과학 계산 응용 실행을 위한 하이브리드 클라우드에서의 SLA기반 VM 오토-스케일링 기법

(A SLA-based VM Auto-Scaling Method in Hybrid Cloud Computing for Scientific Computational Applications)

강 혜 정 † 고 정 인 † 김 윤 희 ††
(Hyejeong Kang) (Jung In Koh) (Yoonhee Kim)

요 약 클라우드 컴퓨팅은 동적인 자원 사용을 가능하게 함으로써 계산 과학 응용 수행에 활발히 이용되어 왔다. 계산 과학 분야에서 클라우드 컴퓨팅 환경을 이용하는 데 있어 중요한 이슈 중 하나는 응용 수행에 실제 필요한 양만큼 자원을 할당하여 사용하는 것이다. 실제 필요한 양만큼의 자원을 사용함으로써 자원 사용비용을 줄이기 위함이다. 그러나 가변적인 작업 부하를 갖는 현대 응용의 특성상 이는 매우 어려운 작업이다. 기존 상용 클라우드들은 규칙 기반의 메커니즘을 이용하여 동적인 자원할당을 시도하고 있으나 대부분의 기법들은 단순히 자원의 성능 지표만을 기준으로 삼음으로써 응용의 Service Level Agreement(SLA; e.g. 데드라인, 비용 등)는 고려하지 않는다. 본 논문에서는 사설/상용 클라우드로 구성된 하이브리드 클라우드 환경에서 가변적인 자원 요구에 따라 자원을 할당하고 SLA 위반을 최소화하는 오토-스케일링 기법을 제안한다. 실험을 통해 주어진 데드라인 내에서 동적인 자원 활용을 통해 작업을 수행하는 결과를 보임으로써 제안하는 오토-스케일링 기법의 우수성을 검증한다.

키워드: 오토-스케일링, 하이브리드 클라우드 컴퓨팅, SLA, 다중 정책

Abstract Cloud computing facilitates on-demand resource usage. Scientists can perform large-scale scientific computational experiments over cloud environment. Executing experiments in a timely manner with the proper number of resources is important for the scientists. Scientists can save the cost by using resources actually needed. However, providing resources for processing dynamic workload of modern applications have difficulty. Rule-based and schedule-based mechanisms have been tried to allocate resources dynamically, but most of the auto-scaling methods just simply support performance metric such as CPU utilization but rarely are aware of Service Level Agreements (SLA) including execution deadline or cost. In this paper, we propose an auto-scaling method that automatically allocates resources depending on variable resource requirements in hybrid clouds satisfying a user's requirements on SLA. We have evaluated the performance of our auto-scaling method. The result shows that the method can perform a scientific application with the proper number of resources in deadline constraints.

Keywords: auto-scaling, hybrid cloud computing, SLA, multi-policies

· 이 논문은 2013년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. NRF-2013R1A1A3007866)
· 이 논문은 2013 한국컴퓨터종합학술대회에서 '하이브리드 클라우드 환경에서의 SLA기반 오토-스케일링 기법'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 숙명여자대학교 컴퓨터과학부
hjkang@sookmyung.ac.kr
jungin@sookmyung.ac.kr
†† 종신회원 : 숙명여자대학교 컴퓨터과학부 교수
yulan@sookmyung.ac.kr
(Corresponding author임)

논문접수 : 2013년 8월 19일
심사완료 : 2013년 10월 11일

정보과학회논문지: 시스템 및 이론 제40권 제6호(2013.12)

1. 서 론

최근 계산 과학 분야에서 대용량 컴퓨팅 자원의 요구를 능동적으로 만족시키기 위해, 자원을 필요한 만큼 빌려 쓸 수 있는 클라우드 기술을 적용하여 과학 클라우드(Science Cloud)를 구축하는 연구가 세계 여러 곳에서 진행되고 있다. 빅데이터 처리 응용이나 계산 과학 응용(예, 과학 워크플로우)은 고성능 컴퓨팅(HPC: High Performance Computing) 또는 다처리 컴퓨팅(HTC: High Throughput Computing)을 요구하는 대규모 사이즈 응용(MTC: Many Task Computing)으로 정의할 수 있다. 이런 응용은 고성능의 자원을 장시간 활용해야 하며 안정적이고 지속적인 자원의 공급이 응용 실행의 필수이다. 이는 엑사스케일(Exascale) 컴퓨팅, 페타스케일(Petascale) 컴퓨팅의 연구로 이어지고 있으며 이런 환경에서 성공적인 작업 수행을 위한 실행관리, 자원 관리 등을 제공하는 계산 문제 풀이 환경(Computational Problem Solving Environment)에 대한 연구가 더욱 절실하다.

