하이브리드 클라우드상의 과학응용을 위한 가상 자원 오토 스케일링 기법

(An Auto-Scaling Technic of Virtual Resources on Hybrid Clouds for Scientific Applications)

안 윤 선 † 정 솔 † 김 윤 희 ††
(Younsun Ahn) (Sol Jeong) (Yoonhee Kim)

요 약 계산 과학 분야에서 자원을 필요한 만큼 빌려 쓸 수 있는 클라우드 기술을 적용하여 과학 클라우드(Science Cloud)를 구축하는 연구가 활발히 진행되고 있다. 장시간 동안 고성능의 자원을 활용해야 하며 안정적이고 지속적인 자원의 공급을 필요로 하는 대규모 작업 계산 응용이 있다. 이러한 응용의 성공적인 작업 수행과 클라우드 자원을 효율적으로 통합 활용하기 위해 온-디맨드 자원 가상화 특성을 활용한 오토 스케일링 기법이 사용될 수 있다. 오토 스케일링 기법은 효율적이고 통합적으로 클라우드 자원을 제공한다. 그러나 대부분의 오토 스케일링 기법은 단순한 하드웨어의 성능을 기반으로 제공되고 있어 응용의 특성이나 작업의 데드라인 고려가 필요하다. 사용자가 요청한 작업의 데드라인 내에서 작업 수행이 가능하도록 오토 스케일링을 수행하는 알고리즘을 제안한다. 워크플로우와 Bag of Tasks의 응용 패턴에서 효율적인 자원의 활용을 위해 동적 자원 스케일링 기법을 제시하고, 워크플로우 형태 작업의 단백질 주석 처리 응용과 Bag of Tasks 형태의 변광 지수 계산 응용을 하이브리드 클라우드 환경에서 오토 스케일링 기법에 적용하여 결과를 보인다. 두 가지 응용과 응용의 패턴에 따라 효율적으로 자원이 오토 스케일링 되는 결과를 보임으로써 제안하는 오토 스케일링 기법의 우수성을 검증한다.

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

Abstract The appearance of Science Clouds allows scientists to facilitate large-scale scientific computational experiments over cloud environment. Cloud computing enables applications to employ on-demand and scalable resources dynamically. It is necessary for many task computing (MTC) to provide high performance resources in a long phase and certificate stable executions of applications even dramatic changes of vital status of physical resources. Auto-scaling on virtual machines offers efficient and integrated utilization of cloud resources. VM Auto-scaling schemes have been actively studied as effective resource management in order to utilize large-scale data center in a good shape. However, most of the auto-scaling methods just simply support performance metrics such as CPU utilization and data transfer latency but are rarely aware of execution deadline or characteristics of an application. We propose an auto-scaling method, guaranteeing the execution of an application within deadline. It can handle two types of job patterns; Bag-of-Tasks jobs or workflow jobs. As a proof-of-concept, two applications such as variable index computation and protein annotation workflow

· 이 논문은 2013년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. NRF-2013R1A1A3007866)

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

논문접수 : 2013년 11월 18일
(Received 18 November 2013)
논문수정 : 2014년 4월 8일
(Revised 8 April 2014)
심사완료 : 2014년 5월 12일
(Accepted 12 May 2014)

정보과학회논문지: 시스템 및 이론 제41권 제4호(2014.8)

applications were simulated in hybrid cloud environment. The experimental results of the simulation show the method arrange resources reasonably economically by deadline and adaptively on the change of resource status.

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

1. 서 론

최근 계산 과학 분야에서 자원을 필요한 만큼 빌려 쓸 수 있는 클라우드 기술을 적용하여 과학 클라우드(Science Cloud)를 구축하는 연구가 세계 여러 곳에서 진행되고 있다. 대규모 작업 계산 응용(MTC: Many Task Computing)은 장시간 동안 고성능의 자원을 활용해야 하며 안정적이고 지속적인 자원의 공급이 필수적이다. 이러한 환경에서 성공적인 작업 수행을 위한 실행관리, 자원 관리 등을 제공하는 계산 문제 풀이 환경(Computational Problem Solving Environment)에 대한 연구가 더욱 필요하다. 이러한 문제 풀이 환경에 필요한 클라우드 자원을 효율적으로 통합 활용하기 위해 온-디맨드 자원 가상화 특성을 활용한 오토 스케일링 기법이 사용될 수 있다. 오토 스케일링 기법은 응용의 수행 도중 효율적인 자원 관리를 위해 가상 자원의 수를 동적으로 변화시킨다.

선행 연구인 [1]에서는 하이브리드 클라우드 컴퓨팅 환경에서 효율적인 자원관리를 위한 오토 스케일링 기법을 제안한다. 하지만 Bag of Tasks (BoT)[2] 형태의 작업뿐만 아니라 워크플로우와 같은 다양한 응용패턴을 지원할 수 있도록 알고리즘을 확대할 필요가 있으며, 다양한 응용의 특성을 고려하여 수행이 가능하도록 하고자 한다. 또한 BoT 형태의 다른 응용에 대한 실험을 추가하여 응용의 특성을 고려한 오토 스케일링 기법의 우수성을 보이고자 한다.

