



## 저작자표시-비영리-동일조건변경허락 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.
- 이차적 저작물을 작성할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



동일조건변경허락. 귀하가 이 저작물을 개작, 변형 또는 가공했을 경우에는, 이 저작물과 동일한 이용허락조건하에서만 배포할 수 있습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

碩 士 學 位 論 文

실시간 게스트 운영체제 상의  
EtherCAT을 지원하기 위한 Xen  
하이퍼바이저 스케줄러 분석

高麗大學校 融合소프트웨어專門大學院

임베디드소프트웨어學科

文 兌 浩

2015 年 6 月

柳 懃 教 授 指 導  
碩 士 學 位 論 文

실시간 게스트 운영체제 상의  
EtherCAT을 지원하기 위한 Xen  
하이퍼바이저 스케줄링 분석

이 論文을 工學 碩士學位 論文으로 提出함

2015 年 6 月

高麗大學校 融合소프트웨어專門大學院  
임베디드소프트웨어學科

文 兌 浩 (印)



文兌浩의 工學 碩士學位 論文  
審査를 完了함

2015 年 6 月

委員長

유

혁

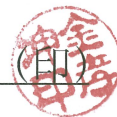
(印) 

委 員

김

효

곤

(印) 

委 員

백

상

헌

(印) 



## 요 약

가상화 기술의 필요성이 서버와 클라우드부터 개인용 데스크톱 PC 그리고 스마트폰, 스마트 자동차와 같은 임베디드 시스템까지 넓혀지고 있다. 그리고 이제 산업용 로봇 제어 시스템에도 가상화 요구가 증가하고 있다. 전통적인 가상화 환경과 다르게 임베디드 혹은 산업용 로봇 제어 시스템 환경은 가상화를 할 때 충분한 하드웨어 자원을 사용 할 수 있는 환경이다. 하지만 지금까지 진행 된 가상화 연구들은 게스트 운영체제 간에 하드웨어 자원을 사용하기 위해 경쟁을 해야 하는 상황을 고려하고 스케줄러가 구현되었다. 때문에 스케줄러에서 불필요한 동작을 하게 되어 오버헤드가 발생한다.

이 논문에서는 기존의 Xen 하이퍼바이저에서 지원하는 스케줄러들의 특성을 코드영역까지 분석하여 오버헤드의 원인을 찾아 보았다. 그리고 가상화 하지 않은 Native 실시간 리눅스에서 측정한 지연시간과 가상화 한 실시간 리눅스에서 측정 한 지연시간을 비교하여 오버헤드가 얼마나 되는지 확인하였다. 오버헤드를 줄이기 위해 Xen 하이퍼바이저 가상화 오버헤드중 가장 큰 원인인 CPU가상화 오버헤드를 감소시키기 위해 원인 중 하나인 로드밸런싱 부분을 제거하였다. 수정하지 않은 CREDIT1 스케줄러와 수정한 CREDIT1 스케줄러에서 측정한 지연시간을 비교했을 때 최소 지연시간은 1us, 평균 지연시간은 4us 그리고 최대 지연시간은 14us 만큼 감소한 것을 확인하였다.



## 목 차

1. 서론 .....	1
1.1 연구배경 .....	1
1.2 논문의 구성 .....	4
2. 배경지식 .....	5
2.1 EtherCAT .....	5
2.2 Xen 하이퍼바이저 .....	7
2.3 Intel VT-x .....	9
3. 관련연구 .....	11
3.1 RT-Xen .....	11
3.2 산업용 자동화 시스템 가상화 .....	12
4. Xen 스케줄러 .....	13
4.1 CREDIT1 .....	13
4.2 CREDIT2 .....	15



4.3 RTDS .....	17
4.4 SEDF .....	19
4.5 스케줄러 오버헤드 분석 .....	20
5. Xen 스케줄러 지연시간 측정 및 개선 .....	21
5.1 실험 환경 .....	21
5.2 실험 요약 .....	22
5.3 Xen 하이퍼바이저 스케줄러별 지연시간 측정 .....	24
5.4 로드밸런싱 제거 후 지연시간 측정 .....	26
6. 결론 .....	27
참고 문헌 .....	28



## 그림 목차

그림 1. 가상화를 이용한 산업용 로봇 제어 시스템의 소프트웨어 구조 .....	2
그림 2. Type-1 가상머신 모니터 .....	7
그림 3. Intel VT-x Ring 모드 변환 .....	9
그림 4. RTDS 스케줄러의 VCPU 관리 자료구조 .....	18
그림 5. Xen 하이퍼바이저와 도메인 CPU할당 .....	24
그림 6. Native 리눅스와 Xen 스케줄러별 게스트 리눅스의 지연시간 비교 ·	26
그림 7. CREDIT1에서 로드밸런싱 제거 후 지연시간 .....	27





## 표 목차

표 1. 스케줄러 변경에 따른 지연시간 측정 .....	25
표 2 로드밸런싱 제거한 CREDIT1 스케줄러의 지연시간 측정 .....	27



# 1. 서론

## 1.1 연구배경

가상화는 하나의 물리적인 컴퓨팅 자원을 추상화 하여 여러 개의 가상머신 (VM, Virtual Machine)만드는 기술이다.[1] 가상머신은 운영체제와 응용프로그램으로 이루어진 소프트웨어들을 실행 할 수 있다. 그 결과, 하나의 물리적인 하드웨어에서 2개 이상의 운영체제와 이 운영체제를 기반으로 한 응용프로그램이 실행 할 수 있다. 이러한 가상화 기술은 과거 서버 시스템과 같이 하드웨어 성능이 뛰어난 시스템에서 하드웨어의 사용률(utilization)을 극대화하기 위해 사용되어 왔으며, 최근에는 개인용 데스크톱, 임베디드 시스템, 모바일 단말까지 적용 범위가 광범위하게 확대되어 왔다.

