■ 2013년도 학생논문 경진대회 수상작

하이브리드 클라우드에서 과학 응용을 위한 정책 기반의 적응형 작업 스케줄링

(Adaptive Policy-based Task Scheduling for Scientific Applications in Hybrid Cloud)

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

요 약 방대한 계산 처리를 필요로 하는 과학 응용은 그리드를 포함한 다양한 계산 자원을 수년간 이용해왔다. 하지만 기존의 계산자원만으로 과학 응용의 가변적인 작업 부하 및 동적 자원 요구를 충족시키기에는 어려움이 있기 때문에 필요에 따라 자원을 빌려 쓸 수 있는 클라우드 자원으로의 확장이 요구된다. 또한 클라우드 자원 이용을 통해 발생하는 비용, 성능 오버헤드 등의 고려를 통해 응용의 특성 및 서비스 수준 협약(Service Level Agreement)를 보장하는 동적인 자원 할당이 가능한 작업 스케줄링 기법이 요구된다. 본 논문에서는 하이브리드 클라우드 자원의 이용 효율을 향상시키고, 주어진 데드라인 안에서 응용 수행을 마칠 수 있도록 수행 중 필요에 따른 작업 재배치를 위해 온디맨드로 다중 인프라 자원 할당이 가능한 정책 기반의 적응형 작업 스케줄링 기법을 제안하고자 한다. 실험을 통하여 정책 조건을 만족하면서 주어진 데드라인 내에서 다중 인프라 자원을 충분히 활용하여 작업을 수행하는 결과를 보임으로써 제안하는 기법의 우수성을 검증하였다.

키워드: 적응형 스케줄링, 하이브리드 클라우드, 정책 기반 스케줄링, 계산 과학 응용

Abstract As many scientific workflows require high computing power to fulfill their goals in a reasonable time period, diverse resources such as Grids have been considered for several years. However, the existing computing resources are difficult to meet dynamic resource demands caused by variable workload of recent scientific applications. As Cloud offers on-demand resources for computation-intensive applications which require large-scale computing power in certain time period, hybrid cloud capable of scaling out multi-infrastructure more easily satisfies the need. In a hybrid cloud, task scheduling is one of difficult problems to provide good matching between diverse resources and the requirement of application characteristics. Particularly, a scheduling policy, which satisfies user's SLA (Service Level Agreement) such as performance or cost, can be the first priority to achieve the goal of application execution. In this paper, we propose a task scheduling service that provides on-demand multiple infrastructures by applying policies to execute an application in a given deadline. The goal of the task scheduling service is maximizing the utility of multiple infrastructures and providing an adaptive scheduling which can change the initial scheduling plan during execution to get better results. We have evaluated the performance of our scheduling service. The result shows that with deadline constraints the service can enhance the utilization of various infrastructures under policies.

Keywords: Adaptive scheduling, hybrid cloud, policy-based scheduling, scientific application

· 본 연구는 숙명여자대학교 2013학년도 교내연구비 지원에 의해서 수행되었음

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

논문접수 : 2013년 5월 22일
심사완료 : 2013년 9월 9일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제19권 제11호(2013.11)

1. 서 론

정보화 사회가 고도화됨에 따라 고성능 컴퓨터 등의 각종 IT계산자원의 수요가 급격히 증가하면서 더욱 높은 계산능력을 가진 자원을 확보하려는 노력이 증가하고 있다. 그 중에서 그리드 컴퓨팅 기술은 지역적으로 분산된 이기종 계산 자원과 대용량 저장 장치, 다양한 과학 기술 연구 장비 등을 통합하여 과학기술 및 정보 서비스를 제공하기 때문에 많은 계산과학 분야에서 이용되고 있다. 하지만 현대 응용의 동적인 작업 부하로 인해 응용 수행에 필요한 계산 자원의 수요가 가변적이고 이에 따라 필요한 양을 유동적으로 제공해야 할 필요가 있다. 그리드 자원의 경우 적절한 양을 탄력적으로 서비스하기에는 어려움이 따른다. 반면, IT분야에서 새로운 패러다임으로 떠오르는 클라우드 컴퓨팅은 각종 컴퓨팅 자원들을 가상화 기술로 통합하여 필요한 양만큼 이용 가능한 IT환경을 제공하는 기술로서 기존 계산 자원들과는 달리 자원 사용의 유연성을 제공한다. 그러나 단순히 클라우드 자원만을 사용하게 될 경우 기존 자원의 낭비가 발생할 수 있으며, 가상화 기술을 기반으로 하하는 클라우드 컴퓨팅의 특성 상, 성능면에서 추가적인 오버헤드가 발생할 수 있다. 따라서 각 분산 자원의 서로 다른 특성을 파악하고 높은 계산 성능을 위해 다양한 분산 컴퓨팅 자원을 포함하는 하이브리드 클라우드에 관한 효율적인 자원 관리 및 작업 스케줄링 기법이 반드시 필요하다. 특히, Amazon EC2[1], Windows Azure[2], Rackspace[3]와 같은 상용 클라우드 컴퓨팅의 경우 사용시간에 비례하여 비용이 증가하기 때문에 자원 사용에 따른 비용 발생 여부 및 비용 발생량이 핵심적인 고려사항이 될 수 있다.

