KVM 가상화 환경에서 워크로드 유형 기반 자원 격리 기법을 이용한 동적 자원 관리자

황나윤, 송충건, 이미현, 최희석, 유헌창
고려대학교 컴퓨터학과
e-mail:{nayoo38, security0730, dnqlalgus, hsrangken, yuhc}@korea.ac.kr

Dynamic Resource Manager Using Workload Type Based Resource Isolation Mechanism in KVM Virtualization Environment

NaYoon Hwang, ChungGeon Song, MiHyeon Lee,
HeeSeok Choi, HeonChang Yu
Dept of Computer Science and Engineering, Korea University

요 약

최근 중앙 집중화된 대규모 클라우드 시스템의 증가로 인해 가상화 환경에서 수행되는 성능 최적화 작업에 대한 많은 연구가 진행되고 있다. 그러나 기존 연구에서는 자원 분배의 공정성을 위해 가상머신 단위로 컴퓨팅 자원을 격리한 정책 내에서 이루어지고 있어 유연한 자원관리에 한계를 가지고 있다. 본 연구에서는 워크로드의 특징을 기반으로 과학적 연산을 수행하는 가상머신과 일반적인 서비스를 수행하는 가상머신을 분류하여 성능 최적화 작업을 수행하는 동적 자원 관리자를 제안한다. 실험을 통하여 제안하는 동적 자원 관리자가 KVM 기본 스케줄링에 비해 49%의 성능 향상을 보였다.

1. 서론

최근 클라우드 컴퓨팅이 다양한 산업에 보편화되고 있으며, 이용자의 증가로 클라우드 시스템의 규모가 점점 더 커지고 있다[1][2]. 이로 인해 클라우드 시스템 내에서 방대한 양의 자원을 효율적으로 관리하는 연구의 중요성이 높아지고 있다.

이러한 환경에서 자원 관리를 수행할 시 2계층 스케줄러에서 생기는 이질적 특징을 동적 자원 관리 정책에 반영해야 한다. 또한 가상화 계층의 스케줄러 정책만으로 공정한 자원 분배를 실현하기 위해서는 가상머신 단위의 자원 격리가 요구된다.

그러나 기존 연구에서는 가상머신 단위의 자원 격리를 완벽하기 유지하는 관점에서 자원 관리를 수행하기 때문에 빈번한 문맥교환이 발생하고 이로 인한 성능저하 문제가 발생한다. 또한 워크로드의 특징을 반영하지 못하여 잠재적 성능저하 문제를 가지고 있다.

본 연구에서는 이러한 한계점을 해결하기 위하여 KVM 기반 가상화 환경에서 워크로드의 특징을 기반으로 가상머신을 분류하고 성능 최적화 작업을 수행하는 동적 자원 관리자를 제안한다. 실험을 통하여 KVM의 기본 스케줄링 정책과 비교하여 49%의 성능 개선을 보였다.

본 논문의 구성은 2장에서 연구를 수행하게 된 배경과 동기에 대하여 설명하며, 3장에서는 성능 최적화 작업을 수행하는 동적 자원 관리자를 제안한다. 4장에서는 제안한 동적 자원 관리자의 성능을 검증하기 위한 실험과 그 결과를 보이고, 마지막으로 연구의 결과에 대한 정리와 향후 연구 방향을 5장 결론에서 설명한다.

2. 연구의 배경 및 동기

이 장에서는 리눅스 기반의 오픈소스 가상화 솔루션인 KVM에서 수행되는 기본 스케줄링 방식에 대하여 설명하며, 가상화된 시스템 환경에서 성능 최적화를 수행하는 기존 연구에 대하여 서술한다. 다음으로 본 연구의 동기가 되는 기존 연구의 한계점에 대하여 설명한다.

2.1 KVM 작업 스케줄러

CFS(Complete Fair Scheduler)는 리눅스 디폴트 스케줄러로서, 모든 프로세스에게 자원을 공평하게 제공하는 것을 목적으로 동작한다. CFS는 가상 수행 시간(virtual runtime)을 기반으로 레드

-블랙 트리를 생성하여 가상 수행 시간이 가장 작은 프로세스에게 CPU를 할당하는 알고리즘을 사용한다. 실제 자원을 할당하는 동작에서는 자원 선정의 우선순위를 결정하는 Nice 값을 참조하는데 -20에서 +19까지의 값을 가진다. 대기 시간(waiting time)이 긴 프로세스는 자동적으로 낮은 Nice 값을 부여하여 공평하게 주어진 시간(timeslice)을 기준으로 작업 스케줄링이 수행된다. 결과적으로 공정한 컴퓨팅 자원 분배 정책을 수행하기 위한 작업에서 빈번한 문맥교환(context switching)이 발생한다.