본 논문은 다양한 형태의 응용 패턴과 과학 계산 응용의 요구사항을 반영하여 클라우드 컴퓨팅 인프라를 기반으로 자원들을 효율적으로 활용하기 위해 [1]에서 확장한 동적 가상 자원 오토 스케일링 자동화 방법을 제시한다. 특히 BoT 응용뿐 아니라 워크플로우 응용에 대한 체계적인 자원 활용이 가능한 오토 스케일링 기법을 제시한다.

본 논문에서는 BoT 응용을 위한 알고리즘에 대해 설명하고 워크플로우 응용의 특성에 맞는 오토 스케일링 알고리즘을 제안한다. 제안한 오토 스케일링 알고리즘을 워크플로우 형태 작업의 단백질 주석 처리 응용[3]과 BoT 형태의 변광 지수 계산 응용에 적용하여 성능을 검증한다.

본 논문의 구성은 다음과 같다. 1장의 서론에 이어 2장에서는 관련 연구들을 살펴보고, 3장에서는 오토 스케일링 알고리즘에 대해 설명하며 4장에서 적용 시나리오 및 실험에 대해 설명한다. 마지막으로 5장에서 결론을 맺는다.

2. 관련 연구

클라우드의 가상화 기술을 이용한 자원의 효율적인 관리를 위해 오토 스케일링 기법에 대한 연구가 활발히 진행되고 있다. 자원 제공자 입장에서 사용자가 정의한 규칙을 통해 자원의 규모를 확장하고 축소하는 AWS [4]의 "Auto-scaling"과 Windows Azure[5]에서 사용되는 Paraleap[6], Scalr[7]이 자원 할당 자동화 기법을 사용한다. 규칙 기반의 스케일링 기법은 비교적 단순한 방법으로 동적 자원 할당이 가능하지만 복잡한 수행 패턴을 갖는 작업에 대해서는 실제 필요한 만큼 가상머신의 개수가 동적으로 변화하지 않을 수 있기 때문에 데드라인이나 자원 사용 비용의 문제를 야기 할 수 있다.

이러한 이유로 사용자의 입장을 고려하여 데드라인과 사용 비용에 대한 고려를 한 연구로는 [8-12]이 존재한다. 먼저 [8]은 BoT 응용을 통해 자원 사용 비용의 최소화를 위한 오토 스케일링 기법을 제안했다. 자원의 개수를 변화시키는 수평적(Horizontal) 스케일링과 자원의 하드웨어 사양을 조절하는 수직적(Vertical) 스케일링 기법을 혼합 활용했다. 그러나 이 연구에서도 복잡한 수행 패턴의 작업에 대해 실행 중 자원사용량에 대한 고려가 부족하여 동적인 자원 할당이 가능하다고 하기엔 어려움이 있다.

[9]에서는 워크플로우 응용을 가지고 데드라인과 자원 사용 비용을 고려한 알고리즘을 제안했다. [10]에서는 자원 프로비저닝 프로세스 자동화 및 자원 사용 비용 최소화를 고려했다. 다양한 워크로드 형태와 워크플로우 형태를 고려하였으나, 공용 클라우드 단일 환경에서의 자원 사용만을 고려하였다는 한계가 있다. [11], [12]의 경우 그리드 환경에서 워크플로우 응용을 가지고 오토 스케일링 기법을 수행하여 하이브리드 클라우드 인프라 자원 활용에 참고한다. [11]은 사용자가 원하는 데드라인 내에서 실행 비용을 최소화하기 위한 알고리즘을 제안했다. [12]는 상호의존성이 있는 작업에 대해 효율적인 스케일링 알고리즘을 제안했다. [11]과 [12]의 워크플로우 스케줄링을 참고하여 워크플로우 형태의 작업이 수행 될 수 있는 알고리즘을 제안한다.

본 논문에서는 [1]에서 제안한 오토 스케일링 기법을 확장하여 BoT 형태의 작업뿐 아니라 워크플로우와 같

은 다양한 응용패턴과 응용의 특성을 고려하여 적절한 때에 동적 자원 할당이 가능하도록 한다.

3. 오토 스케일링 알고리즘

제안하는 알고리즘은 BoT 형태의 작업과 워크플로우 형태의 작업에서 수행이 가능하다. 알고리즘의 가정과 표기는 [1]을 참고한다. 수식에 필요한 표기는 표 1과 같다.

