실시간 클라우드 서버를 위한 Virtual Runtime 보상 기반 가상 머신 공정 스케줄링 기법

김경래*, 노순현*, 홍성수*
*서울대학교 전기·정보공학부
e-mail : {krkim, shnoh, sshong}@redwood.snu.ac.kr

Fair-Share Scheduling of Virtual Machines via Virtual Runtime Compensation for Real-time Cloud Servers

Kyungrae Kim*, Soonhyun Noh*, Seongsoo Hong*
*Dept. of Electrical and Computer Engineering, SNU

요 약

가상화 기술을 기반으로 하는 클라우드 컴퓨팅 환경에서 실시간 응용들을 수행하려는 움직임이 많아지고 있다. 이 응용들의 실시간성을 보장하기 위해 리눅스의 실시간 스케줄러를 활용한 기법들이 제안되었지만, 이 기법들은 가상 머신들의 공정한 수행을 보장하지 못한다는 한계가 있다. 따라서 우리는 실시간 클라우드 환경에서 가상 머신 스케줄링의 공정성을 보장하기 위해 virtual runtime 보상 기반의 가상 머신 공정 스케줄링 기법을 제안한다. 제안된 기법은 실시간 응용으로 인해 CFS 가 인지하지 못한 CPU 시간을 체크하고 이를 virtual runtime 에 반영하여 가상 머신 간의 공정한 수행을 보장한다. 우리는 제안된 기법을 리눅스 커널 상에 구현하였다. 제안된 기법을 적용한 결과, 가상 머신 간의 수행시간 차이가 기존 경우보다 47% 줄어든 것을 확인하였다.

1. 서론

클라우드 컴퓨팅은 사용자의 요청에 따라 컴퓨팅 자원을 네트워크를 통해 제공하는 컴퓨팅 기술이다. 이 기술은 컴퓨팅 자원을 탄력적으로 배치하고 하드웨어 유지 비용을 감소시킬 수 있기 때문에 대규모의 컴퓨팅 자원이 필요한 응용들을 위한 기반 기술로 발전하였다.

가상화는 단일 물리 머신의 자원을 복수의 가상 머신들이 공유할 수 있는 서버 통합 기술이다. 가상화 환경에서는 가상 머신마다 동작하는 서로 다른 게스트 운영체제들의 고립된 성능이 보장되어야 한다. 클라우드 컴퓨팅의 기반 인프라인 클라우드 데이터센터는 이 가상화 기술을 통해 클라우드 서비스를 위한 컴퓨팅 자원을 가상 머신 단위로 관리한다. 구체적으로, 사용자와의 SLA(service level agreement)에 따라 가상 머신에 가중치를 부여하면, 하이퍼바이저가 가중치에 따라 가상 머신들을 공정하게 스케줄링함으로써 SLA 를 보장한다.

최근 대규모 컴퓨팅 자원을 사용하는 실시간 응용이 많아짐에 따라 이들 응용을 클라우드에서 수행하려는 움직임이 많아지고 있다[1-3]. 모바일 기기의 GPS 위치정보 계산을 클라우드를 통해 수행하거나[2], 카메라의 스캔 정보를 클라우드에서 실시간으로 분석하여 관련 정보를 제공하는 서비스들이 이러한 움직임을 보여준다[3]. 현재의 클라우드 환경에서 이러한 응용들을 수행하기 위해서는 다음과 같은 새로운 기능들이 요구된다. 우선 클라우드 관리자가 요청된 응용의 실시간성과 물리 머신들의 자원 상황을 고려하여 가상 머신을 할당해야 하며, 개별 물리 머신에서는 하이퍼바이저를 통해 가상 머신들을 실시간으로 스케줄링해야한다.

이 중 하이퍼바이저를 통해 개별 물리 머신에 가상 머신을 실시간으로 스케줄링하는 연구는 서버 환경에서 가장 범용적으로 활용되는 운영체제인 리눅스를 중심으로 많이 진행되었다. 그 중 대표적인 방법이 실시간 태스크를 수행하는 실시간 가상 머신에 리눅스가 제공하는 실시간 스케줄링 정책을 적용하는 방법이다. J. Kiszka[4]는 단일 실시간 가상 머신이 다른 일반 가상 머신들과 물리 자원을 공유하는 환경에서 리눅스가 실시간 태스크의 지원을 위해 제공하는 스케줄링 정책을 적용한 가상 머신 스케줄링 기법을 제안하였다. 리눅스에서 응용과 가상 머신은 모두 태스크들로 매핑된다. 이 기법은 가상 머신에서 실시간 태스크가 수행되는 동안, 가상 머신이 매핑된 태스크에 실시간 스케줄링 정책을 적용한다. 이후 가상 머신이 실시간 태스크의 수행을 마치면 다시 공정 스케줄러인 CFS(completely fair scheduler)에 의해 스케줄링된다. 또한 B. Zuo[5]는 복수의 실시간 가상 머신들이 일반 가상 머신들과 같이 스케줄링되는 시스템에서