이런 문제 풀이 환경에 필요한 클라우드 자원을 효율적으로 활용하기 위한 방법으로 온-디맨드 자원 가상화 특성을 활용한 오토-스케일링 기법을 고려할 수 있다. 오토-스케일링 기법은 응용의 수행 중에 작업 수행을 위한 가상 자원의 수를 줄이거나 늘리는 방법으로 AWS(Amazon Web Service)[1]의 “Auto-scaling” 서비스가 대표적인 예이다. AWS나 Scalr[2]와 같은 오토-스케일링의 서비스들은 사용자가 정의한 규칙(예, CPU 이용량이 60% 이상일 때, 가상머신 2개 추가 할당)을 기반으로 자원의 규모를 결정한다. 단순한 방법으로 오토-스케일링을 가능케 하는 장점이 있지만 현대 응용의 가변적인 워크로드로 인한 동적인 자원 요구 패턴을 만족하지 못하는 문제점이 있다. 특히, 응용의 자원 요구가 충족되지 않을 경우 응용의 데드라인 위반, 수행 실패의 증가와 같은 SLA(Service Level Agreement) 위반으로 이어질 수 있는 심각한 문제점이 존재한다.

본 논문에서는 하이브리드 클라우드 컴퓨팅 환경에서 자원을 효율적으로 활용하기 위한 오토-스케일링 기법을 제안한다. 제안하는 오토-스케일링 기법은 데드라인, 비용/성능 정책과 같은 SLA를 충족한다. 또한 계산 집약, I/O 집약, 데이터 집약 등 응용의 특성과 함께 상용 클라우드의 인스턴스 타입을 고려함으로써 비용 효율적인 오토-스케일링이 가능하다.

본 논문의 구성은 다음과 같다. 1장의 서론에 이어 2장에서는 관련 연구들을 살펴보고, 3장에서는 본 논문에서 제안하는 SLA기반 오토-스케일링 기법을 적용한 프레임워크 구조에 대하여 설명한다. 4장에서는 SLA 기반 오토-스케일링 기법에 대해 설명하고 5장에서는 실험에 대해 살펴본다. 마지막으로 6장에서 결론을 맺는다.

2. 관련 연구

클라우드 컴퓨팅은 가상화 기술을 통해 자원의 무한한 제공이 가능하고, 자원 규모의 확장과 축소가 용이하다. 이로 인해 효율적인 자원 관리 방법으로 오토-스케일링 기법에 대한 연구가 활발히 이루어지고 있다.

대표적인 오토-스케일링 기술로는 AWS[1]의 “Auto-scaling” 서비스가 존재함을 앞서 언급하였다. 이 서비스는 사용자가 정의한 규칙을 통해 자원의 규모를 확장하고 축소하는 규칙 기반의 자원 할당 자동화 기법으로 CPU 사용량, 디스크 사용량과 같은 하드웨어의 성능과 관련된 수치를 기준으로 한다. 기준이 되는 성능치에 상한 값과 하한 값을 정하고 그 값을 기준으로 미리 정해진 수만큼 가상머신을 추가하거나 제거하여 자원의 규모를 확장하거나 축소시키는 방법이다. 비슷한 서비스로 Windows Azure[3]에서 사용되는 Paraleap[4]과 Scalr[2], RightScale[5]이 존재한다. 규칙 기반의 스케일링 기법은 비교적 단순한 방법으로 동적인 자원 할당이 가능하지만 복잡한 수행 패턴을 갖는 작업에 대해서 실제로 필요한 만큼 가상머신이 추가/제거되지 않을 수 있다. 이러한 문제는 작업 수행 데드라인이나 자원 사용 비용과 같은 SLA를 갖는 작업들에서는 SLA 위반이라는 큰 문제를 야기할 수 있다.

작업 수행 데드라인이나 자원 사용 비용 등을 고려한 오토-스케일링 기법 연구로는 [6,7]이 존재한다. [6]에서는 사용자가 요청한 작업의 데드라인 내에서 작업 수행이 가능하도록 오토-스케일링을 수행한다. 이 연구에서는 작업의 데드라인뿐만 아니라 자원 할당에 따른 비용까지 고려하였다. 그러나 상용 클라우드를 단독으로 사용하였을 경우의 자원 할당만을 고려하였다는 제한사항이 존재한다. [7]은 자원 사용 비용을 최소화하기 위해 물리머신(Physical Machine)의 사용량과 자원 사용료 간에 트레이드-오프가 최적인 물리머신 사용량을 찾아 해당하는 만큼 가상머신을 스케일링하는 기법을 제안한 연구이다. 가상머신의 개수를 추가/제거하여 자원 규모를 확장하고 축소하는 수평적(Horizontal) 스케일링과 가상머신의 하드웨어 사양을 조절하는 수직적(Vertical) 스케일링 기법을 혼합 활용한다. 그러나 이 연구에서도 수평적 스케일링 시에는 고정된 개수의 가상머신만을 추가/제거하여 복잡한 수행 패턴의 작업에 대하여 동적인 자원 할당이 가능하다고 하기엔 부족하다.

본 논문에서는 하이브리드 클라우드 환경에서 데드라인과 자원 사용 비용과 같은 SLA를 고려하여 자원 할당이 가능한 오토-스케일링 기법을 제안한다. 지속적인 모니터링을 통하여 자원 할당이 이루어지며 자원 할당

시에 필요한 가상머신의 개수를 계산하므로 그때그때 적절한 양의 가상머신을 추가/제거할 수 있다.

