Journal of The Korea Society of Computer and Information
Vol. 18, No. 7, July 2013
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2013.18.7.011

가상화 환경에서 세밀한 자원 활용률 적용을 위한 스케일 기법

이 돈 혁* 오 상 윤*

Fine Grained Resource Scaling Approach for Virtualized Environment

Donhyuck Lee*, Sangyoon Oh*

요 약

최근 데이터 센터와 같은 대규모 컴퓨터 자원을 운용함에 있어 가상화 기술을 적용하여 컴퓨팅 자원을 동적으로 사용할 수 있게 됨에 따라 탄력적인 프로비저닝이 가능하게 되었다. 현재 운영되고 있는 클라우드 시스템에서는 이러한 동적 프로비저닝을 위해 스케일 업 또는 스케일 아웃형태의 스케일링을 지원하고 있으며, 이 방식은 사용자 요구조건의 만족을 주목적으로 하며 방대한 컴퓨팅 자원을 기반으로 하는 공공 클라우드 시스템 운용에 부합한다. 그러나 제한된 컴퓨팅 자원으로 하는 사설 클라우드의 운영을 위해서는 보다 높은 운영 효율을 위해 세밀한 자원 활용을 위한 스케일링 기법이 요구된다. 본 논문에서는 사설 클라우드에서 높은 자원 활용률을 얻기 위해 가상화 기술인 동적자원할당과 Live Migration 기법을 이용하여 스케일업과 스케일아웃을 복합적으로 사용한 서버 스케일링 아키텍처를 설계하고 이에 따른 알고리즘을 설계하였다. 이를 통해 세밀하게 단계별로 스케일링을 진행하여 서버 관리와 비용의 부담을 줄이고 서버 자원의 이용률을 최적화함으로써 서비스가 안정적으로 유지되도록 할 수 있다. 성능평가를 통해 제안한 구조와 알고리즘이 접속자 수에 따른 스케일 아웃을 수행하는 방식에 비해 높은 자원활용률을 보이는 것을 확인하였다.

▸ Keywords : 가상화, 자원 스케일링, 라이브 마이그레이션, 클라우드 컴퓨팅, 자원 프로비저닝

Abstract

Recently operating a large scale computing resource like a data center becomes easier because of the virtualization technology that virtualize servers and enable flexible resource provision. The most of public cloud services provides automatic scaling in the form of scale-in or scale-out and these scaling approaches works well to satisfy the service level agreement (SLA) of users.

•제1저자 : 이돈혁 •교신저자 : 오상윤
•투고일 : 2012. 12. 18, 심사일 : 2013. 4. 10, 게재확정일 : 2013. 7. 14.
* 아주대학교 컴퓨터공학과(Dept. of Computer Engineering, Ajou University)
※ 본 연구는 미래창조과학부 및 정보통신산업진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음 (NIPA-2013-(H0301-13-2003))

However, a novel scaling approach is required to operate private clouds that has smaller amount of computing resources than vast resources of public clouds. In this paper, we propose a hybrid server scaling architecture and related algorithms using both scale-in and scale-out to achieve higher resource utilization rate for private clouds. We uses dynamic resource allocation and live migration to run our proposed algorithm. Our propose system aims to provide a fine-grain resource scaling by steps. Thus private cloud systems are able to keep stable service and to reduce server management cost by optimizing server utilization. The experiment results show that our proposed approach performs better in resource utilization than the scale-out approach based on the number of users.

▸ Keywords : virtualization, resource scaling, live migration, cloud computing, resource provisioning

I. 서 론

웹(Web) 기술의 발달과 더불어 웹2.0[1]의 출현으로 웹은 단순히 서비스 제공자가 제공하는 정보를 일방적으로 사용자가 이용만 하는 형태에서 서비스 제공자가 제공하는 플랫폼과 정보를 이용하여 사용자가 스스로 참여와 정보 공유를 통해 또 다른 정보를 창출할 수 있게 되었다. 또한 RIA(Rich Internet Application)플랫폼[2]의 등장으로 데스크톱 어플리케이션을 대체할 수 있는 웹 어플리케이션의 개발까지 가능해지면서 웹을 플랫폼으로 하고 사용자 참여에 기반 하는 새로운 서비스를 제공하는 기업이 기하급수적으로 늘어나게 되어 웹 사용자의 수는 폭발적으로 증가하게 되었고 이에 따라 서버시스템이 받는 부하가 늘어나 서버시스템의 규모는 커지고 서버가 처리해야 하는 일이 더욱 복잡해지게 되었다.

이 같이 많은 요청을 처리해야하고, 따라서 이 요청을 처리하기 위해 복잡해진 서버 시스템 환경에 대응하기 위한 방편으로 웹 서비스를 제공하려는 기업에서는 적절한 사양과 수량의 서버 도입을 위해 서비스를 운영하기 전에 서버 시스템의 초기 용량 기획(capacity planning)[3]을 해야 한다. 그러나 이와 같은 용량기획은 구체적인 계산 및 예측이 어렵고 서버 자원이 부족하면 정상적인 서비스 제공을 할 수 없게 된다. 이와 같은 이유와 함께 현재의 웹 서비스 환경에서 발생할 수 있는 Workload Fluctuation 에 대한 대처를 위해서 각 기업에서는 초기에 불필요한 많은 서버 자원을 확보하여 서비스를 제공하게 된다. 그러나 이 방식은 결국 서버 자원들이 낮은 활용률을 갖게 하며 자원 낭비 및 기업의 불필요한 운영유지비를 지출하게 되는 결과를 가져온다.

이와 같은 문제를 해결하기 위해 최근 가상화 기술이 각광받고 있다. 가상화 기술은 컴퓨터 자원을 추상화 하여 사용자에게 VM(Virtual Machine)의 형태로 논리적 자원을 제공하는 기술로써, 서버의 자원 활용률을 높이며, 기업의 운영유지비를 낮추는 등의 많은 장점을 갖는 기술이다. 또한 자원의 동적 할당 및 Live Migration을 통해 서비스의 중단 없이 자원을 관리할 수 있기 때문에 자원을 탄력적으로 사용할 수 있게 되었다. 하지만 상대적으로 넉넉한 컴퓨팅 자원을 바탕으로 초기에 제시된 사용자 요구조건을 만족 시키는 것을 주요 목표로 하는 대형의 공공 클라우드 제공자에 비해 제한된 컴퓨팅 자원을 바탕으로 운용되는 사설 클라우드에서는 큰 손실을 야기하는 용량 기획의 실패를 방지하고, 높은 자원 활용률을 얻기 위해서는 좀 더 세밀하고 효율적인 새로운 스케일링 기법이 필요하다.