본 논문에서는 슈퍼컴퓨팅, 그리드, 클라우드 컴퓨팅과 같은 분산 컴퓨팅 인프라들을 기반으로 하는 이기종 자원을 효율적으로 활용하기 위한 적응형 작업 스케줄링 기법을 제안한다. 또한 다양한 이기종 자원으로 인해 사용자의 응용 수행 요구가 복잡해지면서 사용자와 제공자간의 서비스 수준 협약(SLA: Service Level Agreement)도 복잡해졌다. 이러한 SLA를 충족시키기 위해서 계산 과학 응용의 특성과 자원의 속성을 고려한 성능 및 비용 정책을 수립하고, 정책을 수용할 수 있는 데드라인 기반의 동적 작업 스케줄링 기법에 대한 연구를 수행하고자 한다. 전산 유체 역학 분야(CFD; Computational Fluid Dynamics)[4]의 응용을 대상으로 제안한 기법의 성능 측정 및 비교/분석을 위해서 스케줄링 기법에 대한 실험을 수행하여 제안하는 기법의 우수성을 검증하고자 한다.

본 논문의 구성은 다음과 같다. 1장의 서론에 이어 2장에서는 관련 연구들을 살펴보고, 3장에서는 본 논문에서 제안하는 스케줄링 서비스를 적응형 스케줄링 방식과 정책 기반의 스케줄링 방식으로 설명한다. 4장에서는 실험 환경 및 다양한 실험 결과에 대해 살펴본다. 마지막으로 5장에서 결론을 도출한다.

2. 관련 연구

여러 가지 자원들을 복합적으로 이용할 경우 다양한 정책들을 고려하여 작업들을 효율적으로 클라우드 자원에 스케줄링 하기 위한 연구가 활발하게 진행되고 있다.

Bossche R, et al.[5]은 그리드와 클라우드 자원을 통합한 하이브리드 클라우드 환경에서 데드라인 내에 계산 비용 및 데이터 전송 비용을 고려하여 가장 효율적인 자원을 선택할 수 있도록 하는 요소들을 확률적 성능 평가 모델을 통해 구하고 수식으로 표현하였다. 응용은 BoT (Bag of Tasks) 워크플로우 모델로 정의하였고 자원은 GoGrid[6]와 Rackspace, Amazon EC2를 실제 속성(인스턴스 가격, 네트워크 등)에 따라 정의하여 시뮬레이션을 통해 비용 효율적인 스케줄링 시 고려해야 할 인자들에 대해 정의하였다. 그러나 이 연구에서 제안한 기법을 활용하기 위해서는 충분한 이력 데이터가 쌓여야 의미가 있으며, 해당 논문에서도 이력으로 쓸 데이터가 부족하여 확률적 모델을 통한 성능 평가를 수행하였다.

HCOC[7]는 워크플로우 응용의 데드라인을 고려한 자원 대여 및 수행 비용 최적화 알고리즘을 제안한다. 데드라인을 워크플로우의 CP(Critical Path)로 설정한다. CP는 최고 사양의 자원 이용 시 응용의 수행 시간을 의미하며 이상적인 데드라인 내에 작업 수행을 보장하기 위한 동적 스케줄링이 가능하다. 그러나 BoT 속성을 가진 응용에 대해서는 데드라인을 CP로 설정하기에는 무리가 있다.

VGrADS[8]는 시간적 요소에 대한 고려가 필수적으로 수반되는 응용의 수행을 위해 스케줄링 정책 및 인프라 구축을 목표로 진행되는 프로젝트이다. Tera-Grid[9], Amazon EC2, 로컬 클러스터와 Eucalyptus[10](사설 클라우드)로 구성된 그리드와 클라우드 기술의 협동 사용으로 사이언스 워크플로우의 수행 성능을 균형 있게 하며 안정성, 비용 고려 방식을 계획하는 새로운 워크플로우를 가능하게 하는 것을 목표로 한다. 하지만 Fault-tolerance를 지원하는 가상 자원 프로비저닝만을 제공하며, 그 밖의 정책(데드라인, 비용 등)을 고려한 프로비저닝 연구는 여전히 진행 중이고 공개된 연구 결과는 존재하지 않는다.

Aneka[11] 프로젝트에서는 데드라인을 우선순위로 고려하는 프로비저닝을 연구한다. 이 시스템은 스케줄러 서비스와 프로비저닝 서비스로 이루어져 있으며, 스케줄러 서비스가 작업들의 평균 실행 시간과 데드라인을 고려해서 자원의 양을 결정하면 프로비저닝 서비스가 해당 자원을 물리적으로 할당하고, 스케줄러 서비스에서 불필

요하다고 결정된 자원은 프로비저닝 서비스에서 회수하는 방식으로 되어 있어 서로 다른 자원들에 대한 유연한 관리를 제공한다. 또한 데드라인 정책을 위한 다양한 프로그래밍 모델들을 지원한다. 그러나 데드라인 설정 시에 가정치를 이용하여 신뢰도가 떨어지며 데드라인을 정의할 때에 수행할 응용 특성은 전혀 고려하지 않는다.

Comet Cloud[12]는 응용과 인프라 구조를 계층별로 구분하여 이 계층들을 고려한 프로비저닝을 제공한다. 이 인프라 구조의 계층에 대한 고려는 데드라인 정책에 따라 제공할 자원의 종류를 달리하여 자원의 활용도를 높이며, 응용 계층을 함께 고려하여 주어진 예산과 시간의 우선순위에 따라 사이언스 워크플로우 수행을 위한 최적의 자원 통합사용 방법을 제안한다. 하지만 작업 스케줄링 시에 데드라인 또는 비용만을 각각 고려하여, 데드라인 고려 시에는 비용 면에서 최고치를 기록하고 비용을 고려하는 경우에는 데드라인을 훨씬 능가(즉, 수행시간 증가)하는 편향적인 성능 결과를 보인다.