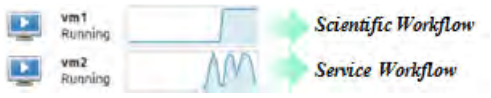
2.2 가상화 환경에서 성능 최적화

기존 가상화 기반의 성능 최적화 연구에서는 크게 가상화 솔루션 내부의 스케줄링을 수정하는 방식과 호스트 운영체제에서 데몬 형태의 관리자를 운영하는 방식, 그리고 가상화 솔루션의 부하 분산 알고리즘을 수정하는 방식으로 나누어진다.

스케줄링 수정방식의 대표적 연구로 Song Wu 외 4명이 제안한 vProve와 Luwei Cheng 외 2명이 제안한 vScale이 있다[3][4]. vProve에서는 NUMA(Non-Uniform Memory Access) 구조에서 메모리 접근 패턴을 고려한 새로운 스케줄링 기법을 제안하였으며, vScale에서는 가상화 환경에서 병렬 작업 동기화에서 발생하는 3가지 문제를 해결하는 스케줄러를 제안하였다. 다음으로 데몬 형태의 관리자를 운영하는 방식의 연구로 Xen 가상화 환경에서 기계학습을 통해 최적의 스케줄링 옵션을 찾는 Faruk Caglar, Shashank Shekhar, 그리고 Aniruddha Gokhale의 iTune이 있다[5]. 이 방식은 호스트 OS의 다양한 성능 관리 도구들과 상호작용을 수행할 시 모듈들 사이의 직접적인 통신이 가능하다. 마지막으로 가상화 솔루션의 부하 분산 알고리즘을 수정하는 방식의 대표적인 연구로 Kenichi Kourai와 Riku Nakata가 수행한 노드 친밀도 관리자가 있다[6]. 기존 Xen 가상화 솔루션의 부하분산 알고리즘에서 NUMA 노드 정보를 참조하여 컴퓨팅 자원을 선택하는 로직을 추가하였다.

2.3 기존 연구의 한계점과 연구동기

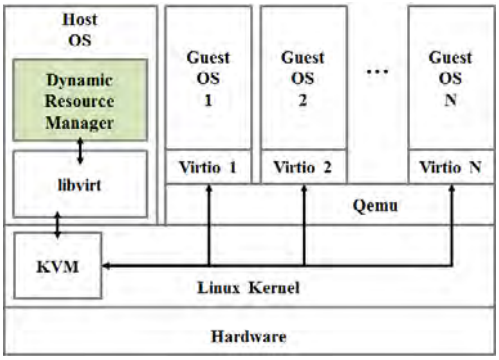
기존 연구에서는 가상머신 자원 분배의 공정성을 위하여 일정 시간동안 유휴자원을 번갈아가며 선점하도록 정책적으로 관리하는 방식을 사용하였다. 이러한 방식은 빈번한 컨텍스트 스위치를 야기한다. 또한 가상머신의 유형을 고려하지 않고 단일화된 목적으로 간주하고 있다. (그림 1)은 VM^{se}(Service Workflow Virtual Machine)과 VM^{sc}(Scientific Workflow Virtual Machine)에 나타나는 컴퓨팅 자원의 이용 패턴을 나타낸다.



(그림 1) VM에서 수행되는 워크로드 유형

3. 제안하는 동적 자원 관리자

3장에서는 본 연구에서 제안하는 동적 자원 관리자를 설명한다. 이는 주기적으로 가상머신의 vCPU 이용 패턴을 분석하고 VM^{se}과 VM^{sc}으로 분류하여 각각의 유형에 최적화된 동적 자원 관리를 수행한다. (그림 2)는 제안하는 동적 자원 관리자의 동작 구조를 나타내고 있다.



(그림 2) 동적 자원 관리자의 동작 구조

동적 자원 관리자는 Host OS 환경에서 리눅스 데몬(Daemon) 형태로 개발되어 백그라운드로 운영된다. 이는 관리자가 설정된 동작 주기를 기준으로 vCPU의 상태 모니터링을 수행하는 모듈과 가상머신을 분류하고 동적 최적화를 수행하는 모듈로 구성된다. 알고리즘에서 다루는 함수는 <Table 1>에서 나타내며, 기호에 대한 설명은 <Table 2>에서 보여 준다.