실시간 가상 머신의 실시간성을 보장하기 위한 스케줄링 기법을 제안하였다.

하지만 이 기법들은 가상 머신의 실시간 태스크가 수행되는 동안, 다른 가상 머신과의 공정한 수행을 보장하지 못한다는 치명적인 단점이 있다. 실시간 태스크를 우선 수행하는 동안 소모된 자원을 공정 스케줄링 과정에서 고려하지 않기 때문이다. 가상 머신의 공정한 수행을 보장하지 못한다면 클라우드 컴퓨팅 환경에서 가장 중요한 사용자의 성능 요구 조건을 만족할 수 없게 된다.

이와 같은 기존 연구의 문제점을 해결하기 위해서, 본 논문에서는 실시간 가상화 환경에서 가상 머신 수행의 공정성을 보장하기 위해 실시간 태스크 수행 시간을 보상하는 기법을 제안한다. 이를 위해 본 연구진은 가상 머신이 실시간 태스크를 수행하면서 소모한 자원을 인지할 수 있는 인터페이스를 통해 이 시간을 공정 스케줄링에 반영하는 스케줄링 기법을 제안한다.

우리는 제안한 기법을 KVM 이 탑재된 리눅스 커널 에 구현하고 일련의 실험을 통해 응용의 실시간성을 만족하면서 가상 머신이 공정하게 수행될 수 있음을 확인하였다.

본 논문의 나머지 부분은 다음과 같이 구성된다. 2 장에서는 리눅스 스케줄러의 동작 원리를 분석한다. 3 장에서 대상 시스템 모델과 본 논문이 해결하고자 하는 문제를 설명한다. 4 장에서는 제안하는 기법에 대해 자세히 설명하고, 5 장에서 실제 리눅스 커널에 적용한 실험 결과를 설명한다. 마지막으로 6 장에서 논문의 결론을 맺는다.

2. 리눅스 스케줄러

리눅스는 2.6.23 커널 이래로 실시간 스케줄러과 공정 스케줄러(CFS)를 모두 제공한다. 실시간 스케줄러는 우선순위 기반의 태스크 스케줄링 정책이며, CFS 는 태스크의 가중치에 비례하여 CPU 시간을 할당하는 스케줄링 정책이다. 각 스케줄러는 자신의 실행큐를 가지고, 실행큐에 대기 중인 태스크를 CPU 에 스케줄링한다. 두 실행큐에 동시에 태스크가 대기 중일 경우 실시간 실행 큐에 있는 태스크를 우선하여 스케줄링한다.

2.1. 실시간 스케줄러

실시간 태스크에는 0 부터 99 사이의 우선순위가 부여된다. 수치가 낮을수록 높은 우선순위를 의미한다. 동일한 우선순위에 가진 태스크들에 대해서는 FIFO 또는 Round-Robin 정책을 사용하여 스케줄링한다.

2.2. CFS

CFS 태스크의 가중치는 nice 값을 사용하여 표현된다. CFS 는 태스크의 상대적인 진행 정도를 파악하기 위하여 virtual runtime 이라는 개념을 사용한다[6]. 어떤 시간 t 에 태스크 τ_i 의 virtual runtime $VR(\tau_i, t)$은 아래와 같이 정의된다.