스케일링 자동화를 위해서는 동시접속자 수를 기반으로 스케일 아웃을 진행하는 방식 [11]및 예측기법을 이용하여 스케일 업을 적용하는 방법 [12] 등이 있으나 접속자 수를 컴퓨팅 자원의 상황에 따라 탄력적으로 적용하는 것이 어려운 점과 예측기법으로는 예상보다 큰 Workload Fluctuation에 반응하기 어려운 문제 들이 있어 높은 자원 활용률을 얻기 힘들다. 따라서 본 연구에서는 가상화된 환경에서 서버 운용유지 비용을 절감하고 동적 자원 할당과 Live Migration을 통해 효과적인 서버 자원의 활용이 가능한 스케일링 시스템을 제안한다. 스케일 업과 스케일 아웃을 복합적으로 사용하는 Hybrid 방식을 통해 서버 자원을 세밀하게 증감시켜 스케일링 시 자원이용률이 크게 떨어지는 것을 막는 것을 목표로 한다. 제안하는 시스템은 다양한 Hypervisor (KVM [18], Xen [17])등을 지원하며 Libvirt API를 통해 이를 제어한다. 그리고 주기적으로 모니터링을 통해 획득된 자원 활용률

정보를 기반으로 스케일링을 하도록 하여 VM의 성능을 보장하고 효과적인 자원의 이용을 가능하게 한다.

이후의 논문 구성은 다음과 같다. 2장에서는 본 논문과 관련된 자원 스케일링 방식과 이에 대한 기존연구들을 살펴본다. 3장에서는 본 논문에서 제안하는 스케일링 시스템의 디자인의 개요, 모델링 및 제안하는 스케일링 알고리즘을 설명한다. 4장에서는 구현된 제안 시스템을 평가하기 위해 구축한 시뮬레이션 환경과 가상화 환경의 자원 동적 스케일링 시스템에 대한 기능 검증, 그리고 기존 방식인 scaling-out방식과 본 제안 방식의 비교실험을 통한 효용성 검증 결과를 제시하며 분석을 수행하며, 5장에서는 본 연구의 결과를 요약하여 결론을 맺고 제안 시스템의 향후 연구 과제를 제시한다.

II. 관련 연구

기본적인 스케일링 방식은 스케일 업과 스케일 아웃 두 가지로 나누어진다. 스케일 업 방식은 수직적 스케일링이라고도 하며 서버의 컴퓨팅 자원을 추가해 성능을 향상시키는 방식이다. 즉 서버의 수는 그대로 유지하면서 CPU나 메모리, 저장소와 같은 하드웨어 컴퓨팅 자원을 추가시켜 용량을 늘리는 것으로 자원 배치문제를 단순화하고, 소프트웨어 간 충돌을 고려할 필요가 없어 비용대비 효과적인 전략이다. 하지만 수직적 스케일링은 서버가 수용할 수 있는 최대 용량에 도달하면 더 이상 추가를 할 수 없다는 단점이 있다.

가상화 환경에서의 스케일 업을 통한 스케일링은 보다 유연하게 처리할 수 있다. 트래픽의 변화에 따라 자원을 다시 조정할 필요가 생기면 기존의 일반적인 서버 스케일링에서처럼 서버 운영을 중지하고 물리적으로 자원을 끼워 넣을 필요 없이 동적으로 자원의 할당이 가능하다[4]. 또한 VM Live Migration[5,6] 기술을 이용해 자원 이용률이 높은 VM을 자원 적 여유 공간이 넓은 다른 물리 서버로 옮길 수 있다.

가상화 환경에서 스케일 업을 수행하는 방식은 그림 1과 같이 두 가지로 구분 할 수 있다. 첫 번째 방식은 그림 1-a와 같이 PM1의 여유 공간이 넉넉한 상황에서 VM1이 원하는 만큼 자원을 동적으로 늘려주는 것이다. 그리고 두 번째 방식은 그림1-b와 같이 PM1이 더 이상 할당할 수 있는 자원의 여유 공간이 없는 상황에서 VM3의 자원을 늘려주기 위해 여유자원을 가진 PM2로 VM3을 Live Migration을 통해 이주시키고 VM3의 자원을 늘려주는 것이다.

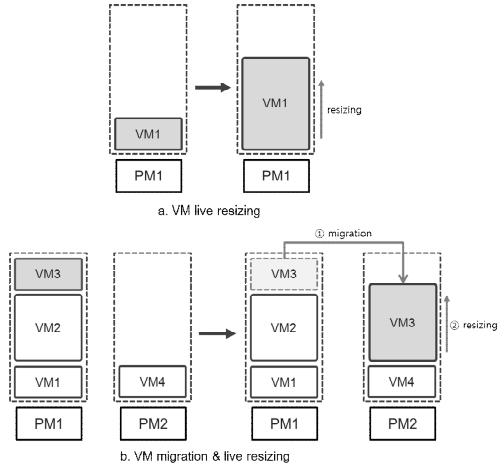

그림 1. 가상화 환경에서의 스케일 업 방식
Fig. 1. Scale Up of Virtualization Environment

스케일 아웃은 수평적 스케일링이라고도 하며 서버를 여러 대로 구성하여 요청이 들어올 시 로드밸런서[7,19]를 통해 부하를 분산시키는 방식이다. 가상화 환경에서는 VM에 로드밸런서 소프트웨어를 설치하여 운영하거나 하드웨어 로드밸런서에 VM을 연결하게 된다. 기존 메인 프레임 방식에서 스케일 아웃은 새로운 서버를 구입하고 OS를 설치한 후 동일한 application을 설치하는 등 반복적인 일을 수행 한 후 로드밸런서에 연결해야 하지만 가상화 환경에서는 virtual appliance[8] 기술을 이용하여 이를 간편화 할 수 있다. Virtual appliance는 한번 구성해 놓은 서버 시스템을 복사해두었다가 필요 시 바로 실행하여 서버를 운영할 수 있다. 이는 시스템을 설치 및 구성하는 시간을 줄여주고 빠르게 시스템을 확장할 수 있도록 한다. 따라서 가상화 환경에서의 기본적인 스케일 아웃은 동일한 작업을 수행하는 VM들을 클러스터로 묶고 로드밸런서에 연결하여 클라이언트로 들어오는 요청을 분산 처리하게 하고 스케일링이 필요할 때 Virtual Appliance 기술을 이용하여 미리 세팅되어진 서버를 실행하여 확장을 하게 된다.

