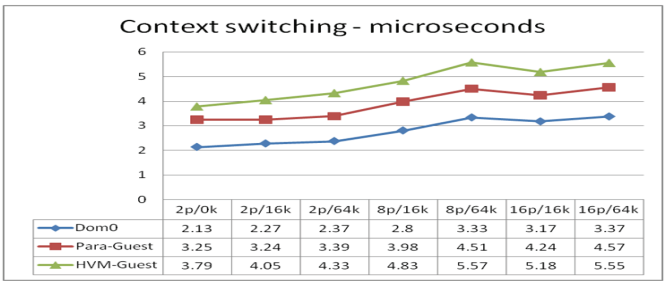
본 연구에서 사용한 실험 환경은 아래의 표와 같다.

<표 1 > 실험 환경

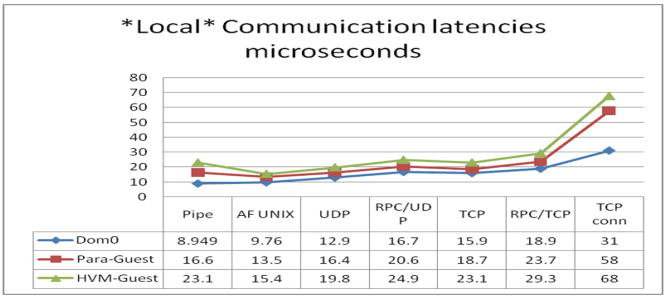
Hardware	
CPU	Intel Core i7 920(2.66GHz)
RAM	4 GB
HDD	Seagate 500GB ST3500418AS
Software	
OS	CentOS 5.4
Xen Hypervisor	3.4.2
Guest OS	CentOS 5.4(Para, Full)

정확한 실험 결과를 측정하기 위하여 lmbench를 이용하여

벤치마크 할 때는 벤치마크 외에 모두 idle 상태로 측정하였으며, 각각 똑같은 VCPU 수와 RAM을 할당 하였다. 가상화 형태에 따라 lmbench를 이용하여 성능을 측정한 결과는 다음 그림과 같다.



(그림 4) 가상화 형태에 따른 문맥전환 시간 그래프



(그림 5) 프로세스 간의 통신 방법에 따른 지연 시간

위에 그래프에서 보듯이 Dom0 이 다른 게스트 운영체제에 비하여 성능이 좋게 나오는데 이는 각 게스트 운영체제의 입출력을 담당하기 위해 스케줄러가 Dom0 에 좀 더 많은 CPU를 할당하기 때문임을 볼 수 있다. 여기서 주목해야 할 것은 반가상화 형태의 게스트 운영체제가 전가상화 형태의 게스트 운영체제보다 좀 더 나은 성능을 보였다는 것이다. 반가상화는 운영체제의 소스 코드를 수정하여 가상화에 맞게 적용되어 수행되기 때문에 하이퍼바이저 위에서 에뮬레이션 되어 수행되는 전가상화 형태보다 성능이 좋게 나왔음을 볼 수 있다. 이런 차이를 극복하기 위해 CPU의 가상화 기술이 계속적으로 발전하고 있기 때문에 시간이 지나면 반가상화와 전가상화 형태의 운영체제 성능의 차이는 줄어 들 것으로 예상된다.

다음으로 각 게스트 운영체제가 스위칭 될 때의 TSC 값을 표2 로 나타내었다. Dom0에서 게스트 운영체제로의 문맥 전환에 걸리는 TSC 값은 평균 84472 정도이며 게스트 운영체제에서 Dom0 으로의 문맥 전환은 평균 124451 이다. 이를 시간으로 환산 하는 식은 다음과 같다.

$$T_{sec} = TSC / CPU\ Clock$$

TSC 값은 1초에 CPU 클럭수 만큼 증가하기 때문에 평균 값에서 CPU 클럭수를 나누면 시간으로 환산 할 수 있다. 실험 환경에서의 CPU 클럭수는 2.66 GHz 이므로 실험에서 나온 값을 초로 환산하면 각각 약 31μs, 46μs 로 나타낼 수 있다.

(표 2) 문맥전환 TSC 값 및 차이 값

Name	TSC	gap
Dom0	164395756624373	84210
Guest1	164395756708583	
Guest1	164395756933260	125955
Dom0	164395757059215	
Dom0	164395757131583	81577
Guest1	164395757213160	
Guest1	164395757321025	118868
Dom0	164395757439893	
Dom0	164395757509373	81457
Guest1	164395757590830	
Guest1	164395757695958	124290
Dom0	164395757820248	

5. 결론 및 향후 연구

하드웨어의 고성능화와 친환경적이며 효율적인 시스템 사용을 위해 가상화 기술이 널리 적용 되고 있다. 그 중 실시간 처리를 요구하는 임베디드 분야와 다른 많은 분야에서도 가상화의 활용도가 높을 것으로 예상된다. 이에 본 연구에서는 가상화 프로젝트인 Xen에서 실시간 스케줄링을 적용하기 위한 사전 준비로 가상화 형태에 따른 성능 및 각 게스트 운영체제가 문맥전환 될 때의 오버헤드를 실험하고 분석하였다. 본 실험결과에 나온 분석 값을 이용하여 가상화 형태에 따른 실시간 처리나 실시간 스케줄링을 적용할 때 고려해야 할 오버헤드를 예측 할 수 있으며, 실시간 스케줄링을 제공하기 위한 기초 자료로 활용 할 수 있을 것으로 기대 된다.

향후 연구로는 Xen에서 기본으로 제공되고 있는 Credit 스케줄러를 변경하여 가상화 기반의 프로젝트에서 좀 더 유연한 실시간 처리를 제공하기 위한 연구를 할 것이다.

참고문헌

[1] Gil Neiger, Amy Santoni, Felix Leung, Dion Rodgers, Rich Uhlig. "Intel® Virtualization Technology: Hardware support for efficient processor virtualization". Intel® Technology Journal. 2006

[2] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, Andrew arfield. "Xen and the art of virtualization". Proceedings of the nineteenth ACM symposium on Operating systems principles, 2003.

[3] http://virtualization.kuka-rtos.com/en/rtos/

[4] http://wiki.xensource.com/xenwiki/CreditScheduler

[5] Insik Shin, Insup Lee, "Compositional Real-Time Scheduling Framework with Periodic Model", ACM Transactions on Embedded Computing Systems (TECS), Vol. 7, No. 2, pp. 30:1-30:39, April, 2008.

[6] http://www.bitmover.com/lmbench/