$$VR(\tau_i,t) = \frac{\omega_0}{\omega_i} \times PR(\tau_i,t)$$

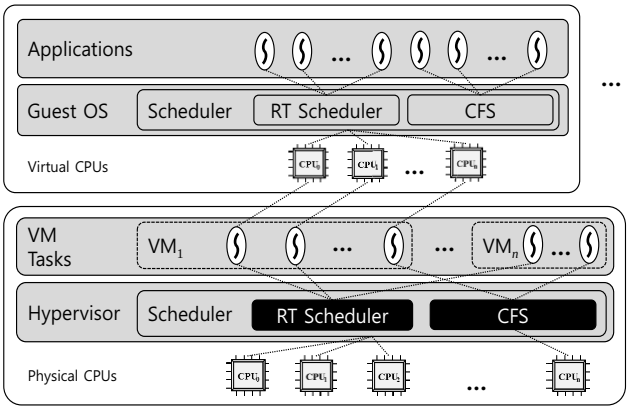
여기서 ω_0, ω_i 는 각각 nice 값이 0 인 태스크와 태스크 τ_i 의 가중치를 의미하고, $PR(\tau_i, t)$는 시간 t 일 때 태스크 τ_i 의 실제 누적 실행시간을 의미한다[7]. CFS 는 스케줄링 주기마다 수행된 태스크의 virtual runtime 을 갱신하고, 가장 작은 virtual runtime 을 갖는 태스크를 다음 수행할 태스크로 선택한다. 새롭게 스케줄링된 태스크는 자신의 가중치에 비례한 CPU 시간동안 수행된다.

3. 대상 시스템 모델과 문제 정의

이 논문에서 제안된 기법의 대상 시스템 모델은 그림 1 과 같다. 가상 머신과 물리 머신은 각자 실시간 스케줄러와 CFS 를 사용하여 태스크를 스케줄링한다. 가상 머신에서 수행되는 태스크는 게스트 스케줄러에 의해 가상 CPU 에 스케줄링된다. 가상 CPU 는 하이퍼바이저에 의해 물리 머신의 태스크로 매핑되어 호스트 스케줄러에 의해 물리 CPU 에 스케줄링된다.

가상 CPU 는 기본적으로 하이퍼바이저의 CFS 에 의해서 공정하게 스케줄링된다. 하지만 가상 머신에서 실시간 태스크가 수행되면, 실시간 가상 머신 스케줄링에 의해 가상 CPU 를 하이퍼바이저의 CFS 실행큐로부터 실시간 실행큐로 옮긴다. 따라서 실시간 태스크가 수행되는 동안 해당 가상 CPU 에는 실시간 스케줄링 정책이 적용되고, 수행을 마친 후에 가상 CPU 는 다시 CFS 실행큐로 이전된다.

가상 CPU 가 실시간 태스크를 수행하는 동안 점유한 CPU 시간은 virtual runtime 에 반영되지 않는다. 이 경우, 해당 가상 CPU 는 공정하게 받을 수 있는 시간보다 더 많은 CPU 시간을 할당 받게 되어 가상 머신 수행의 공정성을 해치게 된다. 따라서 우리는 가상 CPU 에서 수행되는 실시간 태스크의 실시간성을 보장하며 가상 머신 간의 공정성을 유지할 수 있는 스케줄링 기법을 개발하고자 한다.



(그림 1) 대상 시스템 모델

4. Virtual Runtime 보상 기반 가상 머신 공정 스케줄링 기법

앞서 설명한 문제를 해결하기 위해 우리는 virtual runtime 을 보상 기반 가상 머신 공정 스케줄링을 제안한다. 본 장에서는 virtual runtime 보상 기반 가상 머신 공정 스케줄링 기법에 대해 자세히 설명한다. 구체적으로, 제안된 기법은 다음의 두 가지를 수행한다. 1) 가상 CPU 가 실시간 태스크를 수행하는 동안 CPU 를 점유한 시간을 모니터링하고, 2) 점유한 시간만큼 해당 가상 CPU 의 virtual runtime 을 증가시켜 CFS 의 공정성을 보장한다. 본 장의 나머지 부분에서는 두 기법에 대해 자세히 설명한다.

4.1. 실시간 가상 머신의 CPU 소모 모니터링

제안된 스케줄링 기법은 실시간 태스크의 수행으로 인해 기존의 CFS 가 인지하지 못한 CPU 점유 시간을 인지해야 한다. 따라서 가상 CPU 가 실시간 실행큐에 이전될 때마다 실시간 스케줄링을 통해 CPU 를 점유한 시간을 기록한다. 구체적으로 이 작업은 가상 CPU 가 스케줄링 된 시점으로부터 CPU 를 빼앗기는 시점까지의 시간들을 누적한다. 실시간 실행큐로 옮겨진 시점 t_s 로부터 누적된 수행 시간 $\Delta PR_{RT}(\tau_i,\ t)$은 아래와 같다.

$$\Delta PR_{RT}(\tau_i,t) = PR_{RT}(\tau_i,t) - PR_{RT}(\tau_i,t_s)$$

이 시간은 가상 CPU 가 CFS 실행큐로 이전될 때 CFS 에 전달된다.