3. 오토-스케일링 프레임워크 구조

본 논문에서 제안하는 오토-스케일링 기법을 지원하는 프레임워크의 구조는 그림 1과 같다. 응용은 SLA와 함께 프레임워크에 제출되며, 응용의 작업들은 큐에서 수행을 기다리고 SLA는 '작업 메타데이터' 모듈로 전달된다.

오토-스케일링 프레임워크는 네 가지 주요 서비스로 구성되어 있다. '메타데이터 관리 서비스'(MMS)는 작업 정보, 실행 이력, 자원 정보, 가상머신의 정보 등의 정적 정보를 관리한다.

'오토-스케일링 서비스'(ASS)는 프레임워크의 핵심 서비스이다. 동적인 자원 요구를 충족하기 위하여 ASS는 정기적인 모니터링 인터벌마다 수행되고 해당 인터벌에 실제로 필요한 자원만을 작업에 할당하게 된다. ASS서비스는 '스케줄링', 'SLA 모니터링', '실시간 스케일링' 세 가지 모듈로 구성되어 있다. '스케줄링' 모듈은 SLA에서 명시한 정책을 기반으로 자원 요구에 알맞은 가상머신에 작업을 할당하는 역할을 수행한다. 'SLA 모니터링' 모듈은 성능과 비용을 측정한다. '실시간 스케일링' 모듈은 응용 수행을 위해 새롭게 생성해야 할 가상머신과 Release시켜야 할 가상머신의 수를 결정한다. '실시간 스케일링' 모듈의 결정에 따라 '동적 자원 관리 서비스'와 '작업 실행 서비스'가 동작하게 된다.

'동적 자원 관리 서비스'(DRMS)는 ASS에서 결정된 대로 가상머신을 할당하는 동적 자원 할당과 가상머신 모니터링을 담당한다. '작업 실행 서비스'(JES)는 자원이 아닌 작업에 대한 관리를 수행하는 서비스로 '작업 모니터링' 모듈을 통해 작업의 상태를 관찰하고 '작업 실행' 모듈을 통해 DRMS 에서 생성한 가상머신에 작업을 할당한다.

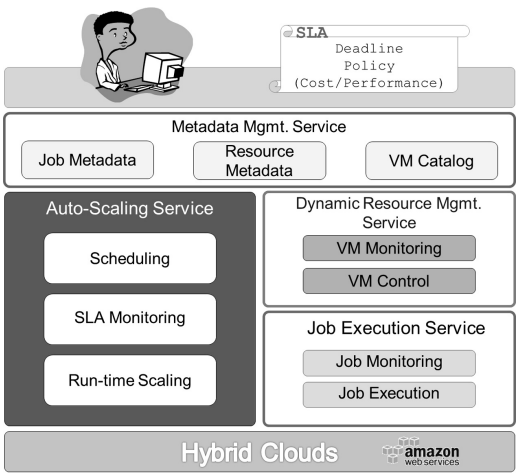

그림 1 오토-스케일링 프레임워크 구조

Fig. 1 Structure of auto-scaling framework

4. SLA 기반 오토-스케일링 기법

본 논문에서는 하이브리드 클라우드 환경에서의 오토-스케일링 기법을 제안한다. 작업 부하가 동적으로 변하는 현대 응용의 특성상 작업 수행에 대한 예측 및 자원 사용량에 관한 예측이 힘들기 때문에 주기적인 모니터링을 통해 시시각각 변하는 자원 요구를 충족시켜줄 필요가 있다. 자원 요구를 만족시키는 것과 동시에 데드라인, 자원 사용 비용과 같은 SLA로 정의되는 사용자 요구사항을 충족시키는 것 또한 중요하다. 따라서 제안하는 기법에서는 비용/성능 두 가지 정책과 함께 데드라인을 SLA로 정의하고 데드라인을 기준으로 사용자가 선택한 정책기반의 자원 할당과 오토-스케일링을 수행한다. 비용 정책은 사용자가 상용 클라우드 자원을 사용함으로써 발생하는 자원 이용료를 고려하여 합리적인 비용으로 데드라인 내에서 작업을 수행하도록 자원을 할당하는 정책이고, 성능 정책은 최대한 빨리 작업 수행을 완료하도록 자원을 할당하는 정책이다. 성능 정책을 선택할 경우 데드라인은 최상 경로(Critical Path) 이상의 값을 가져야 하며, 최소 성능 요구도 SLA로 함께 포함된다. 즉 자원 할당 시에 사용자가 요구한 최소 성능 이상의 사양을 갖는 자원을 할당하는 것이다. 자원 요구에 따라 자원 규모를 확장하는 스케일 아웃의 경우 정책에 따른 자원 할당 시에 이루어지며 자원 규모의 축소인 스케일 인의 경우 자원 할당이 끝난 후 일괄적으로 이루어지게 된다.