공장 자동화를 위한 산업용 로봇의 제어 시스템에서도 가상화에 대한 요구 사항이 점차 증가하고 있다. 산업용 로봇은 공장 생산 라인에서 사용되는 로봇으로 크기가 1m이내의 소형 로봇부터, 3m의 대형 로봇까지 용도에 따라 다양한 크기의 로봇이 사용되고 있다. 과거에는 이들을 제어하기 위한 특화된 운영체제와 응용프로그램이 사용되어 왔다. 그 이유는 산업용 로봇은 기존 컴퓨터 시스템과 달리 고정 실시간 (Hard real-time)을 반드시 충족해야만 했기 때문에 로봇 제조사에서 배포한 실시간 소프트웨어로 제어 시스템을 구축했다. 하지만 공개 소프트웨어군의 실시간 지원을 위한 다양한 연구가 이루어져, 최근 산업용 로봇 제어 시스템은



실시간 리눅스를 기반으로 한 제어시스템이 구축되고 사용되고 있다.

뿐만 아니라 제어용 하드웨어 역시, Intel i7과 같이 고성능 프로세서가 도입되어 하드웨어 성능 제약을 해결하고, 더 나아가 다양한 소프트웨어 기능을 탑재하여 운용하고 있다. 가상화는 이렇게 변화하는 산업용 로봇 시스템 요구사항을 수용하기 위한 해결책으로 떠오르고 있다. 특히, 기존 로봇 제어 시스템과 함께, 비전 인식, GUI 기반 관리 시스템과 같은 부가 기능을 하나의 하드웨어에서 운용하기 위해서는 경성 실시간 제어 시스템의 고립(isolation)과 요구 성능 보장을 동시에 충족해야만 한다. 가상화가 제공하는 가상머신을 통해 하나의 가상머신에는 기존의 제어 시스템을 운용하고 다른 가상머신에는 추가 기능을 운용함으로써, 필요한 기능을 쉽게 추가하고 동시에 실시간성과 안정성이라는 이질적인(heterogeneous) 요구 사항을 동시에 충족시킬 수 있다.

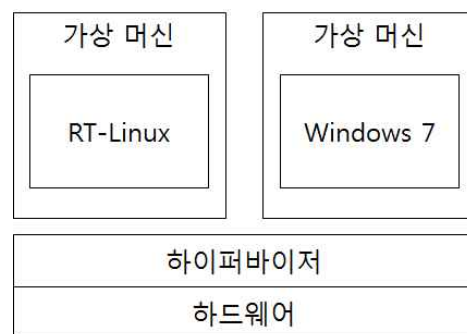


그림1. 가상화를 이용한 산업용 로봇 제어 시스템의 소프트웨어 구조

본 논문은 산업용 로봇 제어 시스템의 가상화를 위한 하이퍼바이저를 실시간성 지원 가능성을 연구한다. 과거 임베디드 시스템에서 하이퍼바이저의 실시간성을 지원하기 위한 연구가 있었으나, 이들 연구는 연성 실시간(Soft real-time) 지원,



비교적 긴 주기의 실시간 태스크의 스케줄링 지원을 목표로 하였다. 하지만 본 연구는 대상인 산업용 로봇은 1ms 주기의 실시간 태스크와 산업용 로봇을 제어하기 위한 이더넷 카드, Timing Peripheral과 통신을 해야 하는 매우 제한적인 실시간 시스템인 EtherCAT을 지원하기 위한 가상화 기술을 연구한다.[2] 이를 위해서는 가상화로 인한 성능 오버헤드를 최소화 해야만 한다. 본 연구는 이러한 오버헤드의 원인을 분석하고 해결 방안을 제시하고자 한다.



## 1.2 논문의 구성

본 논문의 구성은 다음과 같다. 2장에서는 산업용 로봇 EtherCAT과 가상화 기술인 Xen에 대해 설명하고자 한다. 3장에서는 Xen에서 실시간성을 지원하기 위한 연구와 산업용 로봇 가상화와 관련된 기존 연구들을 살펴본다. 4장에서는 Xen에서 제공하는 스케줄러를 분석하고 오버헤드 발생 원인을 설명한다. 그리고 5장에서는 실제로 측정한 지연시간 감소를 위한 Xen구현과 결과 확인을 위한 실험을 하였다. 마지막으로 6장에서는 구현하고 실험한 내용을 바탕으로 결과를 정리한다.



## 2. 배경지식

### 2.1 EtherCAT

EtherCAT은 경성 실시간 성능을 제공하기 위해 Beckhoff가 개발한 이더넷 프로토콜이다.[3] master와 slave 노드로 구성되어 있고, 기본적으로 표준 네트워크 인터페이스를 사용하도록 구현되어 있다. 하지만 실제로는 EtherCAT slave는 매우 짧은 지연시간 안에 패킷을 전달하기 용이하도록 FPGA 또는 ASIC 같은 특수한 하드웨어로 구현된다. 반면에 EtherCAT master는 오직 표준 컴포넌트를 사용해서 구현돼야 한다. 표준 컴포넌트를 사용함으로써 다양한 운영체제 위에서 master를 설치 할 수 있고, 거기에 slave를 연결하여 사용한다. 또한 EtherCAT은 slave와 연결하는 스위치 디바이스들 모두가 표준에 맞춰서 구성되어 있다면 어떤 종류의 표준 이더넷 트래픽이든 사용 가능하다. 만약 해당 이더넷 프레임의 크기가 너무 커서 유효 시간을 넘기거나 어떤 EtherCAT frame의 크기에 맞지 않을 경우에 수신 단에서 프레임을 재구성하여 보낼 수 있다. 이러한 기능은 cycle time내에 산업용 로봇 시스템을 제어하기 위해 중요한 기능이다.

EtherCAT 토폴로지를 구성할 때 최대 연결 가능한 노드의 수는 65,535개이며 노드들의 연결 구성은 master-slave와 master-master 그리고 slave-slave로 이뤄질 수 있다. 이러한 구성의 유연성 덕분에 중앙 집중 시스템 및 분산 시스템 구조 모두에 적합하다. 각 노드들은 통신을 위해 패킷을 주고받을 때 데이터를 모두 자신의 프레임에 기록한다. 이러한 동작 원리를 통해 스위치나 허브의 필요성을 없



애고, 사이클 당 한 프레임 정도로 대역폭 이용률을 높일 수 있다.