<표 1> 함수 설명

함수	설명
$allocate\_SLA(VM_i^{se})$	가상머신에 지정된 SLA를 보장하기 위하여 요구되는 최소한의 컴퓨팅 자원을 할당
$allocate\_PR()$	현재 시스템 전체 유휴 CPU자원을 N_{sc}를 기준으로 나누어 $LIST_{sc}$에 관리되는 모든 VM^{sc}에게 공평하게 분배

<표 2> 기호 설명

기호	설명
$VM_i^{se}(1 \le i \le N_{se})$	i번째 VM^{se}
$VM_i^{sc}(1 \le i \le N_{sc})$	i번째 VM^{sc}
$LIST_{se}$	VM^{se}들을 관리하는 리스트
$LIST_{sc}$	VM^{sc}들을 관리하는 리스트
T_{check}	제안하는 관리자 동작하는 주기
α	CPU 사용량 임계치
β	α를 이탈한 횟수의 임계치
N_{se}	VM^{se}의 개수
N_{sc}	VM^{sc}의 개수

알고리즘 1은 가상머신의 vCPU의 상태 모니터링을 수행하는 모듈의 동작방식을 구체적으로 나타내고 있다. 3~9에서는 VM^{se}에 대한 자원 모니터링 동작방식을 나타내고 있으며 vCPU 사용량이 임계치 α이상으로 증가 했는지 여부를 검사한다. 그리고 11~17에서는 VM^{sc}에 대한 자원 모니터링 동작방식을 나타내고 있으며, vCPU 사용량이 임계치 α 미만으로 감소했는지 여부를 검사한다.

알고리즘 1 Virtual Computing Resource Monitoring Algorithm

```
1:  /* T_check를 주기로 함수가 실행된다. */
2:  /* VM^se 자원 모니터링 */
3:  for i=1 to N_se do
4:     if ( VM_i^se.cpuUsage ≥ α) then
5:        increase VM_i^se.count
6:     else
7:        decrease VM_i^se.count
8:     end if
9:  end for
10: /* VM^sc 자원 모니터링 */
11: for i=1 to N_sc do
12:    if ( VM_i^sc.cpuUsage < α) then
13:       increase VM_i^sc.count
14:    else
15:       decrease VM_i^sc.count
16:    end if
17: end for
```

알고리즘 2는 모니터링 정보를 활용하여 가상머신을 분류하고 동적 최적화를 수행하는 동작을 구체적으로 나타내고 있다. 3~10에서는 VM^{se}에 대한 동적 자원관리 동작방식을 나타내고 있다. 만약 가상머신의 CPU 자원 사용 패턴이 VM^{sc}과 같은 양상을 보일 경우 해당 가상머신을 $LIST_{sc}$로 이동시키고 전체 유휴자원을 조정한다. 12~19에서는 VM^{sc}에 대한 동적 자원관리 동작방식을 나타내고 있다. 만약 가상머신의 CPU 자원 사용 패턴이 VM^{se}와 같은 양상을 보일 경우 가상머신을 $LIST_{se}$로 이동시키고 SLA를 보장하는 최소한의 컴퓨팅 자원을 계산하여 할당한다.

알고리즘 2 Dynamic Resource Management Algorithm

```
1:  /* T_check를 주기로 함수가 실행된다. */
2:  /* VM^se 동적 자원관리 */
3:  for i=1 to N_se do
4:     if ( VM_i^se.count > β) then
5:        VM_i^se.count ← 0
6:        delete_VM(LIST_se, VM_i^se)
7:        add_VM(LIST_sc, VM_i^se)
8:        allocate_PR()
9:     end if
10: end for
11: /* VM^sc 동적 자원관리 */
12: for i=1 to N_sc do
13:    if ( VM_i^sc.count > β) then
14:       VM_i^sc.count ← 0
15:       delete_VM(LIST_sc, VM_i^sc)
16:       allocate_SLA(VM_i^sc)
17:       add_VM(LIST_se, VM_i^sc)
18:    end if
19: end for
```

4. 성능 평가

이 장에서는 제안하는 동적 성능 관리자의 성능을 검증하기 위한 실험을 수행한다. 본 실험에서는 다양한 워크로드가 발생하는 가상머신을 다음과 같이 3가지로 분류하였다.

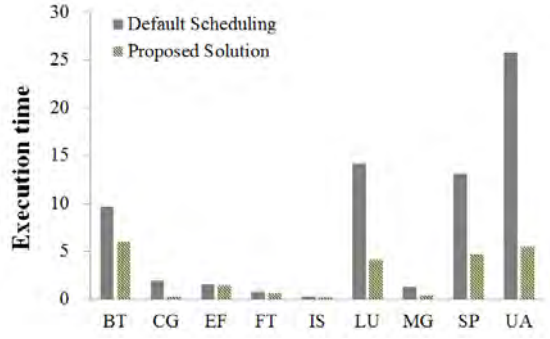
- **PC 목적의 가상머신** - 상용 운영체제가 구성되어 있으며 한정된 시간에 가벼운 워크로드가 발생한다.
- **웹서비스를 수행하는 가상머신** - 불특정 다수의 클라이언트에게서 발생하는 웹서비스 요청으로 인하여 다양한 크기의 워크로드가 무작위로 발생한다.
- **과학적 실험에 활용되는 가상머신** - 특정 연구 분야에서 하나의 가설을 증명하기 위하여 대규모 컴퓨팅 연산을 오랜 시간 수행한다.