4.1 SLA 기반 오토-스케일링 시나리오

그림 2와 그림 3은 제안하는 오토 스케일링 기법의 최초 자원 할당 과정 및 결과를 보여준다. 작업들은 수행시간을 기준으로 내림차순으로 정렬한 후, 자원 이용료가 들지 않는 사설 클라우드 자원에 최대한 많은 작업을 할당한다. 이후 데드라인을 위반할 것이라 예상되면 상용 클라우드에서 자원을 구매하여 작업을 수행한다. 상용 클라우드의 자원에 작업을 할당할 경우, 이미 Running 중인 자원을 우선 고려하는데 이때 새로운 가상머신을 샀을 때의 가격, 인스턴스 타입을 고려하여 새로운 가상머신의 구입 여부를 결정한다. 인스턴스 타입을 고려함으로써 성능대비 비용 효율적인 자원을 사용하도록 하였다. 여기서 성능대비 비용 효율적이라 함은 계산 집약적인 작업을 수행한다고 했을 때, Amazon EC2의 인스턴스 타입을 기준으로 같은 사양의 CPU 성능을 갖는 일반 Standard 인스턴스와 High-CPU 인스턴스의 가격 비교 시, 후자의 가격이 싸기 때문이다.

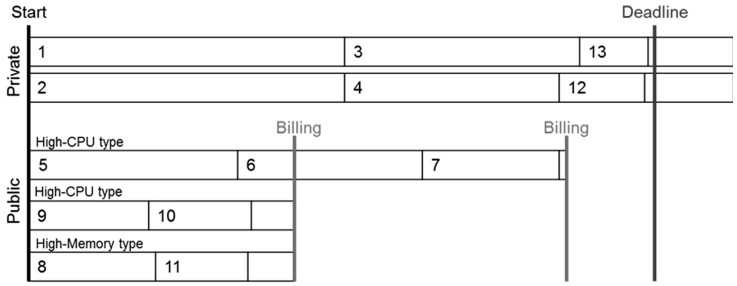

그림 2 최초 자원 할당 - 빌링 인터벌 고려
Fig. 2 Initial resource allocation - consideration of billing interval

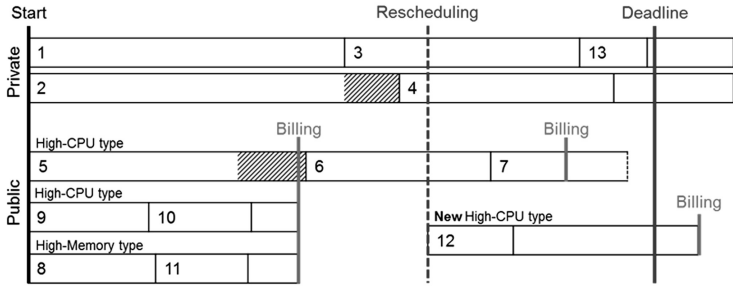

그림 3 최초 자원 할당 결과
Fig. 3 Initial resource allocation

13개의 작업을 작업 길이 순으로 정렬하여 각각 번호를 1부터 13까지 붙였다고 가정하자. 그림 2에서 볼 수 있듯이 1 번 작업부터 4번 작업까지는 사설 클라우드 상에서 데드라인 내에 수행할 수 있으므로 사설 클라우드 자원에 할당된다. 5번 작업의 경우 사설 클라우드 자원에서 수행할 경우 데드라인을 위반할 것이므로 상용 클라우드에 새로운 가상머신을 띄워 작업을 할당하게 된다. 이 때, 5번 작업이 계산집약타입이므로 High-CPU 타입의 가상머신을 띄운다. 그 다음 작업인 6, 7 작업의 경우 모두 계산집약타입이고, 5번 작업이 할당된 가상머신에 할당했을 때에 데드라인을 위반하지 않으므로 그대로 해당 가상머신에 할당한다. 데이터집약타입의 8번 작업은 7번 작업 뒤에 할당될 경우 데드라인을 위반하므로 High-Memory type의 새로운 가상머신을 띄워 할당한다. 9번 작업은 계산집약타입이다. 9번 작업이 현재 띄워져 있는 High-CPU 가상머신과 High-Memory 가상머신 중에서 High-Memory 가상머신에 할당될 경우에 데드라인을 위반하지 않는다. 그러나 만약 그대로 High-Memory 가상머신에 할당될 시에 낮은 CPU 성능으로 인해 지연이 발생하고 이에 따라 수행시간이 가격책정시간을 넘기므로 2달러만큼 비용이 새롭게 발생한다. 반면 자신의 작업타입에 알맞은 High-CPU 가상머신을 새로 띄울 경우 0.74달러의 비용이 발생한다. 따라서 발생 비용이 적은 방법인 새로운 High-CPU 가상머신을 띄워 9번 작업을 할당하게 된다(그림 2). 여기서의 가격 및 자원 성능 기준은 모두 Amazon EC2의 것을 그대로 따랐다. 10번 작업의 경우 계산집약타입 작업이므로 Running중인 가상머신 중에서 가장 빨리 시작할 수 있고 추가 비용이 발생하지 않는 가상머신(9번 작업이 수행중이 가상머신)인 High-CPU 가상머신에 할당된다. 데이터집약타입인 11번 작업도 마찬가지로 가장 빨리 작업 수행을 시작할 수 있고 추가비용이 발생하지 않는 High-Memory 가상머신에 할당된다. 마지막 작업들인 12, 13번 작업은 모두 사설 클라우드 자원에서 데드라인 이내에 수행 가능하므로 사설 클라우드에 할당된다(그림 3).