현재 전 세계적으로 광범위하게 임베디드 시스템부터 시작하여 장비제어, 측정 장비, 의료장비, 자동차 제조, 휴대용 장비 등에서도 사용되고 있다. 그리고 국제적으로 표준화된 개방형 기술로 누구나 자유롭게 사용 가능하며 어느 곳이나 호환 가능한 형태로 기술을 사용 할 수 있다.



## 2.2 Xen 하이퍼바이저

Xen 하이퍼바이저는 Type-1 혹은 베어메탈(bare-metal) 하이퍼바이저로 분류되는 가상머신 모니터이다.[4] 가상머신 모니터란 하나의 머신에서 여러 개의 운영체제가 동시에 실행될 수 있도록 하드웨어 자원을 가상화하여 게스트 운영체제에게 제공해주고, 이것을 관리한다. 이를 위해 각 가상머신들이 하드웨어를 사용하는 것을 스케줄링 한다. 가상머신 모니터의 종류에는 Type-1(Native or bare-metal) 과 Type-2 (hosted) 두 가지가 있다.



그림2. Type-1 가상머신 모니터

그림 1은 Xen이 속한 Type-1 가상머신 모니터를 도식한 것이다. Type-1 가상머신 모니터는 하드웨어 계층 위에 가상머신 모니터가 동작하고, 가상머신 모니터가 가상머신을 생성하여 여러 개의 게스트 운영체제가 작업을 실행할 수 있도록 해준다. 가상머신 모니터가 컴퓨터 시스템 전체를 가상화하는데 CPU가상화, 메모리 가상화, I/O 가상화가 필요하다.

CPU가상화를 위해서는 CPU가 가지고 있는 특권 모드와 비특권 모드에서의 동





작을 구별해야 한다. 기존의 비가상화 상태에서 운영체제가 동작 할 때에는 운영 체제 커널이 CPU의 특권모드에서 실행되고 사용자 응용프로그램은 비특권 모드에서 실행되었다. 하지만 가상화 환경에서는 가상머신 모니터가 특권모드에서 실행되고 운영체제의 커널이 비특권 모드에서 실행되기 때문에 운영체제의 커널에 하이퍼바이저를 부르는 트랩으로 하이퍼 콜을 추가해줘야 한다. 이렇게 커널에 수정하여 가상머신에 올리는 것을 반가상화 기법이라고 한다.

메모리 가상화는 비가상화 환경에서 메모리를 관리하기 위해 사용되던 물리 주소와 가상 주소를 가진 두 단계 주소 체계에 가상 물리 주소 단계를 추가하여 세 단계 주소 체계를 만든 것이다. 게스트 운영체제는 가상머신에서 동작하고 있다는 사실을 모르기 때문에 가상화되지 않은 환경에서 사용하던 두 단계 주소체계를 사용한다. 하지만 게스트 운영체제가 사용하는 물리주소는 실제 물리주소와 다르기 때문에 하이퍼바이저가 가상 물리주소를 제공해주고 이 가상 물리주소를 실제 물리주소로 변환해줘야 한다. Xen 하이퍼바이저의 경우 게스트 운영체제를 반가상화 기법으로 올릴 경우 직접 페이지 테이블을 수정하고 하이퍼바이저가 나중에 확인하도록 할 수 있다.

I/O 가상화는 키보드, 마우스, 네트워크, 디스크, 그래픽 카드와 같이 CPU가 어떤 작업을 실행할 때 주변장치 혹은 주변 머신 그리고 사용자와 통신하기 위해 사용되는 입출력 장치를 가상화 하는 것이다. Xen에서는 게스트 운영체제의 커널을 수정하는 반가상화 기법에서 분리 드라이버 모델을 사용하였다.



## 2.3 Intel VT-x

Intel VT-x란 x86 기반의 하드웨어에서 가상화 머신 모니터를 지원하기 위해 Intel에서 CPU에 가상화 기능을 추가한 것을 말한다.[5] 게스트 운영체제 커널의 소스 코드를 수정하거나 바이너리 변화 방법 없이도 트랩과 에뮬레이터 방식으로 하이퍼바이저를 구현할 수 있도록 하드웨어적인 기능을 추가한 것이다. 첫 번째로 문제가 되었던 특권모드와 비특권 모드를 분리해서 부여한 것이다. 가상머신 모니터는 VMX Root 모드에서 특권모드와 비특권 모드가 있고, 게스트 운영체제에게는 VMX Non-root 모드에서 특권모드와 비특권 모드를 만들어 주었다.

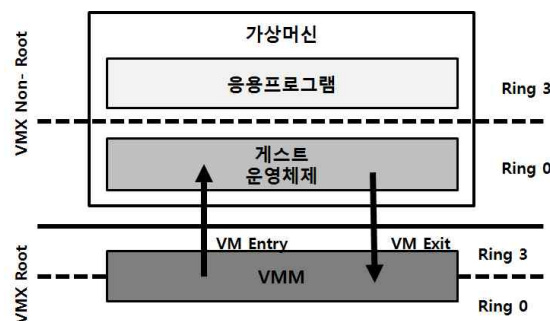


그림3. Intel VT-x Ring 모드 변환

그림 2를 보면 Intel VT-x에서 어떻게 특권모드를 변환하는지 나온다. intel의 x86\_64 CPU 아키텍처에서는 특권모드를 Ring 번호로 구분하는데 특권모드는 Ring0 비특권 모드는 Ring3로 구분하였다. 그림을 보면 가상머신 모니터와 가상머신은 VMX Root모드와 VMX non-Root 모드로 구분되고 그 안에서 Ring0와 Ring3 모드로 구분되는 것을 볼 수 있다. 이때 VMX Root 모드에서도 Ring0와 Ring3를 구분해 만들어 놓은 이유는 x86\_64 아키텍처에서 지원하는 명령어 중에



특권 모드 명령과 비특권 모드 명령의 경계가 모호한 명령어가 있기 때문이다.