4.1 실험 환경

실험은 Ubuntu LTS 14.04.2 버전의 운영체제가 설치된 IBM System x3500 M4 서버에서 수행되었다. 2개의 CPU 칩으로 구성되어 있으며, 각각 8개의 코어를 가지고 있다. 성능 측정을 위한 벤치마크 도구는 SMP기반 시스템에서 병렬처리 성

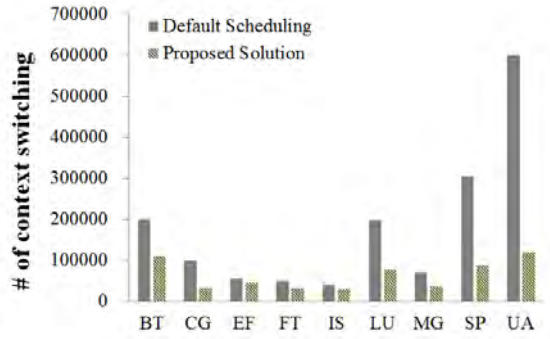
능을 측정하기 위한 NPB (NAS Parallel Bench mark)의 9종류의 어플리케이션을 활용하였다. 또한 컨테스트 스위치 측정을 위해 perf 어플리케이션을 활용하였으며 웹서비스를 운영하는 가상머신에는 Apache 환경을 구성하였다. 마지막으로 KVM 가상화 솔루션은 2.0.0 버전으로 구축하였다. 실험에서는 앞서 설명한 3가지 가상머신을 각각 하나씩 운영하였으며, 과학적 실험에 활용되는 가상머신 환경에서 NPB 어플리케이션의 실행시간을 통하여 성능을 측정하였다.

4.2 실험 결과



(그림 3) 실행 시간

(그림 3)은 NPB의 9가지 벤치마크 실행시간을 나타내고 있다. 실험결과 UA가 78%의 가장 높은 성능 향상을 보였고 EF가 8%의 가장 적은 성능 향상을 보였다. 전체적으로 49%의 성능향상을 보였다. 이러한 성능향상의 원인을 분석하기 위하여 각 벤치마크 실행에서 발생한 문맥교환 발생 수를 측정하였으며 (그림 4)에서 나타내고 있다.



(그림 4) 문맥교환 발생 수

실험결과 UA가 문맥교환이 80% 감소한 가장 큰 차이를 보였으며, EF가 16%로 가정 적은 감소 비율을 보였다. 이러한 실험을 통하여 제안한 동적 자원 관리자가 실행 중 발생하는 문맥교환의 수를 감소시킬 수 있음을 보여주었으며, 결과적으로 실행시간 측면의 성능향상이 이루어졌음을 검증하였다.

5. 결론

본 연구에서 제안하는 동적 자원 관리자는 과학적 연산을 수행하는 가상머신과 일반 서비스 연산을 수행하는 가상머신을 구분하는 여러 자원에 대한 임계값을 관리자가 지정하는 방식으로 구성하였다. 또한 가상머신들의 상태정보를 모니터링하고 워크로드에 따라 분류하며 옵션을 최적화 하는 주기를 특정 시스템에서의 실험을 통하여 결정하는 방식을 사용하였다. 향후 이러한 임계값과 기법을 적용하는 주기를 설정하는 방식에 기계학습 이론을 적용하여 관리자가 설정하는 방식이 아닌 Training 단계에서 최적의 값을 찾는 방식으로 개선하고자 한다.

참고문헌

[1] AWS EC2, aws.amazon.com/ec2

[2] Microsoft Azure, azure.microsoft.com

[3] Luwei Cheng, Jia Rao, Francis C.M. Lau, “vScale: Automatic and Efficient Processor Scaling for SMP Virtual Machines”, EuroSys, 2016.

[4] Song Wu, Huahua Sun, Like Zhou, Qingtian Gan, Hai Jin, “vProbe: Scheduling Virtual Machines on NUMA Systems”, IEEE International Conference on Cluster Computing, pp. 70 - 79, 2016.

[5] Faruk Caglar, Shashank Shekhar, Aniruddha Gokhale, “iTune: Engineering the Performance of Xen Hypervisor via Autonomous and Dynamic Scheduler Reconfiguration”, IEEE Transactions on Services Computing, Vol. 13, No. 9, 2015.

[6] Kenichi Kourai, Riku Nakata, “Analysis of the Impact of CPU Virtualization on Parallel Application in Xen”, IEEE Trustcom/BigDataSE/ISPA, 2015.