스케일링 자동화 시스템을 서비스 하는 곳은 대표적으로 구글 앱 엔진과 아마존 EC2가 있다. 구글 앱 엔진은 PaaS (Platform as a service)로서 서비스를 제공하며 스케일링 자동화와 로드밸런싱을 지원하고 있다 [9]. 구글 앱 엔진은 구글 앱 엔진 내에서 작동 할 수 있도록 한 표준화된 플랫폼 상에서 설정한 자원 이상의 트래픽이 발생하면 스케일 아웃을 통해 자동으로 동일한 VM을 생성해 스케일링을 하게 된다. 아마존EC2에서는 VM을 2대 이상 구입하고 로드밸런서서비

스를 신청한 후 설정한 자원 이상 트래픽이 발생하면 구입해 둔 VM을 작동시켜 스케일링하게 된다 [10].

이처럼 실제 서비스되고 있는 스케일링 자동화 시스템들은 다른 특징을 가지고 있다. 하지만 스케일링 자동화 동작방식은 스케일 아웃 형태로 제공되는 고정된 타입의 VM을 증가시키고 있으며 이는 자원 낭비 및 비용 부담이 될 수 있다.

실제 서비스되고 있는 환경 외에 좀 더 서버를 효율적으로 운용하기 위해 관리를 최소화 하고 트래픽에 따라 자원을 확장, 축소해주는 스케일링 자동화에 관한 연구[11,12]들이 진행되었다. 스케일링 대상을 웹 서버로 한정하고 동시접속자의 수를 스케일링 판단 여부로 스케일링을 하는 방식[11]은 스케일 아웃을 통해 대상 서버를 스케일링 자동화를 지원하고 있다. 하지만 동시 접속자 수를 40,000명으로 한정하고 있으며 이는 서버 사양에 따라 허용할 수 있는 동시접속자의 수가 다르기 때문에 자원 적 낭비가 이루어질 수 있다. SLA(Service Level Agreement)를 위반하지 않기 위해 예측기법을 이용하여 사전에 스케일링을 수행하는 연구[12]에서는 스케일 업만을 사용하여 스케일링 능력이 제한된다. 또한 비용관점에서 스케일링 방식을 제안한 연구[16]에서는 Deadline이전에 모든 요청된 일을 처리하는 관점에서 스케일링 문제에 접근하고 있다. 이 제안에서는 비용모델링을 제시하여 성능과 비용의 정량화된 분석이 가능하도록 하였으나 본 제안연구의 주요문제인 fine-grained 자원 활용은 고려되고 있지 않다.

III. 세밀한 자원 활용 스케일링 기법

1. 디자인 개요

제안 시스템은 VM에서 동작되는 어플리케이션이 최적화된 자원 이용률을 보이는 것을 목표로 시스템의 자원 이용률을 모니터링[20]하고 이를 분석 해 스케일링이 요구될 때 스케일링을 진행한다. 그림 2에서 나타난 것과 같이 제안 시스템은 2가지 프로세스 즉, 모니터 모듈, 스케일링 모듈로 구성되어 동작한다.

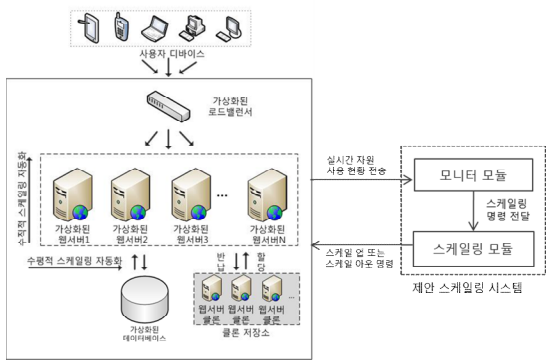

그림 2. 제안 시스템 구조
Fig. 2. Proposed System Architecture

모니터 모듈은 자원관리에 있어서 자원 배분 결정에 필요한 컴퓨팅 자원 상태 정보를 제공하는 기본적이면서 필수적인 모듈이다. 제안 시스템에서 모니터 모듈은 시스템의 자원 이용을 모니터링 하여 저장하는 일반적인 모니터의 역할과 저장된 모니터링 내역을 필요에 따라 다른 모듈에게 제공해주는 역할이 있다. 모니터링 대상은 PM의 물리적 자원 정보와 PM이 가진 각 VM에 할당된 자원 정보 및 주기적인 시점 t의 자원 사용량이다. 시점 t의 자원 사용량은 할당된 자원 이상의 자원 이용률을 보이는지 판단하여 스케일링의 지표로 삼게 된다. 스케일링을 판단하는 요소는 다음과 같으며 어플리케이션 관리자는 해당 요소들을 미리 설정하여 설정된 요소들을 기반으로 스케일링이 진행되도록 하여야 한다.