VT-x가 지원하는 환경에서는 문제가 있는 명령어들이 실행될 때 게스트 운영체

제가 VTX non-Root 모드에서 가상머신 모니터에게 실행 제어가 넘어가도록

Ring0에서 트랩을 발생 시킬 수 있고, 제어를 넘겨받은 가상머신 모니터가

VMX-root모드에서 직접 처리 할 수 있다.



### 3. 관련연구

#### 3.1 RT-Xen

RT-Xen[6]은 기존의 하이퍼바이저들이 경성 실시간성을 지원하기 힘들기 때문에 새로운 스케줄링 알고리즘을 적용한 Xen 기반의 실시간성 지원 하이퍼바이저이다. Xen에서 기존에 지원하는 CREDIT1, CREDIT2는 실시간성을 지원하기에 부족하고 SEDF는 멀티코어 환경을 고려하지 않은 스케줄러이기 때문에 멀티코어 환경에서 경성 실시간성을 지원하기 위한 새로운 스케줄링 프레임워크를 적용하였다.

SEDF스케줄러를 기반으로 멀티코어 환경에서의 CPU load balancing을 추가하였고, 기존에 VCPU를 관리하기 위한 Run Queue와 Waiting Queue 두 개의 Queue로 관리되던 자료구조에서 Run Queue, Ready Queue, Replenish Queue 세 가지 Queue를 두고 I/O 작업으로 인해 우선순위가 밀리는 실시간 게스트 운영체제의 VCPU가 더 높은 우선순위를 가지고 다시 스케줄링 받을 수 있도록 하였다.

하지만 RT-Xen 연구도 한정된 하드웨어 자원을 사용 하는 상황을 가정하였기 때문에 산업용 로봇 제어 시스템처럼 자원을 게스트마다 독립적으로 사용 할 수 있는 환경에서는 1ms의 주기를 가지고 스케줄링 하는 RT-Xen 프레임워크는 스케줄링 간섭으로 인한 불필요한 오버헤드가 남아있어 적합하지 않다.



### 3.2 산업용 자동화 시스템 가상화

산업용 자동화 시스템 가상화 연구[7]에서는 산업용 제어 시스템을 가상화했을 때 기존에 Xen 하이퍼바이저에서 지연시간과 지터값 그리고 네트워크 사용량을 측정하고, 산업용 자동화 시스템이 요구하는 실시간성을 위한 QoS를 제공할 수 있는지 확인하였다.

Xen 하이퍼바이저는 여러 게스트 운영체제가 하드웨어 자원을 이용하기 위한 경쟁상황을 고려했기 때문에 멀티코어 환경에서는 코어마다 할당된 가상머신이 계속해서 바뀌게 된다. 하지만 산업용 제어 시스템에서는 이용할 수 있는 CPU코어보다 게스트 운영체제가 적은 수로 실행되기 때문에 관련연구에서는 게스트 운영체제의 실행이 서로 영향을 미치지 못하도록 CPU를 독립적으로 할당하였다. 다양한 종류의 산업용 제어시스템을 게스트 운영체제로 올리고 성능을 측정해본 결과 평균적으로 345us의 지연시간이 발생했고 이는 실험에 쓰인 산업용 제어시스템의 최소 실행 요구 시간이 1ms이었기 때문에 Xen을 사용하는 것이 적합함을 확인하였다.

하지만 극단적인 경성 실시간성을 요구하는 경우 평균 지연시간보다 최대 지연시간의 범위가 제한되기 때문에 최대지연시간에 대한 분석이 필요하며, CPU를 고정적으로 할당하더라도 가상머신 모니터는 계속해서 멀티코어 환경을 고려하며 불필요한 작업을 실행하기 때문에 가상머신 모니터를 수정하여 가볍게 동작하도록 만든다면 가상화로 인해 발생하는 오버헤드를 더 줄일 수 있다.



## 4. Xen 스케줄러 분석

Xen 하이퍼바이저에서 지원하는 기본 스케줄러는 CREDIT1 스케줄러이다. 이 외에도 CREDIT2, SEDF 스케줄러가 있으며 2015년 1월에 새로운 실시간 스케줄러인 RTDS 스케줄러를 추가 발표하였다. 각 스케줄러의 특성을 분석하여 스케줄링 과정에서 불필요한 동작으로 인해 CPU 가상화시 오버헤드를 증가시키는 원인을 찾고 해결하여 오버헤드를 줄여야 한다.

### 4.1 CREDIT1 스케줄러

CREDIT1 스케줄러는 CPU 균등 분할(proportional fair share) 및 작업 보장성(Work Conserving) 스케줄러이다. 하이퍼바이저의 스케줄러는 게스트 도메인에게 weight값과 cap값을 기준으로 CPU를 사용하게 한다. weight값은 도메인의 VCPU에게 credit을 부여하는 기준으로 credit 양만큼 물리 CPU를 사용할 수 있다. credit은 스케줄러가 30ms마다 각 게스트 도메인의 VCPU에게 할당해준다. 도메인이 CPU를 사용하고 있으면 10ms마다 스케줄러는 credit을 사용한 만큼 차감한다. 도메인은 기본 30ms의 time slice동안 CPU를 사용할 수 있으며 이 시간 안에 credit을 다 사용하지 못하면 해당 VCPU는 run queue에 가장 뒤쪽으로 이동한다. 매 30ms마다 VCPU에게 credit을 재분배 해줄 때 이전에 받은 credit을 전부 사용하지 못한 VCPU는 credit이 계속 쌓이고 임계치를 넘을 경우 비활성화 도메인으로 구분되어 바로 300credit을 소모시킨다.



스케줄러는 도메인에게 UNDER, OVER, BOOST 세 가지 우선순위를 부여하고 이에 따라서 run queue에서 정렬한다. UNDER는 credit을 가지고 있는 도메인의 상태이고, OVER는 credit을 모두 소모한 도메인의 상태, 그리고 BOOST는 I/O 작업으로 블록상태인 도메인이 이벤트를 받아 깨어난 상태이다. UNDER상태는 run queue에서 앞으로 정렬되고, OVER는 가장 뒤로 가며 BOOST는 run queue에서 가장 앞으로 삽입된다.