표 1 알고리즘 수식 표기
Table 1 Notation of the algorithm expression

Notation	Explanation
$i \in \{1, 2, 3, \cdots, x\}$	VM의 번호
$a \in \{1, 2, 3, \cdots, k\}$	작업의 번호
$j_a^i \in \{j_1^i, j_2^i, j_3^i \cdots, j_k^i\}$	VM_i 에서의 a번째 작업
$VM_i \in \{VM_1, VM_2, \cdots, VM_x\}$	모든 자원 중의 하나
$EFT_{VM_i}^{j_a^i}$	Estimated Finish Time of j_a^i
$EST_{VM_i}^{j_a^i}$	Earliest Start Time of j_a^i
$AST_{VM_i}^{j_a^i}$	Actual Start Time of j_a^i
$delay_{VM_i}^{j_a^i} = AST_{VM_i}^{j_a^i} - EST_{VM_i}^{j_a^i}$	작업이 예상 시간보다 늦게 시작된 만큼의 시간
$indelay_{VM_i}^{j_a^i} = EST_{VM_i}^{j_a^i} - AST_{VM_i}^{j_a^i}$	작업이 예상 시간보다 빨리 시작된 만큼의 시간
MIPS (VM_i)	VM_i의 MIPS 값
Length (j_a^i)	작업 j_a^i의 길이
deadline	응용의 데드라인
α	데드라인에 추가로 더해 주는 값
β	parallel 계수
outthreshold	자원의 확장을 결정하는 기준 값
inthreshold	자원의 축소를 결정하는 기준 값

[1]에서 제안한 알고리즘에서 Run-time Scaling 알고리즘은 확장하고, 워크플로우 스케줄링에 적합한 Workflow Scheduling 알고리즘을 제안한다.

알고리즘 1의 Run-time Scaling 알고리즘은 제안하는 오토 스케일링의 전체 알고리즘으로, [1]에서는 두 가지 정책만 존재하였지만 Workflow Scheduling 정책을 추가 하여 세 가지 정책으로 확장한다. SLA(Service Level Agreement)에 명시된 정책대로 각 작업을 스케줄링하게 되는데, 성능지향 정책(line 5), 비용지향정책(line 8), 워크플로우 스케줄링 정책(line 11)의 세 가지 중 하나로 스케줄링 한다. 작업의 형태가 BoT일 경우에는 작업의 크기 순서로 정렬하여 스케줄링 되며, 워크플로우일 경우 작업은 번호 순서대로 스케줄링 된다.

알고리즘 2의 성능지향정책은 BoT의 작업을 가지고 수행한다. 비용 지향 정책을 수반하고 있으며, 작업을 데드라인 내에서 완료 할 수 있도록 최소 성능 요구 이상의 자원만을 이용하여 작업을 수행한다.

새로운 작업 j_a^i 이 VM(Virtual Machine)을 할당 받을 때, 모든 자원의 EFT를 (1)과 같이 계산하여, 작업 j_a^i 이 제일 빨리 시작될 수 있는 VM을 선택한다.

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* → *VMs*}

```
1:  SCALING ← TRUE;
2:  while (true)
3:      if SCALING is TRUE
4:          switch P
5:          case Performance:
6:              Sort waiting jobs in decreasing order of execution length;
7:                  S ← PerformanceOrientedScheduling(sortedJobs, D,
                        minPM);
8:          case Cost:
9:              Sort waiting jobs in decreasing order of execution length;
10:                 S ← CostOrientedScheduling(sortedJobs, D);
11:         case Workflow:
              Sort waiting jobs in sequential order;
12:                 S ← WorkflowScheduling(sortedJobs, D);
13:         each vm where status is running
14:         if no running/waiting jobs on vm then
15:           destroy the vm
16:         send scaling decisions to DRMS
17:         send scheduling decisions to JES
18:         waitForNextInterval();
19:         SCALING ← SLAMonitoring(runningJobs, D);
```

$$EFT_{VM_i}^{j_a^i} = EST_{VM_i}^{j_a^i} + \frac{\text{Length}(j_a^i)}{\text{MIPS}(VM_i)} \quad (1)$$

(2)의 조건처럼, VM_i에서의 k개의 모든 작업의 수행 시간이 데드라인을 넘지 않는지 확인한다(line2, 5).

$$EFT_{VM_i}^{j_k^i} \leq \text{deadline} \quad (2)$$

만약, 작업 j_a^i 에 시작 지연이 발생했을 경우에는 (3)번 식을 이용하여 데드라인을 넘지 않는지 확인한다.

$$EFT_{VM_i}^{j_{a-1}^i} + delay_{VM_i}^{j_a^i} + AST_{VM_i}^{j_a^i} + \frac{\sum_{r=a}^{k} \frac{\text{Length}(j_r^i)}{\text{MIPS}(VM_i)}}{\beta} \leq \text{deadline}$$

$$(\text{If } delay_{VM_i}^{j_a^i} < 0,\ delay_{VM_i}^{j_a^i} = 0) \quad (3)$$

(2)번, (3)번 식 값이 deadline보다 크고 그 값이 outthreshold보다 클 경우, 자원을 확장한다.

$delay_{VM_i}^{j_a^i}$의 값이 양수이고, 기준값(inthreshold)보다 작을 경우 (4), 자원을 축소한다.

$$indelay_{VM_i}^{j_a^i} < \text{inthreshold} \quad (4)$$

Performance-oriented Scheduling 알고리즘에 대한 자세한 내용은 [1]을 참고한다.