- CPU, 메모리 이용률의 상한점과 하한점
- 스케일 업 시 추가, 또는 감소될 자원의 한계
- 스케일 업 시 할당 할 수 있는 최대 자원의 한계
- 스케일 업 시 할당 할 수 있는 최소 자원의 한계
- 스케일 아웃 시 생성할 VM의 기본 자원 크기

제안하는 시스템은 CPU와 메모리 자원의 스케일링을 지원하기 때문에 모니터링 모듈은 각 VM의 CPU와 메모리의 이용률을 수집하게 된다. 컴퓨팅 자원의 이용률은 시스템 내의 정보를 볼 수 있는 /proc 명령어를 통해 데이터를 구할 수 있다[14]. 모니터링 모듈은 각 PM에 각VM들의 자원 이용률을 수집할 수 있는 경량의 데몬을 설치하여 이를 통해 데이터를 전달 받게 된다.

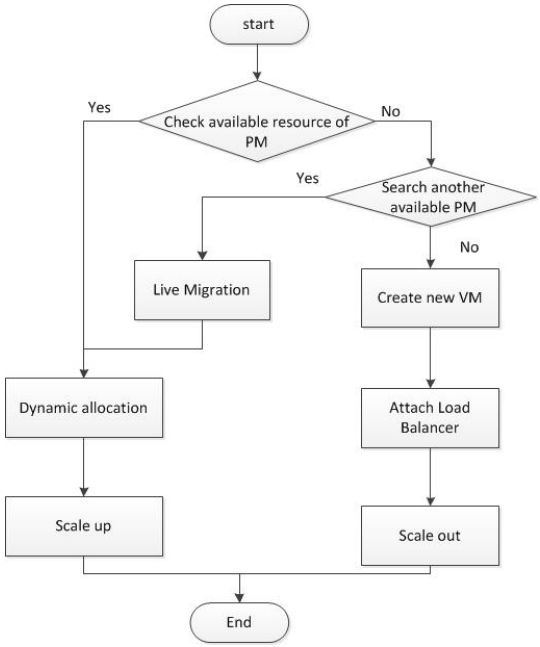

그림 3. 제안 시스템 스케일링 알고리즘 순서도
Fig. 3. Flow Chart of Proposed System Scaling Algorithm

스케일링 모듈에서는 모니터 모듈을 통해 스케일링을 해야 하는 상황을 전달받으면 스케일링을 진행하는 알고리즘을 수행한다. 기본적으로 제안하는 스케일링 자동화 알고리즘은 스케일 업을 우선으로 한 후 스케일 업을 할 공간이 없다면 스케일 아웃을 하게 된다. 이는 스케일 업을 통해 자원의 세밀한 추가가 가능해 자원의 이용 효율을 높이고 운용되는 VM의 수를 최소화하여 공간을 효율적으로 사용할 수 있게 한다. 이 알고리즘은 모니터 모듈에서 수집한 각 VM들의 자원 이용률 데이터 값이 임계치를 넘는지 여부로 판단하여 스케일링을 시작하라는 명령을 받으면 수행하게 된다. 많은 컴퓨팅 자원을 점진적으로 소모하는 VM이라면 그 VM에 할당해야 할 자원의 크기는 시간이 지남에 따라 점점 커지게 된다. 하지만 PM의 자원은 한계가 있기 때문에 더 이상 스케일 업을 할 수 없는 상황이라면 스케일 아웃을 고려하여 효율적으로 자원을 사용할 수 있다.

2. 제안 시스템 모델링

스케일링 자동화 시스템은 가상화 기술을 통해 PM에서 생성된 VM들 위에서 동작되는 Application들의 컴퓨팅 자원 사용량을 근간으로 가상화된 컴퓨팅 자원을 확장 또는 축소 할당하게 된다. 본 절에서는 중요변수와 PM과 VM의 관계를 정립하여 시스템 디자인의 근거로 삼기 위한 모델링 결과를 제시한다.

각 PM의 자원은 다음과 같이 정의한다. 각 PM_i의 자원 사용량은 $R_i = \{R_{i,c}, R_{i,m}\}$로 표현된다. $R_{i,c}$는 각PM의 CPU 자원 사용량을 나타내며 $R_{i,m}$는 각PM의 메모리 자원 사용량을 의미한다. 그리고 각 VM_j의 자원 사용량은 $V_j = \{V_{j,c}, V_{j,m}\}$로 표현된다. $V_{j,c}$는 단일 VM의 VCPU 자원 사용량을 나타내며 $V_{j,m}$는 단일 VM의 메모리 사용량을 나타낸다. 클러스터 안에는 m개의 PM노드가 있고 k개의 VM이 있다고 가정한다. PM 클러스터의 자원 이용률은 $R = (R_1, R_2, \cdots, R_i, \cdots, R_m)$으로 표현하며 마찬가지로 VM 클러스터의 자원이용률은 $V = (V_1, V_2, \cdots, V_j, \cdots, V_k)$로 표현된다.

3. 제안 시스템 알고리즘

제안 스케일링 알고리즘은 제안 시스템의 목적인 컴퓨팅 자원을 비효율적으로 사용하는 VM을 찾아 동적 자원 재 할당을 통해 자원 이용률의 균형을 이루도록 하여 시스템 안정화를 보장한다. 현재 가상화 기술에서 VM을 재구동하지 않고 동적 할당이 가능한 자원은 CPU와 Memory이다. 제안 스케일링 알고리즘에서 쓰이는 데이터는 각 VM들의 모니터링을 통해 수집한 CPU와 Memory자원 이용률을 기반으로 한다. 따라서 본 논문에서는 이 두 가지 자원을 대상으로 스케일링 자동화 알고리즘을 제시하였다.

스케일링 모듈에서 스케일링을 하는 방식은 그림 3과 같이 스케일 업 방식과 스케일 아웃방식 두 가지로 나누어진다. 모니터링 모듈을 통해 스케일링을 해야 한다는 명령을 통지 받으면 스케일링 모듈은 그림 3과 같은 순서에 따라 스케일링을 진행하게 된다. 스케일링을 진행해야 하는 VM이 속해 있는 PM의 자원 적 여유가 있다면 동적 할당을 통해 스케일 업 형태의 스케일링을 한다. 해당 PM의 자원 적 여유가 없다면 자원 적 여유 공간이 있는 다른 PM을 찾는다. 이때 PM들의 자원이 가장 많이 남는 PM을 선택 후 스케일링 대상이 되는 VM을 선택된 PM로 Live Migration을 진행하여 스케일 업 형태의 스케일링을 하게 된다.