3. 하이브리드 클라우드를 활용한 정책 기반의 적응형 작업 스케줄링 기법

3.1 대상 응용

계산과학 응용 중 다양한 분야에서 활용되는 전산 유체 역학(CFD) 분야의 응용은 입력파일을 기반으로 수치해석이 이루어지며, 이 해석단계에서 많은 계산자원을 필요로 한다. 해석 단계 이전에 해석 조건과 범위를 다양한 파라미터 값을 통해 설정하여 입력 값이 다른 다수의 작업을 동적으로 동시에 생성하여 BoT(Bag of Tasks) 형태의 응용 형태를 갖는다. 해석 조건에 따라 작업의 수와 수행 시간이 달라지는 HTC(High Throughput Computing)를 수행하게 된다.

그림 1은 이러한 CFD 응용의 수행 과정을 도식화한 것으로 사용자가 원하는 입력파일과 실행파일을 포함하는 작업을 계산 자원에 제출하였을 때, 그 작업은 사용자가 정의한 입력 값의 범위에 따라 많게는 수 만개의 작업으로 나뉘어져 각각 수행 결과를 도출하게 된다.

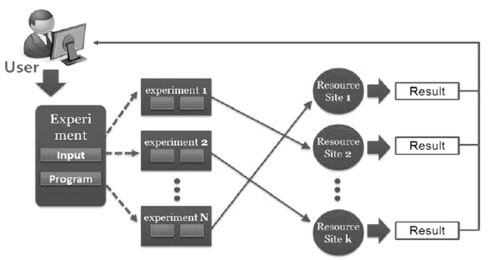


그림 1 계산과학 응용 수행 과정
Fig. 1 Execution Procedure of Scientific Application

BoT 응용의 특성상 동시에 많은 작업을 수행하는 것이 전체 응용 시간의 단축으로 이어질 수 있다. 그러나 기존 그리드 자원은 그 자원의 수가 한정되어 있고 각각의 머신의 사양이 다양하여 작업이 저사양의 자원에 할당될 경우 사용자가 작업 수행을 완료하기 원하는 시간을 지키지 못하는 경우가 발생할 수 있다. 따라서 기존 자원을 최대한 활용하되 클라우드 자원의 차용을 통해 데드라인과 같은 사용자 요구조건과 그 안에서 발생할 수 있는 비용에 대한 고려가 가능한 스케줄링 기법 개발이 필요하다.

3.2 적응형 작업 스케줄링 기법

본 논문에서는 하이브리드 클라우드를 활용한 적응형 작업 스케줄링 기법에 대해 모델링하였다. 일반적으로 작업 수행에 대한 예측이 힘들기 때문에 초기 스케줄링에 차질이 발생할 수 있으므로, 이를 방지하기 위한 주기적인 자원 모니터링을 통해 작업의 지연을 검사하고 발생한 지연이 응용 전체 수행의 데드라인을 위반할 가능성이 발견되면 대기 중인 작업을 재스케줄링한다. 이 때, 재스케줄링 되는 작업들은 데드라인 안에 수행을 마칠 수 있는 방향으로 적정 자원을 할당 받는데, 필요에 따라 상용 클라우드에 관한 추가 확장이 가능하다. 그림 2는 적응형 작업 스케줄링 기법을 적용한 시나리오를 나타낸다. 각각의 세로 열은 하나의 자원에 대응된다. G1, G2의 경우는 그리드 자원의 일부, P1, P2는 사설 클라우드에 해당하는 비-상용 자원들을 나타낸다. 그림 2의 (a)는 상기한 자원들에 대한 초기 스케줄링 결과로서 각 열에 나타난 숫자는 작업 번호를 의미하고, 길이는 작업 수행 시간을 의미한다. 그림 2의 (b)는 작업 지연 발생으로 인한 잠재적 응용의 데드라인 위반 위험성을 예방하기 위한 적응형 스케줄링 기법을 보인다. 제안하는 적응형 작업 스케줄링 기법은 응용의 수행 시간이 데드라인을 초과하지 않도록 작업을 자원에 재할당한다. 또한 기존 자원만으로 응용 수행이 어려울 경우, 온디맨드 서비스를 제공하는 상용 클라우드를 이용한다.

그림 2에서 응용의 데드라인 D는 22시간이고, 한 응용 수행에 속하는 작업이 각각 10시간(작업번호: 1,2,3,4), 5시간(작업번호: 5,6,7,8), 2시간(작업번호: 9,10,11,12), 1시간(작업번호: 13,14,15,16,17,18)이 소요된다고 가정한다. 제안하는 스케줄링 기법을 적용한 수행은 다음과 같다.

1) 작업의 수행시간이 긴 순서대로 작업을 정렬 시킨 후 그리드-사설 클라우드-상용 클라우드 순으로 작업 수행 여부를 판단한 후, 해당 작업을 적정 자원에 스케줄링한다. 주어진 데드라인을 만족하는 응용의 작업에 대한 초기 스케줄링이 그림 2(a)이다.

2) 13시간 경과 후(그림 2(b)에서 점선부분), 선행된 작업(작업번호: 1,2)의 예상치 못한 지연으로 인해 후반 작업들의 수행이 응용의 데드라인을 위반할 가능성이 높아졌다.