Algorithm 2 – Performance-oriented Scheduling

Input –Sorted jobs of the application, deadline *D*, minimum performance requirement *minPM*

Output –Scheduling decision $S = \{jobs \rightarrow VMs\}$, list *toStartUp*.for the VMs to be newly created

1: All *jobs* are scheduled
2: If There is no running *private VM*, find a *private vm* *(PM$_m$ >= minPM)* on which *job* can start the most quickly within the *D;*
3: If There is running *private VM,* schedule *job* to *vm;*
4: Continue with the next *job;*
5: If There is no running *public VM*, find a *public vm* *(PM$_m$ >= minPM)* on which job can start the most quickly within the *D;*
6: If There is running *public VM,* Choose Cheaper *VM* either running *VM or VM* of *ITj* type
7: Schedule *job* to chosen *vm;*
8: Continue with the next *job;*

알고리즘 3은 Workflow Scheduling 알고리즘이다. 제안하는 워크플로우의 스케줄링 기법은 PCH알고리즘[12]를 기반으로 하며, 새로운 클라우드 자원을 확장하거나 축소할 때는 성능 지향 정책을 따르고 있다. PCH 알고리즘[12]를 이용하여 DAG 형태로 정리된 작업의 최상 경로(Critical Path)를 찾고 작업을 그룹으로 나눈 다음, 나누어진 그룹에 따라 작업을 스케줄링 한다. 최상 경로의 작업의 개수가 q개일 때, 사설 VM에서의 최상 경로의 작업의 총 수행시간을 구하여 데드라인으로 설정한다(식 (5)).

$$\text{deadline} = \sum_{r=1}^{q} \frac{\text{Length}\,(j_r^0)}{\text{MIPS}\,(VM_0)} + \alpha \qquad (5)$$

(6)번 식을 만족할 경우, 최상 경로의 작업들의 지연이 발생하지 않아 데드라인 내에서 작업의 스케줄링이 완료 된다(line 2).

$$EFT_{VM_0}^{j_q^0} \leq \text{deadline} \qquad (6)$$

워크플로우 형태의 작업을 할당할 경우 상호 의존성을 고려해야 한다. 상호의존성을 가진 이전 작업의 예상 완료 시간(EFT)을 새롭게 VM을 할당 받게 되는 작업의 가장 빨리 실행될 수 있는 시간(EST)값으로 주어 작업의 순서를 고려하게 된다(line 4-5). 최상 경로내에 있는 작업들을 모두 하나의 클라우드 자원에서 실행하면 통신비용을 줄일 수 있다. 만약, 최상 경로의 작업 중에서 작업 j_a^i 에 지연이 발생할 경우, 다음 작업부터 마지막 작업까지의 총 수행시간이 데드라인 내에 끝나(7)번 식을 만족하는 가장 작은 i값의 새로운 VM을 찾는다.

$$EFT_{VM_0}^{j_{a-1}^0} + delay_{VM_i}^{j_a^i} + \sum_{r=a+1}^{q} \frac{\text{Length}\,(j_r^i)}{\text{MIPS}\,(VM_i)} \leq \text{deadline} \qquad (7)$$

최상 경로 내에 있지 않은 작업들은 상호의존성이 있는 작업이 VM을 할당 받았는지를 확인한 후에 스케줄링 하게 된다(line 6). 자원을 할당 할 때는 사설 클라우드 자원에 우선적으로 할당된다. 최상 경로 내에 있는 작업들을 스케줄링 한 것과 마찬가지로 상호의존성을 고려하여 스케줄링한다. 상호의존성을 여러 개 가진 작업의 경우, 상호의존성이 있는 작업들 중 가장 늦게 끝나는 작업의 예상 수행 완료 시간을 그 작업의 EST 값으로 한다(line 7-10 (사설 클라우드 자원), line 12-16 (공용 클라우드 자원)).

Algorithm 3– Workflow Scheduling

Input –Waiting jobs of the application,

Output –Scheduling decision $S = \{jobs \rightarrow VMs\}$, VM list *toStartUp*.for the VMs to be newly created

1: All *jobs* are scheduled in sequential order
2: *Job* in Critical path is allocated *VM*
3: Find a *vm* on which all *job* in Critical path can run within the *D;*
4: EST_{VM} is related previous *job's* EFT_{VM};
5: Calculate EFT_{VM} by adding EST_{VM} and ET_{VM};
6: *Job* which has allocated related previous *jobs* on *vm*
7: If There is no running *private VM*, find a *private vm* *(PM$_m$ >= minPM)* on which *job* can start the most quickly within the *D;*
8: EST_{VM} is related previous *job's* EFT_{VM};
9: Calculate EFT_{VM} by adding EST_{VM} and ET_{VM};
10: If There is running *private VM,* schedule *job* to *vm;*
11: Continue with the next *job;*
12: If There is no running *public VM*, find a *public vm* *(PM$_m$ >= minPM)* on which job can start the most quickly within the *D;*
13: EST_{VM} is related previous *job's* EFT_{VM};
14: Calculate EFT_{VM} by adding EST_{VM} and ET_{VM};
15: If There is running *public VM,* Choose Cheaper *VM* either running *VM or VM* of *ITj* type
16: Schedule *job* to chosen *vm;*
17: Continue with the next *job;*

4. 오토 스케일링 기법 적용 실험

CloudSim[13]을 이용하여 워크플로우 형식의 단백질 주석 처리 응용, BoT 형식의 변광 지수 계산응용을 대상으로 시뮬레이션을 수행한다. 사설 클라우드와 공용 클라우드를 사용하는 하이브리드 클라우드 환경에서 오토 스케일링 알고리즘을 수행한다.