Credit 스케줄러는 작업 보장성을 위해서 도메인이 할당받은 CPU를 다 사용하고 있을 때 유휴 CPU가 있으면 추가로 CPU를 사용하게 해준다. 이때 도메인이 가지고 있는 cap값이 0으로 되어있으면 유휴 CPU를 전부 사용 할 수 있고, 값이 정해져 있다면 값에 따라서 사용할 수 있는 CPU의 양이 정해져 있다. 예를 들어 200의 cap값을 가지고 있는 VCPU는 물리 CPU 2개의 사용량을 넘어서 사용할 수 없다.

Credits케줄러는 credit 재분배시 남은 credit이 임계치인 300credit을 넘게 가지고 있는 도메인은 비활성화 상태로 놓고 가지고 있는 300credit을 소진시킨다. 이 때문에 CPU를 많이 사용하지 않는 실시간 게스트OS는 credit이 쌓여서 비활성화 상태가 되고, 실제로 작업을 해야 하는 상황에는 다른 VCPU와 경쟁하기 때문에 제대로 작업을 실행하지 못한다.[8]



## 4.2 CREDIT2 스케줄러

CREDIT2 스케줄러는 CREDIT1 스케줄러가 가진 단점을 보완하기 위해 만들어진 스케줄러이다.[9] 첫 번째로 run queue 정렬방식의 변화다. CREDIT1 스케줄러는 실행이 끝난 VCPU를 run queue에 라운드-로빈 방식으로 정렬해서 집어넣기 때문에 아무리 급한 실시간 작업을 실행할 도메인이라고 하더라도 다른 도메인들이 가지고 있는 credit을 전부 소모할 때까지 기다려야 했다. 실시간 작업을 하는 도메인에 weight를 부여하더라도 실시간 작업을 하는 도메인은 실제 CPU를 사용하는 시간이 짧기 때문에 I/O 작업을 할 때 블록상태가 되고 이때 wait queue에 들어가 대기하게 된다. 그렇게 되면 다른 일반 작업을 실행하는 도메인에게 우선순위가 밀리게 되고 실제로 CPU를 사용해야하는 시점이 되었을 때 우선순위 역전으로 인해 CPU를 사용하지 못하게 된다. 이러한 단점을 극복하기 위해서 CREDIT2 스케줄러는 스케줄링 타이밍을 기존의 30ms에서 500us로 줄였으며 run queue 정렬시 라운드-로빈 으로 정렬하지 않고 해당 도메인이 가지고 있는 credit의 잔량에 따라서 다시 정렬 시킨다. 그리고 weight값에 따라서 credit을 부여하지 않고 모든 도메인이 초기 credit은 256 credit으로 같은 값을 받고 weight값에 따라 소모되는 비율이 달라지게 하였다. 이렇게 함으로써 credit1스케줄러에서 문제가 되었던 실시간작업을 하는 게스트 도메인이 가지고 있는 credit의 양이 임계치를 넘겨 비활성 상태가 되는 문제를 해결 할 수 있다. credit의 재분배는 무조건 30ms마다 일어나는 credit1스케줄러와 달리 credit2에





서는 어떤 도메인이더라도 가지고 있는 credit을 모두 소진한다면 모든 VCPU의 credit값을 초기 값으로 설정한다. 이렇게 되면 실시간 작업을 실행하는 도메인이 다른 일반 작업을 실행하는 도메인에 비해 credit을 소모하지 못하더라도 다시 초기 256 credit값을 가지기 때문에 credit 재분배후 우선순위 역전 현상이 발생하지 않는다.



### 4.3 RTDS 스케줄러

RTDS 스케줄러는 Real-Time Deferrable Server 스케줄러로 Xen 기본 스케줄러인 CREDIT1스케줄러와 CREDIT2스케줄러가 실시간성을 지원하기에 적합하지 않아 RT-Xen에서 제안한 스케줄러이다. Xen 4.5에서 공식 지원하기 시작하였다.[10]

RTDS스케줄러는 period와 slice 값을 가지고 게스트 도메인의 VCPU를 스케줄링 한다. period는 게스트 도메인이 설정된 시간동안 CPU를 다른 도메인에게 선점당하지 않도록 보장하는 시간의 길이이다. slice는 설정된 period 시간동안에 실제로 물리 CPU를 사용할 수 있는 시간을 말한다. 따라서 slice값은 period값을 넘어서 설정 할 수 없으며, period시간이 지날 때까지 slice를 전부 사용하지 못하면 다음 period에 slice는 초기화된다.

여러 개의 도메인이 서로 CPU를 사용하려고 경쟁하는 상황일 경우 도메인에 설정된 period값에서 남은 slice값의 차이로 각 도메인의 데드라인을 계산하고 데드라인이 짧은 순으로 먼저 실행하기 때문에 우선순위가 결정된다.

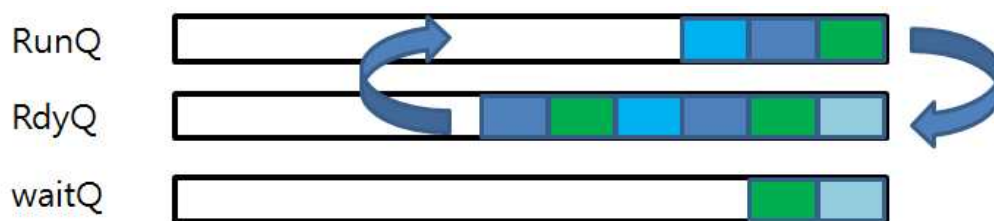


그림4. RTDS 스케줄러의 VCPU 관리 자료구조

그림4는 RTDS 스케줄러가 VCPU를 관리하기 위해 추가한 VCPU 리스트 자료



구조이다. 기존에 Xen 스케줄러에서는 run queue와 wait queue를 가지고 VCPU를 관리하였다. 하지만 2개의 queue를 가지고 VCPU를 관리하면 I/O 작업을 위해 VCPU가 블록 상태가 되었을 때 wait queue에 들어가 있는데 스케줄링을 늦게 받게 된다. 이것을 막기 위해 ready queue를 추가하여 스케줄러에서 VCPU를 run queue에 정렬할 때 ready queue에 있는 VCPU까지 고려하여 스케줄링 하게 되고 실시간 게스트 운영체제가 더 빠르게 CPU를 사용 할 수 있게 된다.