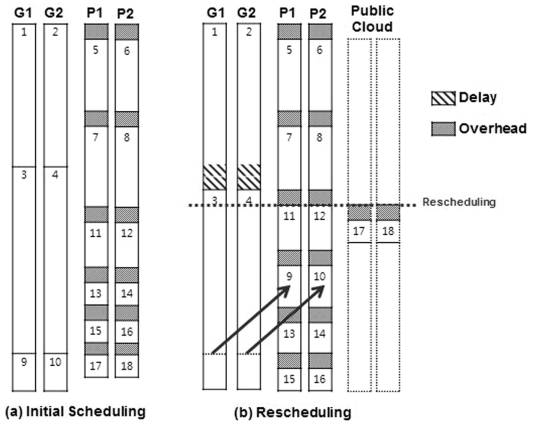


그림 2 적응형 작업 스케줄링 과정
Fig. 2 Adaptive Scheduling to Prevent Deadline Violation

3) 자원 모니터링을 통해서 정해진 주기마다 작업 진행 정도를 측정하여 예상보다 실제 작업의 수행이 늦어진 경우 응용의 데드라인 위반 가능성을 확인한다. 잠재적으로 데드라인 위반 가능성이 있다면 현시점에서 대기 중인 작업들을 적응형 스케줄링을 통해 데드라인에 맞게 재배치한다.

4) 이 때, 보유 중인 자원만으로 데드라인 안에 작업의 수행이 어렵다면 그림 2(b)와 같이 기존 자원의 이용을 최대화한 후, 필요한 만큼 자원을 추가로 상용 클라우드에서 빌려 쓴다. 길이가 상대적으로 짧은 작업에 상용 클라우드 자원을 할당함으로써 비용 절감을 추구한다.

3.3 정책 기반의 적응형 작업 스케줄링 알고리즘

하이브리드 인프라를 공유하여 효율적으로 통합 활용하기 위해서는 응용의 속성, 시간 또는 비용 등의 정책을 수용할 수 있는 하이브리드 자원 상의 동적인 작업 스케줄링 기법이 요구된다. 본 논문에서는 제안하는 스케줄링 서비스는 성능 또는 비용 위주의 스케줄링 정책에 기반을 둔다. 두 정책 모두 사용자가 원하는 데드라인을 응용 수행의 척도로 하고, 이를 위한 작업 스케줄링 변경이 가능하다. 성능 지향 정책의 경우 데드라인 내에 응용 수행을 완료하되 상용 클라우드를 포함한 보유 중인 자원을 최대한 활용하여 작업 수행 시간을 단축하는 성능 향상을 목표로 작업을 스케줄링한다. 따라서 경제적 측면인 비용 절감 보다 성능 향상이 우선적인 고려사항이 된다. 비용 지향 정책은 이와 반대로 데드라인 내의 성공을 보장하되 상용 클라우드 소비에 따른 비용을 최소화하기 위한 스케줄링이다. 즉, 성능 향상 측면보다 경제적 측면을 우선시하는 경우가 이에 해당된다.

본 논문에서 제안하는 정책 기반의 적응형 작업 스케줄링 기법은 데드라인까지 작업 수행을 완료해야 하는 상황에서 성능 지향 정책과 비용 지향 정책에 따라 자원을 적절히 할당하여 자원 활용의 효율을 극대화하는 것을 목표로 한다. 또한 사용자가 원하는 정책을 기반으로 한 스케줄링 수행을 통해 SLA를 제공한다. 제안하는 알고리즘에서 사용하는 기호 및 상세는 표 1과 같다.

본 스케줄링 기법에서는 사용자 요구하는 SLA에 따라 성능/비용 지향 정책 중 하나의 스케줄링 정책이 선택되고, 해당 정책을 이용했던 응용 수행 이력(응용 수행시간, 비용, 데드라인 등)을 사용자에게 제공하여 이를 바탕으로 데드라인을 선택할 수 있도록 함으로써 의미 있는 데드라인을 통한 스케줄링을 제공한다.

전체 스케줄링 알고리즘은 표 2에 나타나 있다. 표 2에서와 같이 선택된 스케줄링 정책에 따른 응용의 수행 시간과 자원 사용에 대한 비용이 명시된 작업 이력을 사용자에게 제시함으로써 이를 바탕으로 사용자가 데드라인을 선택할 수 있도록 한다(line 1-2). 데드라인을 결정한 다음, 사용자의 요구에 따라 성능 지향 정책 또는 비용 지향 정책에 관한 스케줄링을 수행한다(line 3-7). 마지막으로 다음 데드라인 결정을 보다 최적화하기 위해 응용의 수행 시간 및 자원 사용 비용을 응용 수행 이력으로 JobHistory 데이터베이스에 저장한다(line 9).