4.1 단백질 주석 처리 응용 시나리오 및 실험

워크플로우 형식의 단백질 주석 처리 응용을 대상으로 시나리오와 실험을 수행한다. 단백질 주석 처리 응용은 상호의존성을 가진 워크플로우 형태의 작업을 가지며 대량의 정보 처리가 필요하다. 본 실험에서는 참고한 논문[11]과 비교하기 위해서 작업의 실행시간을 유사하게 만들어내는 작업의 사설 클라우드 환경을 구축하여

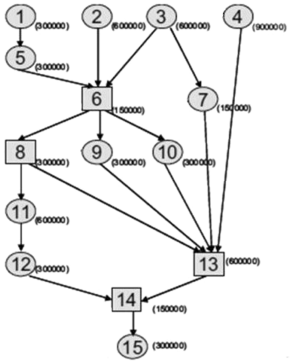

그림 1 단백질 주석 처리 워크플로우 응용 구조
Fig. 1 protein annotation workflow

같은 응용을 가지고 수행하며 공용 클라우드 환경은 아마존 EC2를 기반으로 [1] 논문을 참고로 구축하여 실험을 진행한다.

그림 1은 DAG 형식의 단백질 주석 처리 워크플로우 응용의 워크플로우 형태이며, 작업의 개수는 15개이고 작업 길이는 그림 1의 괄호에 나와 있는 MI로 수행한다.

작업들은 번호 순서대로 수행된다. 사설 클라우드 자원에서 최대한 작업을 수행하도록 하고, 데드라인 내에서 수행을 못할 경우 공용 클라우드 자원을 구매하여 작업을 수행한다. 최상 경로내의 작업인 1, 5, 6, 8, 11, 12, 14, 15번 각 작업들의 EST값을 계산하여 사설 클라우드 자원에 할당된다.

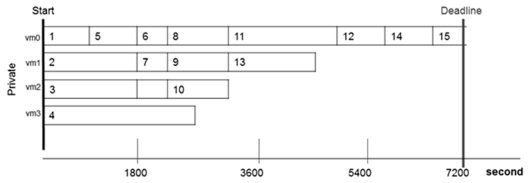

그림 2 오토 스케일링 결과
Fig. 2 Result of auto-scaling

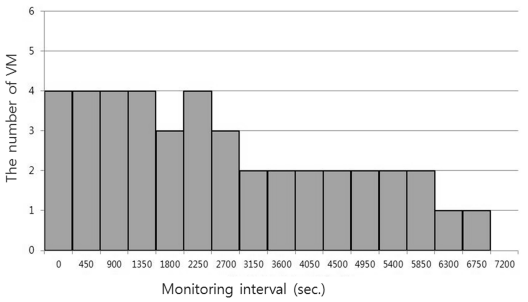

그림 3 VM의 개수
Fig. 3 The number of VM

2, 3, 4번 작업의 경우 최상 경로에 속하지 않고, 상호의존성이 있는 이전의 작업들이 없으므로 평행하게 할당된다. 6번 작업의 경우 2, 3, 5번 작업들의 EFT 값을 확인하고, 작업을 할당한다. 8, 9, 10번 작업의 경우 vm2에서 1800초부터 수행될 수 있는데, 상호의존성을 가지는 6번 작업이 끝나지 않아 VM을 할당 받을 수 없다. 따라서 vm2에 suspend되는 부분이 발생한다. 4번 작업이 vm3에서 끝나고 나면 뒤에 다른 작업을 할당하지 않기 때문에 자원의 축소인 스케일 인이 발생한다. 13번 작업은 상호의존성을 가지고 있는 작업들 중에서 가장 늦게 끝나는 10번 작업의 수행이 끝난 후 자원을 할당 받아 작업을 수행한다. 마지막으로 15번 작업이 데드라인 내에서 수행을 마치면서 모든 작업이 완료된다.

그림 3은 제안하는 Workflow Scheduling 알고리즘을 기반으로 그림 1을 가지고 오토 스케일링 기법을 수행한 결과로 VM의 개수 변화를 나타낸다. 그림2는 수행되는 작업을 나타낸다.

그림 4는 그림 1의 단백질 주석 워크플로우 응용의 워크플로우 형태 4개를 가로로 확장한 형태이다. 작업의 개수는 57개이고, 작업 길이는 그림 4의 괄호에 나와 있는 MI로 수행한다.