그림 4는 지속적인 모니터링을 통해 응용 수행의 데드라인 위반 여부를 확인하고 자원 요구량 변화에 따른 오토 스케일링을 수행한 결과를 보여준다. 현재 자원에서 수행되는 작업들에 대해서 예상 수행 시작 시간과 실제 수행 시작 시간을 비교하여, 이전 자원 할당 시에 예상한 것보다 작업의 시작이 지체되었을 경우(작업 4, 6) 앞서 수행된 작업의 지연(작업 2, 5)으로 판단하고 지연 정도를 계산한다. 만약 이 지연으로 인해 데드라인 위반(작업 12)이 일어날 것이라 판단되면 아직 대기 상태에 있는 작업들(작업 7, 12, 13)에 대하여 자원 할당을 새롭게 수

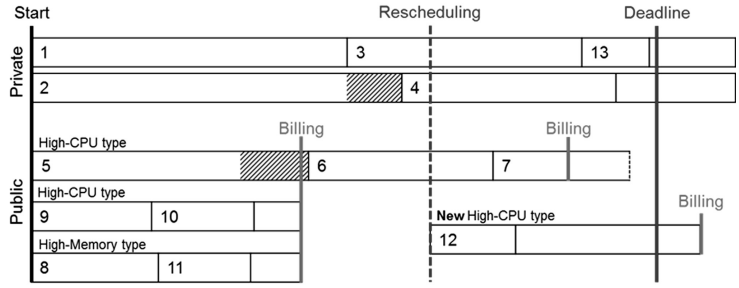

그림 4 모니터링 및 오토-스케일링 결과
Fig. 4 Result of monitoring and auto-scaling

행한다. 작업 길이에 따라 7, 12, 13의 순으로 자원 할당이 이루어진다. 사설 클라우드에 가용 자원이 존재하므로 우선 고려되고 7, 12의 경우 사설 클라우드에서 수행할 경우 데드라인을 위반하므로 상용 클라우드 자원에 할당된다. 작업 7은 Running중인 High-CPU type의 가상머신에 할당되며, 작업 12는 Running중인 가상머신에 할당할 경우 데드라인을 위반하므로 작업타입에 맞는 High-CPU type 가상머신을 새로 띄워 할당한다(Scale-out). 마지막으로 작업 13이 사설 클라우드에 할당되고 자원 할당이 끝난 후 유휴 자원이 존재할 경우 해당 자원을 Release하여 자원 규모를 축소한다(Scale-in).

4.2 SLA 오토-스케일링 알고리즘

제안하는 기법의 알고리즘은 SLA를 기반으로 작업을 스케줄링하고 일정한 주기마다 자원 및 작업에 대한 모니터링을 통해 오토 스케일링을 수행한다. 또한 데드라인까지 작업 수행을 완료해야 하는 상황에서 성능 지향 정책과 비용 지향 정책에 따라 자원을 적절히 프로비저닝하여 자원 활용의 효율 극대화 및 효율적인 비용 산출을 목표로 오토 스케일링을 수행한다. 제안하는 알고리즘에서 사용하는 기호 및 상세는 표 1과 같다.

표 1 오토-스케일링 알고리즘 표기
Table 1 Notation of the proposed auto-scaling

Notation	Explanation
Application = $\{j \mid j_i, i = 1,2,\cdots,n\}$	An application is consist of many jobs
SLA = {a policy P [, a deadline D, minimum_performance_requirement_$minPM$]}	SLA of an application. It has a policy, a deadline and minimum performance requirement in case of performance-oriented policy. (*[] means optional)
$Cost$	The total cost for resource usage.
IT_j	Instance type of public cloud which matches the job j.
vm	Running VMs.
EFT_{vm}	Estimated finish time of a VM
EST	Earliest start time of a job on a VM.
AST	Actual start time of a job on a VM.
ET	Execution time of a job on a VM.
PM_{vm}	Performance specification of a VM.

4.2.1 전체 알고리즘

Algorithm 1은 제안하는 오토 스케일링의 전체 알고리즘이다. 만약 SCALING이 true라면(line 3) 작업의 길이가 긴 순서대로 모든 작업을 정렬하고 SLA에 명시된 정책대로(비용 또는 성능 지향 정책) 각 작업을 스케줄링하게 된다. 비용 지향 정책은 사용자로부터 응용 수행을 완료하기 원하는 데드라인을 입력받음으로써 주어진 데드라인 내에서 비용 효율적인 스케줄링 될 수 있도록 작업을 적정 자원에 할당한다(line 9). 그리고 성능 지향 정책은 사용자로부터 이용 자원에 대한 최소 성능 요구 조건을 추가로 입력받아 최소 성능 요구 조건을 만족하는 자원에 작업을 할당한다(line 7). 작업에 대한 스케줄링을 마친 후, 작업을 수행하고 있지 않거나, 큐에 대기 중인 작업도 없는 VM이 있는지 알고리즘을 통해 확인한다. 만약 그런 VM들이 존재한다면 해당 VM들을 끄기 위해서 toShutDown 리스트에 추가한다(line 13). 자원 스케일링 및 작업 스케줄링 정보는 각각 동적 자원 관리 서비스와 작업 수행 서비스로 전달되게 된다. SLA에 대한 자원 및 작업 모니터링은 일정한 간격으로 수행되고 SCALING 값을 true또는 false로 설정함으로써 오토 스케일링에 대한 수행 여부를 결정한다.