표 1 스케줄링 알고리즘 기호 명세
Table 1 Scheduling algorithm notation

Notation	Explanation
$\mathcal{R}q_i \ni \{j_1^i, j_2^i, \ldots, j_n^i\}$	$\mathcal{R}q_i$ is an application, j_k^i is kth job of the application
D_i	D_i is a deadline of application
$Cost_i$	$Cost_i$ is a total cost for the execution.
$P \ni \{P_1, P_2, \ldots, P_n\}$	A set of job profiles
P_{new}	A new job profile, which will be saved in Job history database (*JobHistory*)
$r \ni \{r_1, r_2, \ldots, r_n\}$, $R \ni \{r_1, r_2, \ldots, r_m\}$	r is a set of all types of resources, R is a set of the deadline -safe resources.
$fr \ni \{fr_1, fr_2, \ldots, fr_n\}$, $FR \ni \{fr_1, fr_2, \ldots, fr_m\}$	fr is a set of all available free charge resources(Grid, Private cloud), FR is a set of the free charge resources which are deadline-safe.
$\mathcal{pb} \ni \{\mathcal{pb}_{type_1}, \mathcal{pb}_{type_2}, \ldots, \mathcal{pb}_{type_n}\}$ $\mathcal{pb}_{type_x} \ni \{vm_1, vm_2, \ldots, vm_m\}$	$\mathcal{pb}$ is a set of public cloud resources, $\mathcal{pb}_{type_x}$ is a set of virtual machines which have an instance type, $type_x$
EFT_{r_x}	Estimated Finish time in r_x
$EST_{j_k^i}, AST_{j_k^i}$	Estimated Start time and Actual Start time of j_k^i
$ET_{r_n}(j_k^i)$	In resource r_x, estimated execution time of j_k^i

표 2 전체 알고리즘
Table 2 Overall Algorithm

Algorithm1. Overall Resource Scheduling

1: User choose a scheduling policy
2: User choose deadline $\mathcal{D}_i$ for $\mathcal{R}q_i$ based upon P from *JobHistory*
3: **switch** (scheduling policy of $\mathcal{R}q_i$)
4: **case** *Performance*:
5: Performance-oriented_Scheduling($\mathcal{R}q_i, \mathcal{D}_i$); break;
6: **case** *Cost*:
7: Cost-oriented_Scheduling($\mathcal{R}q_i, \mathcal{D}_i$); break;
8: **end switch**
9: Accumulate $P_{new}(\mathcal{R}q_i, \mathcal{D}_i, Cost_i)$ in *JobHistory*

성능 지향 정책이 선택된 경우는 표 3에 나와있는 성능 지향 정책 스케줄링 알고리즘이 수행된다. 이 알고리즘은 응용에 속해있는 작업들이 각 자원에서 수행될 경우, 각 자원의 예상 수행 종료 시간들 중에 가장 빠른 수행 종료 시간을 갖는 자원에 작업을 할당한다. 이 때, 수행할 작업(j^i_k)들을 작업의 길이에 따라 정렬시킨 후, 그리드, 사설/상용 클라우드 자원을 포함하는 모든 자원(r)에서 각 작업에 대한 해당 자원의 예상 수행 종료 시간(EFT_{r_x})을 다음과 같은 수식으로 계산한다(line 4-7).

$$EFT_{r_x} = EarliestStartTime_{r_x}(j_k{}^i) + ET_{r_x}(j_k{}^i)$$

그리고 각 자원에서 얻어진 예상 수행 종료 시간 중에서 데드라인 안에 작업의 수행이 가능한지를 확인하고, 가능한 자원 중 가장 빨리 작업 수행을 완료시킬 수 있는 자원에 작업을 스케줄링한다(line 8-13).

모든 작업이 적정 자원에 스케줄링된 후, N간격의 주기적인 자원 모니터링을 수행한다(line 17-18). 표 4에 나온 모니터링 알고리즘은 작업의 지연을 감지하고 이로 인한 데드라인 위반 가능성을 확인한다. 현재 자원에서 수행되는 작업들에 대해서 예상 수행 시작 시간($EST_{j^i_k}$)과 실제 수행 시작 시간($AST_{j^i_k}$)을 비교한다(line 3). 이 때 예상한 것보다 실제 시작 시간이 늦어졌을 경우 앞서 수행된 작업의 지연으로 판단하고 지연 정도(delay)를 계산한다(line 4). 지연 정도가 해당 자원에서 가장 마지막으로 수행되는 작업의 수행을 방해할 경우, 모니터링은 데드라인 위반 가능성을 알린다(line 6). 이를 통해 스케줄링 알고리즘(표 3 또는 5)에서 대기 중인 작업들에 대해서 재스케줄링이 이루어진다(line 3).

비용 지향 정책 기반의 스케줄링은 가능한 많은 작업을 비-상용 자원에 스케줄링하고 난 후, 비-상용 자원에서 스케줄링 시 데드라인을 위반할 가능성이 있는 후반 작업들에 대해서는 상용 클라우드 자원들 중 비용 대비 높은 성능의 효율을 가지는 클라우드 순으로 자원을 할당한다. 표 5는 비용 지향 정책 기반의 스케줄링 알고리즘이다. 이 정책 또한 그리드와 사설 클라우드 자원으로 이루어진 비-상용 자원(fr)에 대해서는 성능 지향 정책과 동일한 과정을 보인다(line 4-14). 하지만 데드라인을 위배하게 되는 경우 구매한 상용 클라우드 자원에 스케줄링을 시도한다. 보유 중인 상용 클라우드 인스턴스 타입에 대한 가격 대비 성능을 Efficiency라 하고 구하는 수식은 다음과 같다.

$$Efficiency = \frac{(MIPS \times Number\ of\ Cores)}{Price\ per\ an\ hour}$$

이 값은 작업의 스케줄링을 위해 할당되는 인스턴스 타입의 우선 순위가 된다(line 15). Efficiency가 높을수록 자원 고려 순위가 높아지고, 높은 Efficiency순으로 자원