그림 4의 워크플로우 형태를 통해 오토 스케일링 기법을 수행한 결과가 그림 5에 나타난다. 그림 5는 Workflow Scheduling 알고리즘을 기반으로 하여 수행시간에 따른 클라우드 자원 개수의 변화를 나타낸 그래

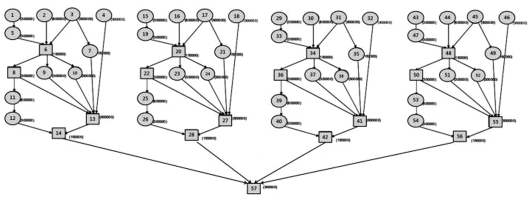

그림 4 확장된 단백질 주석 처리 워크플로우 응용 구조
Fig. 4 Extension of protein annotation workflow

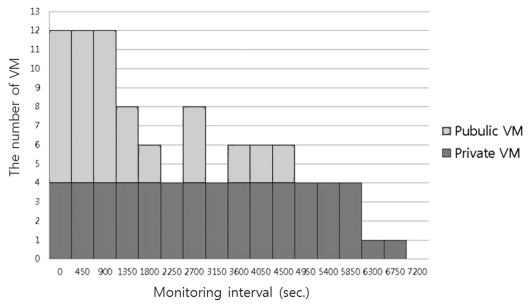

그림 5 VM의 개수
Fig. 5 The number of VM

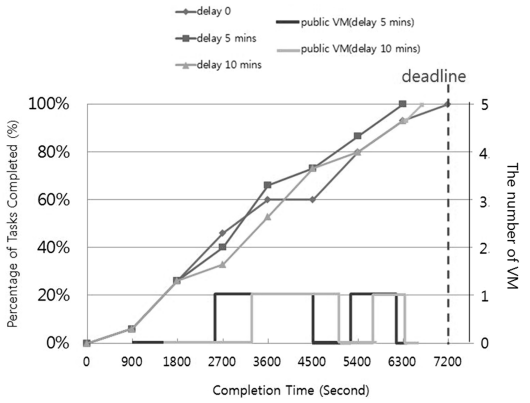

그림 6 지연을 고려한 워크플로우 스케줄링
Fig. 6 Workflow Scheduling with delay

프이다. 앞서 수행한 실험과 같은 환경에서 실험을 진행하였다. 본 실험에서 데드라인은 그림 4의 최상 경로에 속하는 모든 작업들의 예상 수행 완료 시간으로 7200초이며, 모니터링 인터벌은 450초이다. 수행 시간에 따른 그래프를 보면 클라우드 자원의 사용 개수가 동적으로 변화하는 스케일 인과 스케일 아웃의 형태가 나타난다. 그래프를 통해 워크플로우 형태 작업의 상호의존성이 클라우드 자원 사용에 영향을 미침을 알 수 있다. 그림 5를 통해 본 논문에서 제안하는 오토 스케일링 기법이 워크플로우 형태의 작업에서도 성공적으로 수행한 것을 볼 수 있다.

그림 6은 Workflow Scheduling 알고리즘을 기반으로 지연이 발생한 단백질 주석 처리 워크플로우 응용을 수행한 결과이다. 단백질 주석 처리 워크플로우 응용 구조는 그림 1에 나타나 있으며, 그림 6은 수행시간에 따른 작업의 완성 정도를 나타낸다. 지연에 의해 수행시간에 따른 작업의 완성도가 다르게 나타나는 것을 볼 수 있다. 지연이 발생하지 않은 경우 데드라인은 7200초이며, 사설 클라우드 자원에서만 작업을 수행한다. 본 실험에서는 [11]을 참고하여 6번 작업에 5분과 10분의 지연을 주어 시뮬레이션을 수행한다. 지연이 발생하면 데드라인 내에서 작업이 끝날 수 있도록 리스케줄링이 발생한다. 리스케줄링을 통해 8번~15번의 최상 경로에 있는 작업이 공용 클라우드 자원에서 실행되며, 데드라인을 초과하지 않고 작업이 완료된다. 성능이 더 좋은 공용 클라우드 자원을 사용하여 데드라인보다 빠른 시간에 작업이 완료되지만 시간 당 비용을 지불해야 한다.

4.2 변광 지수 계산 실험

BoT 형식의 변광 지수 계산 응용을 대상으로 시뮬레이션을 수행한다. 성능지향정책 알고리즘을 기반으로 실험을 진행한다. 변광 지수 계산 응용은 수집된 데이터의

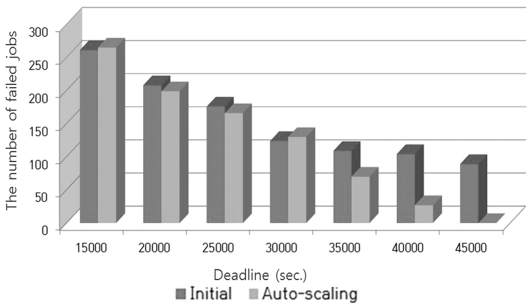

그림 7 데드라인 별 수행 실패한 작업 개수 비교
Fig. 7 The number of Virtual Machine by each deadline

양이 1600만개로 매우 크기 때문에 작업의 크기 또한 크며, 작업들은 BoT 형태이다. 또한 파라미터에 따라 작업의 개수가 결정된다는 특성을 가진다. 시간에 따라 밝기가 변하는 별의 변광 현상은 별들의 물리적인 특성을 이해함에 있어서 중요한 자료가 된다. 본 논문에서는 영국의 SuperWASP 프로젝트 [14]의 관측 자료를 바탕으로 분석된 데이터를 활용하여 실험을 진행한다. 본 실험에 사용된 자원의 사양은 [1]과 동일하게 한다.