만약 스케일 업을 할 여유 공간이 있는 다른 PM을 찾지 못했다면 새로운 VM을 생성할 PM을 찾고 VM 복사본을 찾아 로드밸런서에 추가하여 해당 VM을 동작시켜 스케일 아웃을 진행하게 된다. 이때도 마찬가지로 PM의 자원 공간이 가장 많은 PM을 선택한다. 스케일 업을 먼저 시행함으로써 시스템의 중지 없이 적은 자원의 세밀한 추가가 가능하고 스케

일링 속도가 스케일링 아웃에 비해 빠른 스케일링을 하게 된다. 때문에 제안하는 스케일링 방식은 스케일 업을 우선시하여 스케일링하도록 한다. 또한 Live Migration기법은 물리 머신의 수를 최소화하여 에너지 절감을 꾀할 수 있도록 [13]을 적용하여 동작하도록 한다.

그림 4는 제안하는 스케일링 알고리즘을 Pseudo Code로 나타낸 그림이다. 앞서 설명한 스케일링 기법에 따라 스케일 업과 아웃을 이용하여 진행하게 된다.

Algorithm FinegrainedScaling(R, V)
Input : a set of resource usage state of the PM cluster(R), a set of resource usage state of the VMs(V)
Output : scaling of V

```
1:  for i=1 to (K-1)      // Step 1
2:    if ( V_i Lower Threshold ) then
3:      addScaleDownList( V_i)
4:    if ( V_i Upper Threshold ) then
5:      addScaleUpOrOutList( V_i)
6:  breakfor
7:  for i=1 to ScaleDownList()      // Step 2
8:    if ( numberOfSameApplication( V_i) > 1) then
9:      removeFromLoadBalancer( V_i)
10:     shutdown( V_i)
11:   else if ( V_i Lower Threshold) then
12:     reduceResource( V_i)
13: breakfor
14: for i=1 to ScaleUpOrOutList()      //Step 3
15:   if ( V_i ≥ Upper Threshold and R_i ≥ V_i) then
16:     incrementResource()
17:   else if ( V_i ≥ Upper Threshold and R_i < V_i )
18: then
19:     addToLoadBalancer(startNewVM())
20: breakfor
```

그림 4. 제안 스케일링 알고리즘 Pseudo Code
Fig. 4. Pseudo Code of Proposed Scaling Algorithm

4. 기존 방식과 제안 시스템의 비교

본 연구에서는 모니터링 된 정보를 통해 스케일 업과 스케일 아웃을 복합적으로 사용하는 Hybrid 스케일링 방식을 통해 스케일링 영역을 넓히고 SLA 위반을 방지하도록 하는 시스템 및 알고리즘을 제안 한다. 여기서 스케일링 기법은 수학적 알고리즘을 통해 해결하기 보다는 constraint programming을 통한 제약 조건을 준 후 제약 조건을 만족했을 시 스케일링을 하도록 하는 기법을 사용한다.

표1은 참고문헌[11,12]에서 제시된 방식들과 본 논문의 제안 방식과의 비교를 통해 장·단점을 분석하여 정리한 표이다. 분석 결과에서 볼 수 있듯이 Hybrid 방식을 통해 두 시스템의 장점만을 취할 수 있음을 확인할 수 있다.

표 1. 제안 방식들의 각 항목별 장단점 비교
Table 1. Comparison between various scaling approaches

	접속자수 기반 Scale-out [11]	Scale-up (CloudScale) [12]	본 논문의 Hybrid Scaling 방식
Prediction	Low	High	Low
Resource Consuming	Low	High	Low
Scaling Overhead	High	Low	Low & High
Scalability	High	Low	High
Fine Grained	Low	High	Mid

•Prediction & Resource Consuming

CloudScale[12]의 경우에는 스케일링을 위해 앞으로의 사용량을 예측하여 결정하므로 다른 방법에 비해 효과적일 수 있다. 하지만 Prediction을 위해 Computing resource를 소비하기 때문에 이를 고려할 필요가 있다.

•Scaling Overhead

Scalable Web App Architecture[11]의 경우에는 Scale out만을 하기 때문에 Scale up 보다는 스케일링의 오버헤드가 크다고 할 수 있다. 제안 시스템에서는 Scale up과 Scale out을 번갈아 진행하기 때문에 작을 수도 있고 클 수도 있다.

•Scalability

CloudScale의 경우에는 Scale up만 하기 때문에 VM의 자원 사용량이 PM의 허용범위를 넘어설 수 없다. 따라서 PM의 능력을 넘어서는 사용자 request가 들어왔을 때에는 대처할 수 없다. 따라서 Scalability가 낮다고 할 수 있다. Scalable Web App Architecture 시스템에서는 Scale out을 하기 때문에 사용자 request에 따라 유연한 대처가 가능할 것이다.

•Fine Grained(세밀함)

CloudScale의 경우에는 앞으로의 사용량을 예측하여 스케일링하기 때문에 다른 시스템에 비해 세밀함이 더 뛰어날 수 있다. 제안 시스템은 Scale up을 통해 Scale out보다는 세밀한 스케일링을 제공할 수 있다. Scalable Web App Architecture의 경우에는 Scale out을 하기 때 때문에 스케

일링의 세밀함이 상대적으로 떨어질 수 있다.

IV. 성능평가

본 장에서는 제안시스템을 평가하기 위한 2가지 실험을 진행하여 이를 분석하였다. 첫째로 본 논문에서 제안하는 스케일링 시스템의 성립여부를 판단하는 요소 중 하나인 운영 중인 가상화된 서버의 자원을 동적으로 확장이 가능한지를 입증한다. 둘째로 제안시스템의 효용성을 검증하기 위해 스케일 아웃기법을 사용한 시스템과의 비교를 하였다. 가상의 환경을 구성하여 시뮬레이션을 진행하여 결과를 분석한다.