Algorithm 1 – Run-time Scaling
Input – An application,
SLA={a policy *P*, a deadline *D* [, minimum performance requirement *minPM*]}
Output – Scaling decision $S = \{ toStartUp, toShutDown \}$
Scheduling decision $S = \{ jobs \rightarrow VMs \}$

```
1:  SCALING ← TRUE;
2:  while (true)
3:      if SCALING == TRUE
4:          Sort waiting jobs in decreasing order of execution length;
5:          switch P
6:              case Performance:
```

```
 7:                S ← PerformanceOrientedScheduling(
                                  sortedJobs, D, minPM); break;
 8:            case Cost:
 9:                S ← CostOrientedScheduling(sortedJobs, D); break;
10:        end switch
11:        for each vm where status is running do
12:            if no running/waiting jobs on vm then
13:                add vm to toShutDown;          //destroy the vm
14:            end if
15:        end for
           //send scaling decisions to DRMS
16:        send(DRMS, {toStartUp, toShutDown});
           //send scheduling decisions to JES
17:        send(JES, S);
18:        waitForNextInterval();
19:        SCALING ← SLAMonitoring(runningJobs, D);
20:    end if
21: end while
```

Algorithm 2는 SLA 위반을 검사하는 모니터링 메커니즘이다. 일정 주기마다 각 작업의 예상 수행 시간(EST)과 실제 시작 시간(AST)의 차이를 계산한다(line2). 이 값을 delay 변수로 선언하지만 음수 또는 양수일 수 있는데 작업에 대한 지연이 발생하면 양수인 delay를 얻게 되고, 예상 시간 보다 작업들이 일찍 수행될 경우에 대해서는 음수 delay를 얻게 된다. 양수인 delay는 해당 작업이 속해있는 VM에 스케줄링 되어있는 가장 마지막 작업의 예상 완료 시간에 delay 값을 더해서, 후반 작업들의 수행이 데드라인을 위반하는지 확인한다. 반대로 음수인 delay는 가상자원을 축소시키는 기준값(threshold) τ를 넘기는지 확인한다. 두 조건 중 하나라도 만족하게 되면 SCALING 값을 ture로 설정하고, 이를 통해서 실시간 스케일링 알고리즘이 가상자원 풀을 재조정하게 된다.

Algorithm 2 – SLA Monitoring
Input – Running jobs of the application, the deadline *D*
Output – Whether scaling is required or not, *SCALING*

```
1: for each job j do
2:     delay ← AST - EST;
3:     if (EFT_currentVM + delay) > D or -delay > τ then
4:         return TRUE;
5:     end if
6: end for
7: return FALSE;
```

4.2.2 정책기반 스케줄링 알고리즘

제안하는 스케일링 기법은 기본적으로 비용 지향 정책을 수반한다. 수행할 작업들에 대해서 사설 클라우드 자원에 우선 할당을 원칙으로 함으로써 기본적인 상용 클라우드 자원 사용 비용을 최소화하고자 하였다. 또한 상용 클라우드의 자원에 작업을 할당할 경우, 이미 Running 중인 자원을 우선 고려하는데 이때 새로운 가상머신을 샀을 때의 가격, 인스턴스 타입을 고려하여 새로운 가상머신의 구입 여부를 결정한다.

Algorithm 3은 이러한 비용 고려사항이 포함되어 있는 정책 기반 알고리즘을 보인다. 알고리즘 입력에서 PMvm은 최소 성능 요구 조건으로 이 항목이 SLA에 포함되어 있을 경우 성능 지향 정책으로 판단하여 오토-스케일링을 수행한다. 성능 지향 정책의 경우 최소 성능 요구 이상의 자원만을 이용 자원 후보군으로 지정하고 수행 기준이 되는 데드라인은 수행 응용의 Critical Path로 가정한다. 알고리즘에서는 각 작업들이 각 자원에서 가능한 빨리 실행될 수 있는 시간(EST)에 작업의 수행 시간을 더함으로써 작업의 예상 완료 시간(EFT)를 구해서 가장 빠른 예상 완료 시간을 갖는 자원에 해당 작업을 할당한다(line 4-9(private), line 11-20(public)). 만약 응용이 데드라인을 위반할 가능성이 있다면 상용(공용) 클라우드 자원을 구매해서 수행 자원의 규모 확장을 통해 응용의 전반적인 수행 속도를 가속화할 수 있다(line 22). 그리고 만약 수행 자원으로 상용 클라우드 자원을 고려할 경우에는 상용 클라우드의 가격 책정 방식을 고려하여 작업 수행에 이용 중인 VM이 우선적으로 선택된다. 만약 이용 중인 VM의 가격 책정 시간이 어느 정도 남아있고 수행하려는 작업의 특성이 이 자원에서 제공하는 특성과 일치한다면 ChooseCheeperVM()함수를 이용해서 새로 VM을 띄울 때 드는 비용과 비교해서 더 저렴한 쪽을 선택한다(line 14-16). 하지만 수행 중인 자원 중에서는 작업이 불가능하다면 해당 작업의 특성에 맞는 인스터스 타입의 VM을 새롭게 띄우고 이 VM을 toStartUp 리스트에 추가해서 자원의 확장을 알린다.