그림 7은 데드라인이 변화함에 따라 초기 스케줄링과 오토 스케줄링 기법에 대해 실패한 작업의 개수의 비교를 나타낸 결과이다. 본 실험에서는 성능 지향 스케줄링 정책을 통해 변광 지수 계산 응용을 수행한다. 초기 스케줄링은 성능 지향 정책에 기반을 두고 작업을 수행하는 것이며, 오토 스케일링 기법은 초기 스케줄링을 마친 후 일정 간격으로 모니터링을 수행함으로써 데드라인 위반 가능성이 발견되는 작업에 대해 리스케줄링을 수행한다. 그림 6에 나타난 것과 같이 초기 스케줄링과 오토 스케일링 기법 모두 데드라인이 증가할수록 실패하는 작업의 수가 감소한다. 초기 스케줄링은 지연이 발생하여도 대응을 할 수 없으나 오토 스케줄링의 경우 리스케줄링을 통해 작업 수행 동안 발생하는 지연에 대처할 수 있기 때문에 상대적으로 오토 스케일링 기법이 초기 스케줄링보다 작업 실패율이 낮다.

그림 8은 오토 스케일링 기법을 사용하여 시간의 경과에 따라 자원이 확장되는 정도를 나타낸 그래프이다. 본 실험에서 작업의 개수는 1060개로 설정하였고 그 중 118개의 작업에 지연을 주었다. 데드라인은 140000초로 설정하였으며 모니터링은 5000초마다 수행되었다. 오토 스케일링 기법을 통해 작업 수행 중 지연이 발생하여 데드라인 위반 가능성이 발견되었을 때 필요한 자원을 동적으로 할당함으로써 데드라인 위반율을 낮춘다. 그래프를 보면 초기 실행부터 70000초까지는 VM을 15개만 수행시키고 그 후부터는 자원을 점차적으로 확장시킨다. 115000초부터는 더 이상 데드라인 위반 가능성이 발견

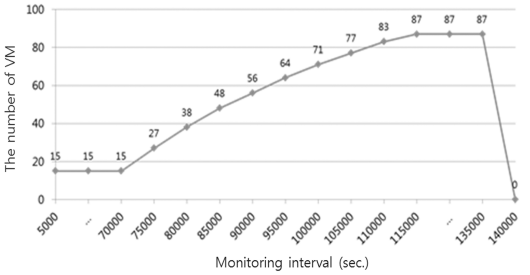

그림 8 지연이 발생한 경우, VM의 개수 변화

Fig. 8 A changes in the number of Virtual Machine when delay occurs

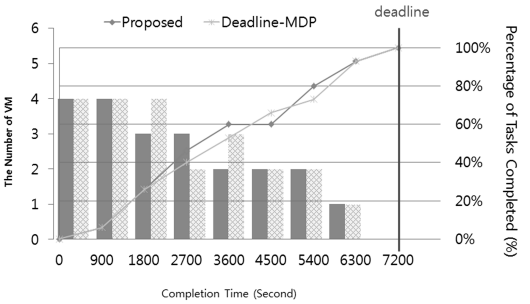

그림 9 Deadline-MDP와 제안한 Workflow Scheduling 비교

Fig. 9 Proposed Workflow Scheduling and Deadline-MDP

되지 않기 때문에 자원의 확장을 멈추고, 1400000초에서 모든 작업의 수행을 완료한다. BoT 형식의 변광 지수 계산 응용을 대상으로 시뮬레이션을 수행한 결과 데드라인 내에서 VM이 필요에 따라 동적으로 할당되어 작업이 수행되었다.

4.3 실험 결과 분석

응용의 특성을 고려하여 제안한 오토 스케일링 기법은 응용의 특성에 맞게 자원을 필요한 시기에 적절하게 동적으로 할당하며 프로비저닝 서비스를 제공하고 있다. 변광 지수 계산 응용의 경우 파라미터 값을 바꾸면서 반복적으로 실행이 되는 응용으로 과거 실험의 결과가 중요하여 프로비저닝 서비스가 필요하다. 작업의 패턴이 일정하고, 작업의 실행 개수는 파라미터 개수에 직접적인 영향이 있어 전체 수행 시간을 예측하는 것이 가능하여 변광 지수 응용에서 프로비저닝 서비스가 가능하게 된다. 제안한 오토 스케일링 기법으로 실험한 결과 효율적인 프로비저닝 서비스가 제공된 것을 알 수 있다. [10]에서는 다양한 워크로드 패턴을 고려하여 오토 스케일링을 수행하였으나 수행 중에 패턴이 달라지거나 복합적인 패턴을 갖는 응용의 경우 제대로 된 오토 스케일링이 수행 되지 못할 수 있으므로 응용의 특성을 고려하는 것이 필요하다. 응용의 특성을 고려하여 스케줄링을 하는 것이 효율적인 자원의 활용을 가능하게 한다.