## 4.4 SEDF 스케줄러

Simple Earliest Deadline First (SEDF) CPU 스케줄러는 EDF 실시간 스케줄러를 수정하여 Xen에 적용한 것이다. 특정 도메인의 CPU 사용 요구를 보장하기 위해 slice, period, extra-flag 세 가지 값을 가지고 CPU사용량을 결정해준다. 스케줄러는 각 도메인에게 부여된 period마다 slice값에 비례하는 CPU사용시간을 부여해준다. 그리고 Extra flag가 0, 1로 설정되어있는지에 따라서 work conserving mode(WC-mode)를 지원할지 안할지 결정한다. WC-mode일 경우 유휴상태에 있는 CPU를 추가로 사용 할 수 있게 해준다. SEDF 스케줄러는 VCPU를 스케줄링을 위해 run queue에서 실행을 기다리고 있는 VCPU중에서 가장 데드라인 짧은 것, 즉 period가 얼마 남지 않은 도메인이 먼저 실행 될 수 있도록 정렬한다. 그리고 VCPU가 현재 남은 period동안 사용할 수 있는 CPU 사용량이 얼마나 되는지 확인하여서 블록상태에서 깨어났을 때 남아있는 slice를 사용하게 한다.

SEDF스케줄러는 period와 slice값에 따라서 CPU를 할당해주기 때문에 period 값이 차이가 나면 fairness를 보장할 수 없다. 예를 들어 똑같이 30%의 CPU사용을 할당받더라도 slice=3ms, period=10ms, extra=0 으로 받은 도메인과 slice=30, period=100ms, extra=0으로 받은 도메인이 CPU를 사용하는데 차이가 생긴다.



## 4.5 스케줄러 오버헤드 분석

Xen 하이퍼바이저가 게스트 운영체제를 스케줄링 할 때 발생하는 오버헤드 원인을 분석해보면 다음과 같다. Xen 하이퍼바이저는 스케줄링을 위해 타이머 인터럽트를 설정하여 정해진 시간마다 `schedule()` 함수를 호출한다. 이때 타이머 인터럽트를 설정하는 시간의 크기는 `do_schedule()` 함수를 호출하여 각 스케줄러 인터페이스에 맞춰서 다르게 설정된다. 기본 설정으로 CREDIT1 스케줄러는 30ms, CREDIT2 스케줄러는 500us, RTDS 스케줄러는 1ms, SEDF 스케줄러는 100us 마다 스케줄러 함수를 호출하여 스케줄링을 하게 된다. 각 스케줄러가 호출될 때마다 VCPU를 관리하기 위해 할당한 값들을 계산하고, 다 소모하였을 경우 재분배하는 연산을 실행하기 때문에 불필요한 오버헤드가 발생할 것이다. 그리고 CREDIT1, CREDIT2, RTDS 스케줄러의 경우 멀티코어 환경을 고려하여 유휴 CPU가 있을 경우 로드밸런싱을 위해 각 코어의 run queue를 검사하고 VCPU 리스트를 재 정렬하기 위한 연산 때문에 오버헤드가 더 발생할 것이다.

산업용 로봇 제어 시스템 가상화 환경에서는 게스트 운영체제가 하드웨어 자원을 충분히 독립적으로 사용 할 수 있기 때문에 로드밸런싱 함수 같은 불필요한 함수를 제거하면 오버헤드가 감소 할 것이다.



## 5. Xen 스케줄러 지연시간 측정 및 개선

### 5.1 실험 환경

실험환경을 구성하기 위한 하드웨어 플랫폼에 CPU는 Intel® Core i7-2600 3.40GHz을 사용하였으며 코어는 4개이고, 하이퍼-쓰레딩 기능은 비활성화 되었다. 마더보드는 ASUSTeK COMPUTER INC.의 P8Z77 WS 이다. 메인 메모리의 크기는 16GB이며 디스크는 WD500AAKX 500GB를 사용했다. 가상화를 위해 Xen 하이퍼바이저는 4.5.0 버전을 사용했으며 게스트 운영체제는 Dom0 커널은 Ubuntu 15.04에 3.19.0-15 버전을 올렸고, DomU-RT에는 Ubuntu 12.04에 3.10.32 버전 커널에 RT-patch 3.10.70-rt75 버전을 적용했다. DomU-GP는 Windows 7 Enterprise K를 사용했다.

지연시간을 측정하기 위해 DomU-RT의 실시간 리눅스에서 cyclicttest를 실행하여 지연시간을 측정하였고 cyclicttest는 0.91 버전을 컴파일 하여 설치하였다.



## 5.2 실험 요약

산업용 로봇 제어 시스템인 EtherCAT을 가상화 했을 때 오버헤드가 얼마나 증가하는지 측정하였다. 가상화를 위해 Xen 하이퍼바이저 위에 실시간 리눅스 도메인과 window7 도메인을 올리고 실시간 리눅스 도메인에서 cyclicttest를 실행하여 지연시간을 측정하였다. cyclicttest는 주기적인 시간마다 타이머 인터럽트를 발생 시켜서 프로세스가 sleep상태에서 깨어난 현재시간과 타이머 설정 시간의 차이를 계산하여 지연시간을 측정하는 벤치마크이다. cyclicttest의 지연시간 계산식은 다음과 같다.

$$\text{Latency} = \text{TIME}_{\text{curr}} - \text{TIME}_{\text{set}}$$

Xen 4.5 버전에서 지원하는 CREDIT1, CREDIT2, RTDS, SEDF 네 가지 스케줄러를 변경하며 지연시간을 측정하였다. CPU가상화 오버헤드 외에 다른 영향을 최소화하기 위해 I/O는 PCI-passthrough로 게스트 도메인들이 직접 이용하도록 하였으며 실시간 리눅스 도메인과 windows7 도메인 각각에게 독립적으로 코어를 할당하여 서로의 작업 실행이 다른 도메인에게 영향을 미치지 않도록 하였다. 그림5는 하이퍼바이저와 도메인에 CPU 코어가 어떻게 할당되었는지 도식화 한 것이다.