Algorithm 3 – Performance-oriented Scheduling
Input –Waiting jobs of the application,
SLA={a policy *P*, a deadline *D* [, minimum performance requirement *minPM*}
Output –Scheduling decision *S* = { *jobs* → *VMs*}, VM list *toStartUp* for the VMs to be newly created

```
 1: VM ← null;
 2: toStartUp ← null;
 3: for each job j do
 4:     for each private vm where PM_vm >= minPM do
 5:         VM ← find a vm on which j can start the most quickly
                                              within the D;
 6:     end for
 7:     if VM is not null then
 8:         schedule j to vm;
 9:         continue with the next job;
10:     else
11:         for each public vm where PM_vm >= minPM do
12:             VM ← find a vm on which j can start the most quickly
                                              within the D;
13:         end for
14:         if VM is not null then
15:             VM ← ChooseCheaperVM( VM, IT_j );
```

```
16:            schedule j to vm;
17:            if VM is IT_j then
18:                add IT_j to toStartUp; //to create a new VM of IT_j type
19:            end if
20:            continue with the next job;
21:        else
22:            add IT_j to toStartUp;      //to create a new VM of IT_j type
23:        end if
24:    end if
25: end for
26: return S;
```

5. 실 험

하이브리드 클라우드 환경에서 오토 스케일링 알고리즘의 성능을 검증한다. CloudSim[8]을 이용해서 Bag of Tasks 형식의 CFD[9] 기반의 응용을 대상으로 시뮬레이션을 수행한다. 특히, CFD 기반 응용은 e-AIRS 2.0[10]을 이용한 1년간 축적한 작업 프로파일을 분석하여 응용의 평균 길이 예측치를 계산하였으며 이를 실험에 이용하였다.

표 2 실험에 사용된 자원 사양
Table 2 Resource Specification for Experiment

Type of Resources		Specification				
		# of Cores /a VM	MIPS	Memory	Price /1 hour	# of VMs
Private Cloud		2	1000	4096 MB	-	2
		2	2000	4096 MB	-	2
Public Cloud	VM_1	5	600	2048 MB	$0.185	auto -scale
	VM_2	5	2400	2048 MB	$0.740	auto -scale

본 성능평가에서는 각 응용의 작업들을 5,000개로 설정하였다. 표 2에 명시되어있는 컴퓨팅 환경을 기반으로 시뮬레이션을 수행하였다. 먼저 사설 클라우드는 각각 2개씩 총 4개를 사용할 것이며, 각 자원 당 코어수는 2개, Memory는 4096MB로 동일하지만 MIPS는 각각 1000, 2000을 주었다. 그리고 공용 클라우드로는 MIPS만 600, 2400으로 다르게 하여 제출되는 작업에 따라 오토 스케일링 기법을 이용하여 VM 이용 개수를 달리한다. 또한 CFD 응용의 작업의 평균 길이는 660,000MI(Million Instruction)을 기준으로 하고, 5,000개의 작업 길이를 정규분포를 이용해서 랜덤하게 생성한다.

5.1 데드라인에 따른 작업 실패 수 비교

그림 5는 비용 지향 스케줄링 정책을 이용해서 CFD 응용을 수행한 결과이다. 초기 스케줄링과 제안하는 오토 스케일링 기법에 대해서 수행할 데드라인의 변화에

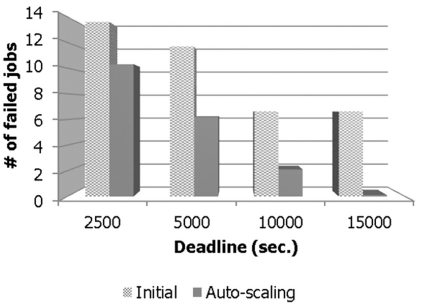

그림 5 초기 스케줄링과 오토-스케일링했을 때 수행 실패한 작업 개수 비교
Fig. 5 Comparison of the number of failed jobs between Initial scheduling and Auto-scaling

따라 수행에 실패한 작업들의 개수를 비교하였다. 초기 스케줄링은 비용 지향 정책에 기반을 둔 초기 스케줄링에 따라 작업을 수행하는 것이고, 본 연구에서 제안하는 오토 스케일링 기법은 초기 스케줄링 이 후, 일정 간격마다 수행 중인 작업에 관한 모니터링을 통해 데드라인 위반 가능성이 발견되면 대기 중인 작업에 대해서 리스케줄링을 수행하는 것이다. 각 작업이 수행되기 전에 해당 작업의 수행 시간을 예측해서 자원에 작업을 스케줄링하였기 때문에 수행 도중 발생하는 작업의 지연 및 예측 오차로 인한 데드라인 위반이 발생할 수 있다. 제안하는 기법은 이 같은 상황을 예방함으로써 데드라인 위반율을 감소하는 방향으로 작업을 리스케줄링 하게 된다. 그래프에 나타난 것과 같이 두 방법 모두 데드라인이 길어질수록 실패하는 작업의 개수가 감소하지만 초기 스케줄링보다 제안하는 기법이 상대적으로 더 많은 작업을 성공적으로 수행한 것을 볼 수 있다. 데드라인 위반 가능성을 보이는 작업들에 대해 오토 스케일링을 통해서 자원을 확장시킴으로써 데드라인 내에 수행을 최대화한다. 데드라인이 15000초일 경우 제안하는 기법은 자원 확장을 통해서 모든 작업을 데드라인 내에 수행시켰다.