4.2. 실시간 태스크 수행 시간의 Virtual Runtime 환산

실시간 스케줄러에 의해 가상 CPU 가 수행된 시간을 전달받아, $\Delta PR_{RT}(\tau_i,\ t)$만큼의 virtual runtime 증가분을 가상 CPU 의 virtual runtime 에 더한다. Virtual runtime 증가분이 더해진 virtual runtime $VR_{total}(\tau_i,\ t)$ 은 아래와 같이 계산된다.

$$VR_{total}(\tau_i,t) = VR(\tau_i,t) + \frac{\omega_0}{\omega_i} \times \Delta PR_{RT}(\tau_i,t)$$

가상 CPU 가 실시간 태스크의 수행을 마친 후에 virtual runtime 을 $VR(\tau_i,\ t)$ 에서 $VR_{total}(\tau_i,\ t)$ 로 대체한다. 그 후 CFS 는 갱신된 virtual runtime 를 바탕으로 가상 머신을 스케줄링한다. 결과적으로 실시간 태스크를 수행한 가상 머신은 수행된 시간에 비례하여 다른 가상 머신보다 적은 CPU 시간을 할당 받게 된다. 따라서 CFS 는 가상 머신 간 공정한 스케줄링을 수행할 수 있게 된다.

5. 실험 및 검증 결과

제안된 가상 머신 공정 스케줄링 기법의 성능을 평가하기 위해서 실험을 수행하였다. 실험 환경은 표 1 과 같다.

클라우드 서버 환경에서 공정성을 평가하기 위해 단일 물리 머신에서 복수의 가상 머신을 생성하였으며, 리눅스 커널의 대표적인 하이퍼바이저인 KVM 을 선택하였다.

<표 1> 실험 환경

항목	내용
CPU	3.40GHz 64-bit Intel i7-4770
메모리	32GB
호스트 운영체제	리눅스 3.16.3 커널
하이퍼바이저	Kernel Virtual Machine (KVM)
게스트 운영체제	리눅스 3.16.3 커널

5.1 실험 응용

우리는 주기적으로 실시간 태스크를 수행하는 가상 머신과 CFS 태스크만을 수행하는 가상 머신을 실험 응용으로 재현하였다. 가상 머신에서 수행된 태스크의 명세는 표 2 와 같다.

<표 2> 실험 응용

가상 머신 A	·우선순위 50, 주기 5초의 실시간 태스크 ·CPU 집약적인 nice 값 0의 CFS 태스크
가상 머신 B	·CPU 집약적인 nice 값 0의 CFS 태스크

[5]의 기법이 구현된 리눅스 커널과 제안된 기법이 추가로 구현된 리눅스 커널에서 각각 두 응용을 동시에 5 분간 수행하고 태스크가 수행된 가상 CPU 의 누적 실행 시간의 차이 $\Delta PR(t)$ 를 측정하였다.

가상 머신 A, B 모두 하나의 가상 CPU 만을 가지며, 두 가상 CPU 의 가중치는 nice 값 0 의 가중치로 같다. 호스트 운영체제의 로드 밸런싱으로 인한 영향을 막기 위해서 두 가상 CPU 를 동일한 물리 CPU 코어에 고정하여 실험을 수행하였다.

5.2 실험 결과

(그림 2) 시간에 따른 누적 수행 시간 차이

그림 2 은 시간의 흐름에 따른 $\Delta PR(t)$를 확인한 결과이다. 제안된 기법이 적용된 경우는 두 가상 머신 사이의 CPU 점유시간 차이가 적용 전에 비해 47% 줄어드는 것을 확인하였다. 기존의 [5]의 기법만이 적용된 경우는 $\Delta PR(t)$가 선형적으로 증가하는 반면, 시간에 따라 $\Delta PR(t)$의 증가폭이 점점 감소하며 두 가상 CPU 가 점점 공정하게 스케줄링됨을 확인할 수 있다.