1. 가상화 서버의 동적 자원 할당과 Live Migration 가능 여부 검증

가상화는 동적 자원 할당을 통해 관리에 탄력성을 갖는다는 장점을 갖는다. 따라서 이 장에서는 본 논문에서 제안하는 스케일링 시스템의 성립여부를 판단하는 중요한 요소 중 하나인 운영 중인 가상화된 서버의 자원을 동적으로 확장이 가능한지를 입증하고자 한다. 실제 가상화 환경을 구축하고 VM을 생성한 후 본 논문에서 제안하는 스케일링 알고리즘을 적용하여 VM이 동적으로 자원할당이 가능한지 실험하였다.

1.1 환경 구성

스케일링 테스트를 위해 사용된 물리적 서버는 표 1과 같이 Turing과 Dijkstra 두 대로 진행하고 VM은 2대를 생성하였다. VM에 할당 가능한 VCPU는 VM별로 총 4 Unit까지 가능하며 메모리는 물리적인 서버가 가지고 있는 최대 메모리양인 4GB까지이다. 실험을 위해 각 VM에 VCPU 1Unit, 메모리 1GB를 할당하여 운영체제 CentOS6을 설치하였다.

표 1. 물리 서버 환경
Table 1. Physical Server Environment

서버명	VMM	CPU	RAM(GB)
Turing	KVM	Intel(R) Xeon(R) X3430@2.40GHz (Quad core) 1EA	4GB
Dijkstra	KVM		

1.2 실험 결과

본 장은 1.1에서 설정한 테스트환경을 가지고 서버 운영 중에 자원할당이 동적으로 이루어지는지 판단하기 위해 생성된 VM을 실행시킨 후 스케일링 시스템을 동작시켰다.

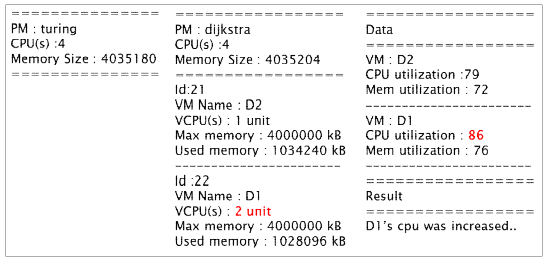

그림 5. 동적 할당 결과
Fig. 5. Result of Dynamic Allocation

그림 5는 예측된 자원 량에 따라 VM의 자원이 동적으로 변경된 상태를 캡처 한 결과이다. 예측된 값들 중 D1의 VCPU 자원 사용량이 86%로 80%를 넘었기 때문에 스케일링 알고리즘에 따라 D1의 VCPU를 1Unit 증가시켜 D1의 VCPU가 2 Unit으로 변경되었다.

그림 6은 D1의 VCPU는 3 Unit, D2의 VCPU는 1 Unit인 상태에서 D1의 예측된 CPU 자원 값이 83%로 상한 값을 넘었기 때문에 해당 PM에서 늘려줄 CPU자원이 없어 자원 할당이 낮은 D1을 Turing PM으로 Live Migration하고 D1의 VCPU 자원을 1Unit 늘려주었다.

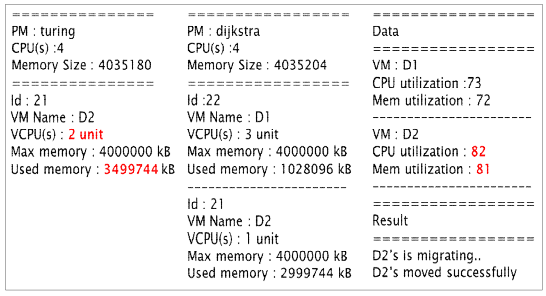

그림 6. Live Migration 결과
Fig. 6. Result of Live Migration

2. 제안 시스템과 스케일 아웃을 사용한 시스템을 비교를 통한 효용성 검증

본 장에서는 제안시스템을 평가하기 위한 시뮬레이션 환경 설정과 테스트 환경을 설명하고 시뮬레이션 결과에 대해 분석한다. 제안 시스템의 효용성을 검증하기 위해 가상의 환경을 구성하여 시뮬레이션을 진행해 제안 시스템과 접속사용자 수 기반의 Scale-out 기법인 Scalable Web App Architecture[11]를 비교 분석하였다.

2.1 환경 구성

시뮬레이션 내에서 스케일링 자동화가 이루어지도록 설정한 환경은 표 2와 같다. 표 2설정에 의거하여 스케일링이 이루어지게 된다. 클라우드 환경내의 PM 수는 무한하며 시간당 서버로 유입되는 동시접속자 수를 그림 7과 같이 변화시켰다. Hui Li[15]는 슈퍼컴퓨팅 환경에서의 워크로드를 분석하였으며, 이 연구 결과를 샘플링하여 그림 7과 같이 동시접속자 수를 변화 시켜 실험을 진행하였다. 총 250시간 동안 시뮬레이션을 진행하였으며 동시접속자 수의 변화에 따라 스케일링 시스템이 유연하게 대처할 수 있는지, 그리고 자원 이용률을 높게 유지할 수 있는지 확인할 수 있도록 하였다.

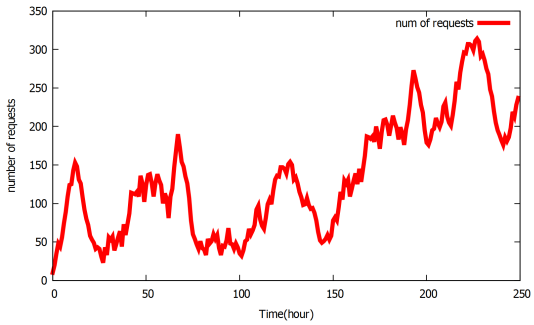

그림 7. 시간에 따른 동시 접속자 수의 변화
Fig. 7. Changes of the number of simultaneous connections

초기 VM1개를 운영하고 3개의 VCPU core와 3GB의 메모리를 갖도록 구성한 후 서버의 트래픽이 증감하는 상황에서 스케일링에 따른 서버 이용률을 측정하였다. 자원 이용률의 임계치 는 80%로 정하였다. 자원 이용률이 80%가 넘어가게 될 시 스케일링을 하게 된다. 스케일링 시 PM의 자원 상 여유가 있다면 스케일 업을 진행하고 PM의 자원 상 여유가 없다면 스케일 아웃을 진행한다.