5.2 지연 발생 시 VM 수의 변화

그림 6은 비용 지향 스케줄링 정책을 이용하여 CFD 응용을 수행하였을 때 시간의 경과에 따라 자원이 확장되는 정도를 나타낸 그래프이다. 앞서 수행한 실험과 같은 환경에서 실험을 진행하였다. 본 실험에서 데드라인은 앞선 실험에서 데드라인 위반이 없었던 15000초이고, 모니터링 인터벌은 500초이다. 그림 6은 본문에 맞추기 위하여 1500초마다 VM의 개수를 나타내었다. 그림을 통해 본 논문에서 제안하는 오토-스케일링 기법이 작업 수행 지연이 발생하였을 때 필요한 자원을 동적으로 추가함으로써 응용의 자원 요구에 유연하게 대처함을 알 수 있다.

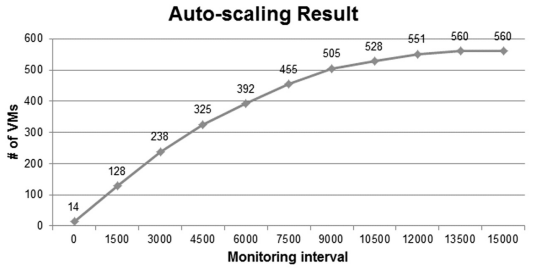

그림 6 각 모니터링 인터벌에서의 오토-스케일링 결과
Fig. 6 Result of auto-scaling at each monitoring interval

6. 결론 및 향후 연구

본 논문에서는 하이브리드 클라우드 컴퓨팅 환경에서 동적인 자원요구를 충족시키기 위한 오토-스케일링 기법을 제안하였다. 제안하는 오토-스케일링 기법을 통하여 데드라인 내에서 작업 수행에 필요한 자원을 필요할 때에 필요한 만큼만 구매하여 사용할 수 있으며, 이를 통해 사용자의 자원 사용 비용을 최소화할 수 있다. 또한 제공되는 비용과 성능 정책 중, 성능 정책 선택 시 비용에 대한 고려를 최소화하고 작업의 수행시간을 단축하여 시간적인 이득을 볼 수도 있다.

향후에는 전력 소모량을 고려한 에코-스케일링 등의 다양한 정책을 추가로 연구할 예정이다.

References

[1] Amazon Web Service, http://aws.amazon.com/
[2] Scalr, http://scalr.com/
[3] Windows Azure, http://www.windowsazure.com/
[4] Paraleap, https://www.paraleap.com/
[5] RightScale, http://www.rightscale.com/
[6] Ming Mao and Marty Humphrey, "Auto-Scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows," *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 12-18, 2011.
[7] Dutta, S., Gera, S., Verma, A., and Viswanathan, B., "SmartScale: Automatic Application Scaling in Enterprise Clouds," *Proceedings of 2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, pp.221-228, Jun. 2012.
[8] Rodrigo N., Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol.41, no.1, pp.23-50, 2011.
[9] Computational Fluid Dynamics, http://www.cfd-online.com.
[10] Kim, Y., Kim, E.-k., Kim, J. Y., Cho, J.-h., Kim, C. & Cho, K. W., "e-AIRS: An e-Science Collaboration Portal for Aerospace Applications," *HPCC LNCS (Lecture Note in Computer Science)*, vol.4208, pp. 813-822, 0302-9743, 2006.

강 혜 정
2010년 제주대학교 컴퓨터공학과(학사) 2012년 숙명여자대학교 컴퓨터과학부(석사). 관심분야는 그리드 컴퓨팅 환경(PSE) 온톨로지, 지능형 시스템

고 정 인
2011년 숙명여자대학교 컴퓨터과학부(학사). 2013년 숙명여자대학교 컴퓨터과학부(석사 수료). 관심분야는 클라우드 컴퓨팅 환경, 태스크 스케줄링, 하이브리드 클라우드 컴퓨팅

김 윤 희
1991년 숙명여자대학교 전산학과(학사) 1996년 Syracuse University 전산학과(석사). 2000 Syracuse University 전산학과(박사). 1991년~1994년 한국전자통신연구원 연구원. 2000년~2001년 Rochester Institute of Technology 컴퓨터공학과 조교수. 2001년~2004년 숙명여자대학교 컴퓨터과학과 조교수. 2004년~2009년 숙명여자대학교 컴퓨터과학과 부교수 2009년~현재 숙명여자대학교 컴퓨터과학부 교수. 관심분야는 그리드 컴퓨팅 환경(PSE), 워크플로우 제어, 그리드/클라우드 관리