표 3 성능 지향 정책 스케줄링 알고리즘
Table 3 Performance-oriented policy Scheduling Algorithm

Algorithm2. Performance-oriented Scheduling

1: Performance-oriented_Scheduling($\mathcal{R}q_i, \mathcal{D}_i$){
2: **while**(*ANY_JOBS_TO_RUN*){
3: **if** (*POSSIBILITY_OF_DEADLINE_VIOLATION*){
4: Sort $\mathcal{R}q_i$ in decreasing order of execution time of j^is, which are not running
5: **for each** $j_k{}^i \in \mathcal{R}q_i$ AND $j_k{}^i$ is not running **do**
6: **for each** resource r_x **do**
7: $EFT_{r_x} = EarliestStartTime_{r_x}(j_k{}^i) + ET_{r_x}(j_k{}^i)$
8: **if** ($EFT_{r_x} \leq \mathcal{D}_i$) **then**
9: $R = R \cup \{r_x\}$
10: **end if**
11: **end for**
12: **if** ($R \not\ni \phi$) **then**
13 Schedule $j_k{}^i$ in $r_x \in R$ such that EFT_{r_x} is minimum
14: **end if**
15: **end for**
16: **end if**
17: waitForNextMonitoring(N)
18: *POSSIBILITY_OF_DEADLINE_VIOLATION* = Monitoring($\mathcal{D}_i$)
19: **end while**
20: }

표 4 모니터링 알고리즘
Table 4 Monitoring Algorithm

Algorithm3. Monitoring

1: Monitoring($\mathcal{D}_i$){
2: **for each** $j_k{}^i \in \mathcal{R}q_i$ AND $j_k{}^i$ is running **do**
3: **if** ($AST_{j_k{}^i}$ *is latter than* $EST_{j_k{}^i}$) **then**
4: $delay = AST_{j_k{}^i} - EST_{j_k{}^i}$
5: **if** ($EFT_r + delay > \mathcal{D}_i$) **then**
6: return TRUE
7: **end if**
8: **end if**
9: **end for**
10: return FALSE
11: }

표 5 비용 지향 정책 스케줄링 알고리즘
Table 5 Cost-oriented policy Scheduling Algorithm

Algorithm4. Cost-oriented Scheduling

1: Cost-oriented_Scheduling($\mathcal{Rq}_i, \mathcal{D}_i$){
2: **while** (*ANY_JOBS_TO_RUN*){
3: **if** (*POSSIBILITY_OF_DEADLINE_VIOLATION*){
4: Sort $\mathcal{Rq}_i$ in decreasing order of execution time of j^is, which are not running
5: **for each** ${j_k}^i \in \mathcal{Rq}_i$ AND ${j_k}^i$ is not running **do**
6: **for each** free charge_resource fr **do**
7: $EFT_{fr_x} = EarliestStartlrTime_{fr_x}({j_k}^i) + ET_{fr_x}({j_k}^i)$
8: **if** ($EFT_{fr_x} \leq \mathcal{D}_i$) **then**
9: $FR = FR \cup \{fr_x\}$
10: **end if**
11: **end for**
12: **if** ($FR \not\ni \phi$)**then**
13: Schedule ${j_k}^i$ in $fr_x \in FR$such that EFT_{fr_x} is minimum
14: **else**
15: Sort type of public instance type $type_x$ by order of its $Efficiency = \frac{(MIPS \times Number\ of\ Cores)}{Price\ per\ an\ hour}$
16: **for each** public cloud $\mathcal{pb}$ **do**
17: $EFT_{\mathcal{pb}_{type_x}} = EarliestStartTime_{\mathcal{pb}_{type_x}}({j_k}^i) + ET_{\mathcal{pb}_{type_x}}({j_k}^i)$
18: **if** ($EFT_{\mathcal{pb}_{type_x}} \leq \mathcal{D}_i$) **then**
19: schedule ${j_k}^i$ in $\mathcal{pb}_{type_x}$
20: **else**
21: Check next $type_x$ which has next-highest $Efficient$
22: **end if**
23: **end for**
24: **end if**
25: **end for**
26: **end if**
27: waitForNextMonitoring(N)
28: *POSSIBILITY_OF_DEADLINE_VIOLATION* = Monitoring($\mathcal{D}_i$)
29: **end while**
30: }

에서 수행되는 작업 완료 시간을 구한다(line 17). 데드라인을 만족하면 작업을 해당 자원에 스케줄링하고 그렇지 않은 경우 다음으로 높은 Efficiency를 가진 인스턴스 타입을 가진 자원에 스케줄링을 시도한다(line 18-22). 이렇게 비용 지향 정책으로 스케줄링 된 작업들은 N간격으로 수행되는 자원 모니터링을 통해서 예상치 못한 작업의 지연으로 인한 데드라인 위반을 예방한다(line 27-28).

4. 성능평가 및 분석

4.1 실험 환경

시뮬레이션을 통해 본 논문에서 제안하는 하이브리드 클라우드 적응형 작업 알고리즘의 우수성을 평가한다. 이를 위해 CFD분야의 항공우주 응용을 이용하여 시뮬레이션을 수행하였다.