6. 관련 연구

하이퍼바이저를 통해 개별 물리 머신에 가상 머신을 실시간으로 스케줄링하기 위해서 정적으로 가상 CPU 에 실시간 속성을 부여하는 방법과 실시간 태스크 수행시 동적으로 가상 CPU 에 실시간 속성을 부여하는 방법들이 활발하게 연구되어 왔다. 전자에는 가상 머신 할당 단계에서 실시간 태스크를 수행할 가상 CPU 에 실시간 우선순위를 부여하고 해당 CPU 에 매핑된 태스크를 독립된 큐에서 우선적으로 스케줄링하였다[8, 9]. 그러나 이들 연구는 제한적인 실시간 스케줄링 방법으로, 동적으로 태스크가 수행되는 클라우드 환경에서 실시간성을 보장할 수 없다.

후자의 연구들은 동적으로 수행되는 태스크의 실시간성 보장을 위해 태스크의 실시간 속성을 가상 CPU 에 전달하고, 하이퍼바이저가 해당 가상 CPU 들을 실시간 스케줄링하는 것에 중점을 두었다. Xen 하이퍼바이저에서는 SEDF(Simple Earliest Deadline First) 스케줄러를 통해 동적인 실시간 스케줄링을 지원한다[10]. 또한 리눅스가 실시간 태스크의 지원을 위해 제공하는 스케줄링 정책을 적용한 가상 머신 스케줄링 기법들이 제안되었다[4, 5]. 하지만 이들 연구들은 실시간 태스크를 수행하는 가상 머신과 다른 가상 머신들을 공정하게 수행하지 못한다는 치명적인 단점이 있다.

7. 결론

본 논문에서는 가상 머신의 공정한 수행이 필수적인 클라우드 환경에서 실시간 응용의 실시간성을 보장하면서 동시에 가상 머신을 공정하게 수행할 수 없던 문제를 해결하고자 virtual runtime 보상 기반의 가상 머신 공정 스케줄링 기법을 제안했다. 제안된 기법은 기존의 실시간 가상 머신 스케줄링 기법들이 가상 머신의 공정한 수행을 보장하지 못하는 한계를 가상 CPU 가 실시간 응용을 수행한 시간만큼 virtual runtime 을 증가시켜 극복한다. 리눅스 커널에 구현하여 실험한 결과, 제안된 기법은 가상 CPU 들이 공정하게 물리 CPU 에 스케줄링 되는 것을 확인하였다. 향후 연구로서, 우리는 제안된 알고리즘을 확장하여 I/O 작업으로 인해 virtual runtime 이 변경되는 경우에 대해서도 가상 머신 스케줄링의 공정성을 보장할 예정이다.

Acknowledgement

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학 ICT 연구센터육성 지원사업의 연구결과로 수행되었음(IITP-2015-(H8501-15-1015)).

참고문헌

[1] 이준형, 허의남. "클라우드 환경의 Thin-Client 모바일을 위한 동적 자원 분배 기술." 정보처리학회논문지 A 19.3 (2012): 161-168.

[2] J. Liu, et al. "Energy Efficient GPS Sensing with Cloud Offloading." Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems. ACM, 2012.

[3] Amazon Fire Phone. http://www.amazon.com/Fire_Phone_13MP-camera_32GB/dp/B00EOE0WKQ#firefly.

[4] Jan Kiszka. "Towards Linux as a Real-Time Hypervisor." Proceedings of the 11th Real-Time Linux Workshop. 2009.

[5] B. Zuo et al. "Performance Tuning Towards a KVM-Based Low Latency Virtualization System." 2nd International Conference on Information Engineering and Computer Science. 2010.

[6] S. Huh, et al. "Providing Fair Share Scheduling on Multicore Cloud Servers via Virtual Runtime-Based Task Migration Algorithm." Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on. IEEE, 2012.

[7] 허승주 외. "멀티코어 시스템에서 공정성 향상을 위한 Virtual Runtime 기반 로드 밸런싱 메커니즘." 한국정보과학회 학술발표논문집 38.2A (2011): 107-110.

[8] S. Xi, et al. "Rt-Xen: Towards Real-Time Hypervisor Scheduling in Xen." Embedded Software (EMSOFT), 2011 Proceedings of the International Conference on. IEEE, 2011.

[9] S. Xi, et al. "Real-Time Multi-Core Virtual Machine Scheduling in Xen." Embedded Software (EMSOFT), 2014 International Conference on. IEEE, 2014.

[10] Xen SEDF http://wiki.xen.org/wiki/Xen_Project_Schedulers