표 2. 스케일링 설정
Table 2. Scaling Pre-configuration

PM CPU Core	6 Unit
PM RAM	6 GB
Initial VM CPU Core	3 Unit
Initial VM RAM	3GB
스케일업 시 CPU Core 증가량	1 Unit
스케일업 시 RAM 증가량	1GB
스케일업 시 VM CPU Core 한계치	5 Unit
스케일업 시 VM RAM 한계치	5 GB
임계치	80%

2.2 실험 결과

실험에서는 제안하는 Hybrid 기법과 Scalable Web App Architecture를 시뮬레이션 하여 VM 수의 변화와 서버의 이용률을 측정하여 비교 분석하였다.

그림 8은 시간에 따라 증감하는 동시접속자 수에 맞추어 시스템이 사용하는 VM의 수를 보여준다. 제안한 스케일링 방법이 스케일 아웃만을 사용한 방법보다 더 적은 VM을 운영하고 있음을 확인할 수 있다. 관리하는 VM의 수가 적을수록 시스템 복잡도가 낮아지고 운용하는 PM의 수가 줄어들기 때문에 전체 시스템 관리비용 또한 줄일 수 있다.

표 3은 VM의 변화를 분석한 결과이다. 제안한 스케일링 기법은 스케일 아웃의 횟수가 9번인데 비해 스케일 아웃 기법은 21번의 스케일 아웃을 함을 확인할 수 있다. 스케일 아웃은 PM에 VM의 복사본을 만들어야 하는 등 스케일 업에 비해 오버헤드가 큼을 고려했을 때 제안 시스템이 스케일 아웃 기법에 비해 스케일링 자원 사용량 측면에서도 효율적임을 확인할 수 있다.

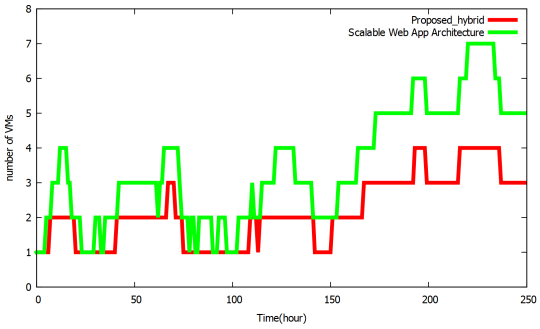

그림 8. 동시 접속자 수 증가에 따른 VM수의 변화
Fig. 8. Changes of the number of VMs caused by simultaneous connections

표 3. VM 수 변화 분석 결과
Table 3. Changes of the number of VMs

	본 논문에서 제안하는 Hybrid Scaling 방식	접속자수 기반 Scale-out 방식 [11]
Scale up 횟수	33	0
Scale out 횟수	9	21

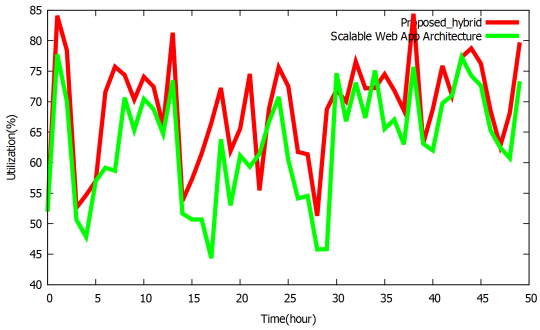

그림 9. 동시 접속자 수 변화에 따른 VM 자원 이용률 변화
Fig. 9. Changes of VM's utilization caused by simultaneous connections

표 4. 자원 이용률 평균 및 표준편차
Table 4. Average and standard deviation of utilization

	본 논문에서 제안하는 Hybrid Scaling 방식	접속자수 기반 Scale-out 방식 [11]
평균	68.92	63.22
표준편차	12.60	13.82

그림 9는 트래픽의 변화에 따른 시스템의 자원 이용률의 변화를 나타낸다. 실험 결과를 시각적 분석에 용이하도록 5구간의 평균을 이용하여 추세선으로 표현하였다. 표4는 자원 이용률 변화의 평균과 표준편차를 계산하였다. 스케일 아웃 기법은 자원 이용률의 변동이 크고 전체적으로 낮은 자원 이용률을 유지하는 반면, 제안 스케일링 기법은 자원 이용률의 변동이 상대적으로 크지 않고 전체적으로 높은 자원 이용률을 유지한다. 이와 같이 안정된 자원 이용률을 유지함으로써 자원을 효율적으로 사용하고 데이터센터의 전력낭비를 줄여 서버 운용비용을 낮추는 효과를 얻을 수 있다.

V. 결론

최근 다양한 웹 기반 애플리케이션 및 서비스의 사용이 활발해 짐에 따라 이를 지원하는 컴퓨팅 자원의 규모가 커지고 구조 또한 복잡해지고 있다. 이에 따라 효과적인 서버 자원 사용을 위해 기업체에서는 클라우드 컴퓨팅과 가상화 기술을 사용하여 IaaS를 도입, 서버의 낮은 활용률과 불필요한 서버 확보 문제를 해결하고 있다. 가상화 기술은 Hypervisor를 이용하여 서버의 물리적 자원을 추상화하여 가상화된 자원을 VM에 제공하며 서비스에 필요한 운영체제와 어플리케이션을 VM에서 운영하도록 한다. 또한 단일 서버에서 두 개 이상의 VM을 운영하도록 함으로써 탄력적이고 효율적인 자원의 활용이 가능하게 된다. 현재 공공 클라우드 환경에서 사용되는 스케일링 기법은 스케일 업 또는 스케일 아웃 기법 중 하나를 선택하여 사용되고 있다.