본 성능평가는 그리드(Globus[13] 기반), 사설 클라우드(OpenNebula[14] 기반), 상용 클라우드(Amazon EC2 기반)에서 해당 응용 프로그램을 실제로 수행하여 얻은 성능 측정값들을 바탕으로 작업을 정의하였으며, 시뮬레이션 상의 작업을 50,000개로 설정하였다. e-AIRS 2.0[15]을 이용해서 1년간 축적한 작업 프로파일을 통해 얻은 응용의 평균 길이는 시뮬레이션을 위한 응용 작업의 기준치로 이용되었다. 시뮬레이션 상에 정의된 컴퓨팅 환경에 대한 자세한 명세는 표 6과 같다. 자원 중 그리드와 사설 클라우드는 실제 응용이 수행되었던 자원의 성능치를 기준으로 정의하였으며, 상용 클라우드의 경우 Amazon EC2의 인스턴스 타입의 성능치를 기반으로 정의하였다. 표 6에서 "DMIPS"는 마이크로 프 로세서의 성능을 테스트하기 위한 벤치마크 테스트 방법인 Dhrystone MIPS(DMIPS)[16]을 이용하여 기존 응용이 수행되었던 자원의 CPU성능을 측정한 값을 기술한 항목이다.

4.2 실험 결과

그림 3은 비용 지향 정책 기반의 작업 스케줄링 결과이다. 그림 3(a)는 데드라인은 1시간 단위로 변화를 주어 그 변화에 따른 응용의 수행시간과 처리 비용을 보여 주고 있다. 데드라인이 7시간인 경우에 비용은 $77.14, 8시간인 경우에는 $48.64, 9시간인 경우에는 $34.2, 10시간인 경우에는 $7.6의 비용이 발생하고, 그 이상의 데드라인에서는 응용이 비-상용 자원만으로 수행이 가능했기 때문에 비용이 발생하지 않는다. 데드라인을 짧게

표 6 성능 측정을 위한 자원 명세
Table 6 Resource Specification for Performance Evaluation

Type of Reources		Resource Characteristics				
		# of Core per resource	DMIPS	Memory	Price per hour	# of vm/host
Grid		16 cores	634.1	15 GB	-	5
Private Cloud		2 cores	1028.4	1 GB	-	20
Public Cloud	Large	2 cores	49707	7 GB	$0.368	40
	Extra Large	4 cores	608.5	15 GB	$0.736	20
	High-CPU Medium	2 cores	656	1.7 GB	$0.190	20
	High-CPU Extra Large	8 cores	657.6	7 GB	$0.760	10

할수록 처리속도는 빨라지고 처리비용은 증가하는 반면, 데드라인을 길게 설정할수록 처리속도는 상대적으로 느려져서 수행 시간이 길어지지만 데드라인 안에서 응용의 수행을 보장하고 처리비용이 절감되는 것을 확인할 수 있다. 그림 3(a)에서 데드라인을 10시간으로 설정하였을 때 데드라인 안에 응용의 수행이 완료되고 비용이 급격하게 감소한다. 또한 10시간 이상으로 데드라인을 증가시켜도 수행 시간 및 비용 측면에서는 큰 차이를 보이지 않기 때문에 본 실험에서는 해당 CFD 기반의 응용에 대한 수행을 데드라인 10시간으로 가정했다.

그림 3(b)에서는 데드라인 10시간을 기준으로 모니터링 간격에 대한 변화를 통해 적응형 스케줄링을 반복적으로 수행한 결과를 실패한 작업 개수로 보여주고 있다. 작업의 지연으로 인한 데드라인 위반 가능성을 적응형 스케줄링을 통해서 일정부분 배제가 가능하기 때문에 그림 3-7(b)와 같이 적응형 스케줄링을 한 번 돌렸을 때, 50,000개의 작업 중 실패한 작업의 개수가 약 3232개로 약 0.065% 데드라인 위반율을 보이고 두 번의 적응형 스케줄링으로 인해 단 한 개의 작업 실패도 없는 데드라인 위반율 0%를 보이고 있다. 이는 3232개의 작업들이 두 번의 적응형 스케줄링을 통해 적정 자원을 다시 할당 받음으로써 데드라인 위반율이 감소했음을 알 수 있다. 하지만 세 번 이상의 적응형 스케줄링은 응용 수행에 지장은 없으나 두 번의 적응형 스케줄링 수행을 한 경우에 비해 추가적인 비용이 발생한다.

이 실험결과를 통해 자원 모니터링 주기가 잦을수록 작업 지연 발생으로 인한 적응형 스케줄링 수행 횟수가 늘어나며 작업 수행 실패를 미연에 방지할 수 있다는 것을 알 수 있지만 불필요한 스케줄링을 수행함으로써 비용 낭비를 초래할 수 있다. 즉, 적응형 스케줄링 수행

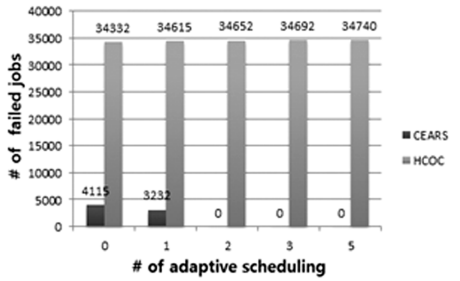
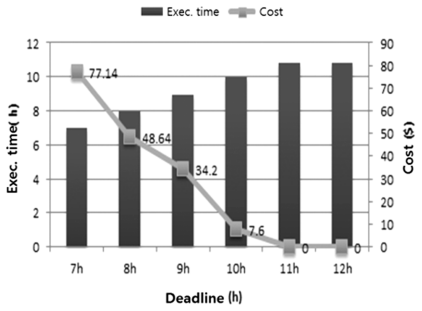