또한 수정하지 않은 Xen 하이퍼바이저에서 측정한 지연시간과 불필요한 동작을 제거한 Xen 하이퍼바이저에서 측정한 지연시간을 비교하여 오버헤드가 감소한 것을 확인하였다.



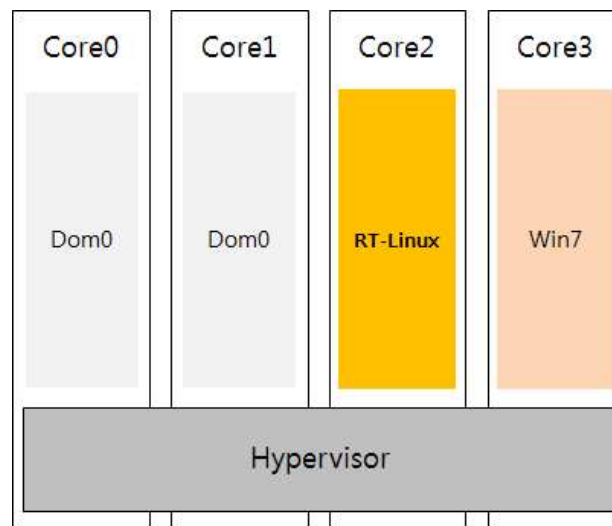


그림5. Xen 하이퍼바이저와 도메인 CPU할당



### 5.3 Xen 하이퍼바이저 스케줄러별 지연시간 측정

Native 리눅스에서 cyclicttest를 실행하여 측정한 지연시간과 Xen 하이퍼바이저 위에 게스트 리눅스에서 cyclicttest를 실행할 때 측정한 지연시간을 비교하여 가상화로 인해 증가한 지연시간을 확인하고 그 원인을 분석하기 위한 실험이다. Xen에서 지원하는 CREDIT1, CREDIT2, RTDS, SEDF 스케줄러 네 가지를 변경하며 가상화 오버헤드가 얼마나 되는지 확인하였다.

스케줄러	Native	CREDIT1	CREDIT2	RTDS	SEDF
최소 지연시간(us)	1	5	6	6	4
평균 지연시간(us)	3	27	22	26	12
최대 지연시간(us)	15	133	157	168	116

표1. 스케줄러 변경에 따른 지연시간 측정

표1은 스케줄러 변경에 따른 지연시간 측정 결과이다. Native 리눅스와 Xen 하이퍼바이저에서 제공하는 네 가지 스케줄러를 변경하며 측정한 최소 지연시간과 평균 지연시간 그리고 최대 지연시간이 나와 있다. 실시간성 지원을 위해서는 최대 지연시간의 크기가 작아야 실시간 작업의 테드라인을 보장 할 수 있기 때문에 최대 지연시간의 크기를 비교해 본다. 측정 결과를 보면 Xen 하이퍼바이저 스케줄러별로 Native와 비교했을 때 CREDIT1 스케줄러는 118us, CREDIT2 스케줄러는 142us, RTDS 스케줄러는 153us, SEDF 스케줄러는 101us 만큼 지연시간이 증가한 것을 확인 할 수 있다. 실험 환경은 Xen 하이퍼바이저 가상화가 추가 된 것을 제외하고 실험 머신의 하드웨어와 리눅스 버전 그리고 실시간 패치는 모두 동일하기 때문에 증가한 오버헤드는 가상화로 인한 오버헤드이다. 최소 101us에서



최대 153us까지 평균 143.5us 만큼 오버헤드가 증가한 것을 볼 수 있다. 현재 실험환경은 cyclicttest만 실행한 최소 작업 환경임을 고려했을 때 1ms 의 작업주기를 갖는 EtherCAT과 같은 산업용 로봇 제어 시스템의 경우 실제 실행시간을 847us에서 899us까지 만큼 확보 할 수 있는 것을 얘기한다.

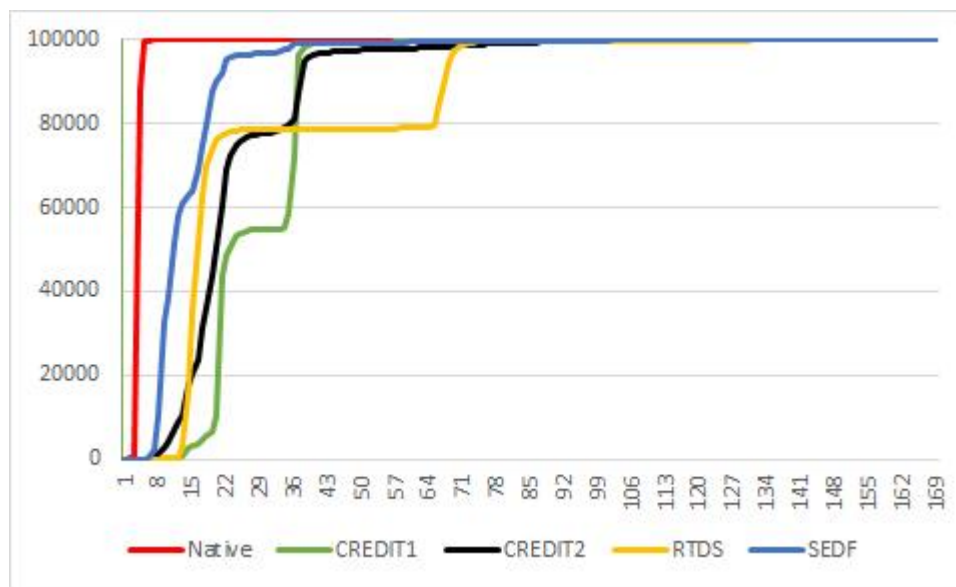


그림6. Native 리눅스와 Xen 스케줄러별 게스트 리눅스의 지연시간 비교

그림6을 보면 cyclicttest를 10만 번 실행하는 동안 발생한 지연시간 크기별 빈도수를 볼 수 있다. 모든 스케줄러에서 평균크기 이상의 지연시간을 관찰 할 수 있는데 각 스케줄러가 게스트 운영체제의 CPU사용을 관리하기 위한 알고리즘에서 불필요한 동작을 거치게 될 때마다 오버헤드가 증가하기 때문이다.