제안하는 오토 스케일링 기법은 지연에 대처하여 사용자가 요구하는 데드라인 내에서 작업을 모두 수행한다. [11]은 워크플로우 형태의 작업에서 지연이 발생하였을 때 데드라인 내에서 작업들이 완료될 수 있도록 실행 중에 모니터링과 리스케줄링을 수행한다. 그러나 [11]이 워크플로우 형태의 작업만을 가지고 지연에 대한 고려하였다. [11]과 제안하는 Workflow Scheduling 기법을 사설 클라우드 환경에서 그림1에 나타난 워크플로우 구조를 가지고 비교하였다.

사설 클라우드 환경에서 [11]과 제안하는 오토 스케일링 기법을 비교하여 그림 9에 나타내었다. 제안하는 Workflow Scheduling 기법이 초반에 작업을 빨리 완료하는 모습을 볼 수 있다. 두 기법 모두 데드라인 내에서 작업들이 완료될 수 있도록 필요한 시기에 적절하게 VM을 할당하여 작업이 수행되도록 하고 있다.

본 논문에서는 워크플로우 형태의 작업뿐만 아니라 BoT 형태의 작업의 응용에서도 지연을 대처하여 오토 스케일링 기법이 효율적으로 수행이 되는 것을 보이고 있다.

5. 결론 및 향후 연구

본 논문에서는 하이브리드 클라우드 환경에서 BoT 형태의 작업과 워크플로우 형태의 작업들이 수행 가능한 오토 스케일링 기법을 제안하였고, 2가지 응용을 적용해서 시뮬레이션을 수행하였다. BoT 형태의 변광 지수 계산 응용의 시뮬레이션에서는 데드라인에 따라서 초기 스케줄링의 작업 실패율이 오토 스케일링의 작업 실패율보다 큰 것을 볼 수 있었고, 발생하는 지연에 대처하여 자원의 개수를 동적으로 추가하는 것도 확인하였다. 단백질 주석 처리 워크플로우 응용의 시뮬레이션의 경우, 상호의존성에 의해서 데드라인 내에서 자원의 스케일 인, 스케일 아웃 형태가 나타났다. 워크플로우 형태의 작업에서도 오토 스케일링 기법이 적절히 적용되어 동적 자원 할당이 가능하였고 지연에도 적절히 대처하였다. 제안한 오토 스케일링 기법을 통해 두 가지 응용 패턴이 다른 계산과학분야의 응용을 대상으로 시뮬레이션 한 결과, 필요에 따라 자원이 효율적으로 사용된 것을 알 수 있다.

향후에는 응용의 특성을 고려한 의미론적 스케일링과 저전력 컴퓨팅 기술 활용한 에코 스케일링 기법을 추가할 예정이다.

References

[1] Hyejeong Kang, Jung-in Koh, Yoonhee Kim, "A

SLA-based VM Auto-Scaling Method in Hybrid Cloud Computing for Scientific Computational Applications," *Journal of KIISE accepted*, 2013. (in Korean)
[2] W. Cirne, F. Brasileiro, J. Sauv?, Na. Andrade, D. Paranhos, E. Santos-Neto, R. Medeiros, "Grid Computing for Bag of Jobs Applications," *Proceedings of the 3rd IFIP Conference on E-Commerce*, E-Business and E-Government, Sep. 2003.
[3] A. O'Brien, S. Newhouse and J. Darlington, "Mapping of Scientific Workflow within the e-Protein project to Distributed Resources," In *UK e-Science All Hands Meeting*, Nottingham, UK, Sep. 2004.
[4] Amazon Web Service, http://aws.amazon.com/
[5] Windows Azure, http://www.windowsazure.com/
[6] Paraleap, https://www.paraleap.com/
[7] Scalr, http://scalr.com/
[8] S. Dutta, S. Gera, A. Vermam, and B. Viswanathan, "Smartscale: Automatic application scaling in enterprise clouds," in *5th IEEE International Conference on Cloud Computing (CLOUD)*, pp.221-228, Jun. 2012.
[9] L. Bittencourt, and E. Maderia, "HCOC: A Cost Optimization Algorithm For Workflow Scheduling in Hybrid clouds," *Journal of Internet Services and Applications*, vol.2, Springer-Verlag, pp.207-227, Dec. 2011.
[10] M. Mao, and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, Nov. 12-18, 2011.
[11] J. Yu, R. Buyya, and C. K. Tham, "Cost-based scheduling of scientific workflow applications on utility grids," *e-Science and Grid Computing, 2005 First International Conference on IEEE*, pp.8-147, 2005.
[12] L. F. Bittencourt, and E. R. Madeira, "A performance oriented adaptive scheduler for dependent tasks on grids," *Concurrency and Computation: Practice and Experience*, vol.20, Issue. 9, pp.1029-1049, 2008.
[13] 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.
[14] SuperWASP, http://www.superwasp.org/

안 윤 선
2013년 숙명여자대학교 컴퓨터과학부(학사). 2013년~현재 숙명여자대학교 컴퓨터과학부 석사 과정. 관심분야는 클라우드 컴퓨팅 환경, 온톨로지, 지능형 시스템

정 솔
2010년~현재 숙명여자대학교 컴퓨터과학부 학사 과정. 관심분야는 클라우드 컴퓨팅 환경

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