그러나 제한된 컴퓨팅 자원을 사용하는 사설 클라우드를 구축해 운영하며 높은 자원 활용률을 얻기 위해서는 좀 더 세밀화 된 자원 스케일링 기법이 필요하다. 본 연구에서는 가상화 환경에서 서버 운용유지 비용을 최소화하고 효과적인 서버 자원의 활용을 위한 Hybrid 스케일링 시스템을 제안하였다. 제안 시스템은 주기적으로 모니터링 된 자원 활용률 정보를 기반으로 스케일링 필요 여부를 판단하여 VM에 대한 동적 자원 할당과 Live Migration을 통해 자동화된 스케일링을 수행한다. 그리고 제안 시스템은 스케일 업과 스케일 아웃을 동시에 사용하기 위해 모델링 및 알고리즘을 개발하였다. 이를 통해 제안 시스템은 기존에 제안되었던 스케일 업과 스케일 아웃을 단독으로 사용하는 기법에 비해 더 효율적인 스케일링 성능을 가진다.

본 연구에서는 제안 시스템의 유효성 검증을 위해 실제 시스템의 프로토타입을 구현하였다. 프로토타입 시스템을 실제 서버에 구축을 하고 시나리오 기반의 테스트를 통해 제안 시스템의 기능 및 성능 확인을 통해 유효성을 검증했다. 또한 시스템에서 사용되는 알고리즘의 효율성 평가를 위해 시뮬레이션을 진행하였다. 시뮬레이션 결과를 통해 제안 알고리즘이 기존의 스케일링 기법에 비해 상대적으로 더 높은 스케일링 성능과 자원 활용률을 가짐을 확인할 수 있었다. 이러한 결과를 통해 제안하는 시스템 및 알고리즘은 클라우드 컴퓨팅 및 가상화 환경에서 효율적인 자원 관리 및 활용이 가능하게 할 것으로 판단된다.

본 연구에서 진행한 실험에서 스케일링이 처리할 수 없는 대용량 요청에 대해서는 자원이용률이 임계치를 넘는 현상이 짧은 시간이지만 나타난 것을 볼 수 있다. 이 문제의 해결방안에 대한 연구를 향후에 진행하여 회귀분석(Auto Regression)과 같은 예측모델을 적용한 수정된 알고리즘을 도출한다면 SLA를 만족시키면서 더욱 효율적인 스케일링이 가능할 것으로 보인다.

참고문헌

[1] O'Reilly, T. What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. 30 September 2005. Available from: http://www.oreillynet.com/lpt/a/6228
[2] J. Farrell, G. Nezlek, Rich Internet Applications

The Next Stage of Application Development, 29th Int. Conference on Information Technology Interfaces, 2007. ITI 2007, 2007, pp. 413-418.

〔3〕 D.A. Menascé and V.A.F. Almeida, Capacity Planning for Web Services: Metrics, Models, and Methods, Prentice Hall, Upper Saddle River, N.J., 2002.

〔4〕 K. Lee, S. Park. "A Dynamic Allocation Scheme for Improving Memory Utilization in Xen". Journal of Kiise : Computer System and Theory, Jun 2010.

〔5〕 C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, et al, "Live migration of virtual machines," NSDI, 2005.

〔6〕 W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation". Technical report, Clouds Laboratory, University of Melbourne, Australia, 5 April, 2009

〔7〕 T. Cucinotta, D. Giani, D. Faggioli, and F. Checconi, "Providing Performance Guarantees to Virtual Machines using Real-Time Scheduling," in Proc. of 5th Workshop on Virtualization and High- Performance Cloud Computing (VHPC 2010), August 30-31, 2010.

〔8〕 C. Sapuntzakis , D. Brumley , R. Chandra , N. Zeldovich , J. Chow , M.S. Lam, M. Rosenblum, "Virtual Appliances for Deploying and Maintaining Software", Proceedings of the 17th USENIX conference on System administration, October 26-31, 2003, San Diego, CA

〔9〕 A. Zahariev, "Google App Engine", Apr 20

〔10〕 Amazon Web Services: Elastic Load Balancing http://aws.amazon.com/elasticloadbalancing/

〔11〕 T. C. Chieu, A. Mohindra, A.A. Karve, A. Segal, "Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment", ICEBE 2009: 281-286

〔12〕 Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. CloudScale: Elastic resource scaling for multi-tenant cloud systems. SOCC '11, 2011.

〔13〕 A. Murtazaev, S. Oh, "Sercon: Server Consolidation Algorithm using Live Migration of Virtual machines for Green Computing", IETE Technical Review, Vol 28, Issue 3, 2011

〔14〕 T. Wood, P.J. Shenoy, A. Venkataramani, M.S. Yousif, "Sandpiper: black-box and gray-box resource management for virtual machines", Computer Networks 53(17) (2009) 2923-2938.

〔15〕 H. Li, D. Groep, L. Wolters, "Workload Characteristics of a Multi-cluster Supercomputer", Job Scheduling Strategies for Parallel Processing, New York, NY, Jun. 2004, pp. 176-193

〔16〕 M. Mao, "Cloud Auto-Scaling with Deadline and Budget Constraints", 11th IEEE/ACM International Conference on Grid Computing, 2010

〔17〕 P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. "Xen and the Art of Virtualization", In Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003), Bolton Landing, USA, Oct. 2003.

〔18〕 A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. "KVM: The Linux Virtual Machine Monitor", In Proceedings of the Linux Symposium, pages 225-230, 2007

〔19〕 S. Son, S. Jun, "Load Balancing Policy for Xen-based Virtual Desktop Service", Journal of The Korea Society of Computer and Information, Vol. 13, No. 1, January 2008

〔20〕 D. Cho, S. Park, "Development and Implementation of Monitoring System for Management of Virtual Resource Based on Cloud Computing", Journal of The Korea Society of Computer and Information, Vol. 18, No. 2, February 2013

저 자 소 개

이 돈 혁

2008 : 단국대학교 멀티미디어공학과 공학사

2010 - 현재 :

아주대학교 컴퓨터공학과 석사과정

관심분야 : SOA, 클라우드컴퓨팅

E-mail : grifith82@ajou.ac.kr

오 상 윤

2006 : 미 인디애나대학교 전산학 박사

2006 - 2007 : SK 텔레콤

2008 - 현재 :

아주대학교 컴퓨터공학과 부교수

관심분야 : 고성능컴퓨팅 (HPC), Large Scale Software, RDF, 클라우드 컴퓨팅

E-mail : syoh@ajou.ac.kr