## 5.4 로드밸런싱 제거 후 지연시간 측정

Xen 하이퍼바이저는 도메인 간에 CPU사용 경쟁을 고려하여 만들어졌기 때문에 불필요한 동작을 제거하면 오버헤드를 줄일 수 있다. 이 실험은 CREDIT1 스케줄러에서 로드밸런싱을 하지 않도록 하고 지연시간을 측정한 것이다.

스케줄러	NATIVE	CREDIT 1	No Load Balancing
최소 지연시간(us)	1	5	4
평균 지연시간(us)	3	27	23
최대 지연시간(us)	15	133	119

표2. 로드밸런싱 제거한 CREDIT1 스케줄러의 지연시간 측정

표2에서 보이는 바와 같이 CREDIT1 스케줄러에서 로드밸런싱 기능을 비활성화 시키고 지연시간을 측정하면 수정하지 않은 CREDIT1 스케줄러와 비교했을 때 최소 지연시간은 1us, 평균 지연시간은 4us 그리고 최대 지연시간은 14us 만큼 감소한 것을 확인 할 수 있다. 그림 7을 보더라도 모든 구간에서 지연시간을 감소시키는 현상을 볼 수 있다.

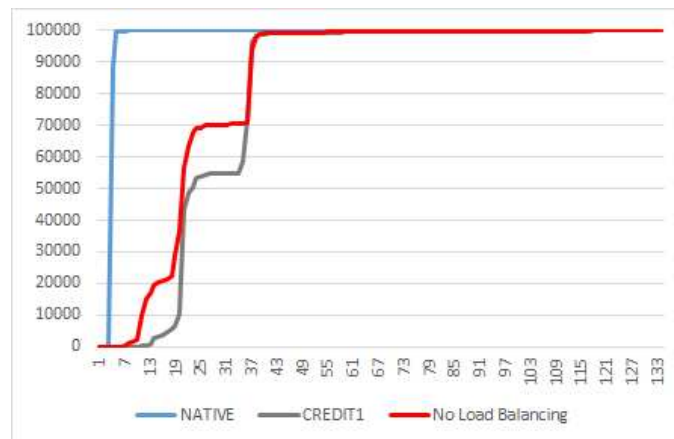


그림7. CREDIT1에서 로드밸런싱 제거 후 지연시간



## 6. 결론

본 연구에서는 일반적인 가상화 솔루션인 Xen을 산업용 로봇 제어 시스템인 EtherCAT을 가상화 했을 때 발생하는 오버헤드의 원인을 분석하고 해결 방법을 제시하여 가상화로 인한 성능저하를 최소화 하고자 하였다. 산업용 로봇이 극단적인 경성 실시간성을 요구하기 때문에 오버헤드의 크기를 줄여야 테드라인 내에 작업을 완료 할 수 있다.

Xen 하이퍼바이저의 가상화 오버헤드 요소는 CPU가상화, 메모리 가상화, I/O 가상화 세 가지가 있지만 메모리 가상화와 I/O 가상화 오버헤드는 거의 없기 때문에 CPU 가상화 오버헤드가 주요한 원인이다. 따라서 가상머신의 VCPU를 스케줄링 하는 과정에서 일어나는 불필요한 동작들을 제거함으로써 오버헤드를 최소화 할 수 있다. 이 오버헤드를 측정하기 위해 cyclicttest를 사용하여 지연시간을 측정하였고, 불필요한 동작들을 찾고 제거하기 위해 Xen 하이퍼바이저의 코드를 수정하여 로드밸런싱 기능을 제거하고 cyclicttest로 지연시간을 측정한 결과 기존 Xen의 CREDIT1 스케줄러와 비교하였을 때 최대 지연시간의 크기가 14us 만큼 감소한 것을 확인 할 수 있었다. 하지만 Native 리눅스에서 측정한 최대 지연시간과 비교하였을 때 104us만큼 지연시간이 증가하였기 때문에 차후 로드밸런싱 외에 불필요한 동작을 제거하여 지연시간을 더 줄여야 한다. 또한 실제로 EtherCAT의 작업을 수정된 Xen에서 실행하여 결과를 확인해야 한다.



## 참고 문헌

- [1] Menascé, Daniel A. "Virtualization: Concepts, applications, and performance modeling." Int. CMG Conference. 2005.
- [2] A performance analysis of EtherCAT and PROFINET IRT
- [3] Song, Il-Seuk, et al. "Implementation and analysis of the embedded master for EtherCAT." Control Automation and Systems (ICCAS), 2010 International Conference on. IEEE, 2010.
- [4] Barham, Paul, et al. "Xen and the art of virtualization." ACM SIGOPS Operating Systems Review 37.5 (2003): 164-177.
- [5] Uhlig, Rich, et al. "Intel virtualization technology." Computer 38.5 (2005): 48-56.
- [6] Xi, Sisu, et al. "Rt-Xen: Towards real-time hypervisor scheduling in Xen." Embedded Software (EMSOFT), 2011 Proceedings of the International Conference on. IEEE, 2011.
- [7] Mahmud, Nesredin, Kristian Sandstrom, and Aneta Vulgarakis. "Evaluating industrial applicability of virtualization on a distributed multicore platform." Emerging Technology and Factory Automation (ETFA), 2014 IEEE. IEEE, 2014.
- [8] Hnarakis, Ryan. In Perfect Xen, A Performance Study of the Emerging Xen



Scheduler. Diss. California Polytechnic State University San Luis Obispo, 2013.

- [9] Dunlap, George W. "Scheduler development update." Xen Summit Asia (2009).
- [10] Xi, Sisu, et al. "Real-time multi-core virtual machine scheduling in Xen." Embedded Software (EMSOFT), 2014 International Conference on. IEEE, 2014.