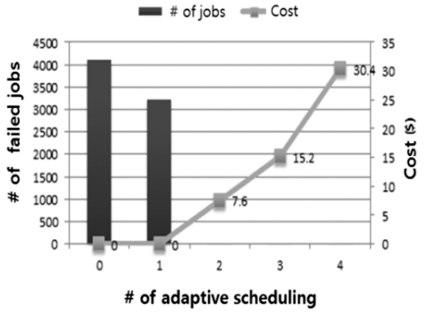
그림 4 HCOC 스케줄링과의 데드라인 위반율 비교
Fig. 4 Comparison with HCOC Scheduling in Deadline Miss Rate of CFD

빈도수에 따른 오버헤드 증가를 막기 위한 최적의 스케줄링 수행 주기 값을 결정하는 것 또한 중요하다.

그림 4에서는 HCOC 스케줄링은 데드라인 설정 시 수행 응용의 Critical Path(CP)에 속하는 DAG 노드들에 대해 가장 성능 좋은 자원에서 수행한 총 시간을 구해 데드라인으로 설정하기 때문에 이 실험에서 수행하는 Bag of Task 응용 경우, CP에 속하는 작업은 가장 긴 작업 한 개로 이 작업의 수행 시간이 데드라인이 된다. HCOC 스케줄링도 CEARS와 유사하게 필요에 따라 스케줄링의 재수행(적응형 스케줄링)이 가능하다는 점에서 스케줄링의 재시도에 대한 데드라인 위반율을 비교해본 결과, 재수행 횟수에 관계없이 50,000개의 작업들 중 평균적으로 약 3,500개의 작업들이 수행에 실패하였기 때문에 총 5번의 실험에서 5번 모두 수행 실패를 보임으로써 100%의 응용 수행 실패율을 보였다. HCOC 스케줄링은 데드라인을 설정할 때 응용의 특성만 고려하여 현재 사용 가능한 자원의 속성을 고려하지 않아 위와 같은 결과로 응용 수행이 불가능한 상황이 되었다. 본 연구에서 제안하는



(a) 데드라인에 따른 응용 수행시간 및 처리비용
(Execution time and cost based on each deadline)



(b) 적응형 스케줄링 횟수에 따른 실패한 작업 개수 및 처리비용
(Number of failed tasks and cost based on adapive scheduling times)

그림 3 CFD 응용에 대한 비용 지향 정책 기반의 작업 스케줄링
Fig. 3 Cost Policy-based Scheduling in CFD

CEARS는 데드라인 기준을 마련할 때, 응용의 특성과 수행 자원을 모두 고려한 데드라인을 정하여 응용 수행의 성공률이 높아짐을 보여주었다. 이 실험 결과를 통해서 응용의 특성과 수행 자원에 대한 고려가 응용 수행에 결정적인 영향을 미친다는 것을 알 수 있다.

5. 결 론

본 논문에서는 주어진 데드라인 내에 응용을 수행 시, 다중 인프라 자원 고려를 통한 자원의 효용성을 증대시키고 유연한 스케줄링을 통해 사용자의 SLA를 만족시키는 작업 스케줄링 서비스를 제안하였다. 또한 정책에 따른 응용 수행 보장하기 위해 필요에 따라 작업을 적정 자원에 재배치함으로써 데드라인 위반율을 감소시킴을 확인하였다.

향후 연구과제로는 과학 워크플로우의 파이프라인, 병렬, 하이브리드 등의 다양한 응용 모델에 대한 확장과 저전력 정책 등 추가 정책 수립을 포함하는 스케줄링 기법으로 확장할 계획이다.

References

[1] Amazon Elastic compute cloud, [Online]. Available: http://aws.amazon.com/ec2/

[2] Windows Azure, [Online]. Available: http://www.windowsazure.com/ko-kr/

[3] Rackspace, [Online]. Available: http://www.rackspace.com/

[4] Computational Fluid Dynamics, [Online]. Available: http://www.cfd-online.com/

[5] Bossche R., Vanmechelen K., and Broeckhove J.," Cost-Efficient Scheduling Heuristics for Deadline Constrained Workloads on Hybrid Clouds," *Proc. of the 3rd IEEE International Conference on Cloud Computing technology and Science*, pp.320-327, 2011.

[6] GoGrid, [Online]. Available: http://www.gogrid.com/

[7] Bittencourt L., and Maderia E., "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.

[8] Ramakrishnan L., Koelbel C., Kee Y., Wolski R., Nurmi D., Gannon D., Obertelli G., YarKhan A.,Mandal A., Huang T. M., Thyagaraja K., and Zagorodnov D., "VGrADS: enabling e-Science workflows on grids and clouds with fault tolerance," *Proc. of the SC '09 Conference on High Performance Computing Networking, Storage and Analysis*, ACM New York, NY, 2009.

[8] TeraGrid. [Online]. Available: https://www.xsede.org/home

[10] Eucalyptus. [Online]. Available: http://www.eucalyptus.com/

[11] Vecchiola C., Chu X., Mattess M., and Buyya R.," Aneka-a software platform for NET-based cloud computing," *Proc. of High Performance and Large Scale Computing*, IOS Press, pp.267-295, 2009.

[12] Kim, Hyunjoo, et al., "An autonomic approach to integrated hpc grid and cloud usage," e-Science, 2009. e-Science'09. *Proc. of the fifth IEEE International Conference on. IEEE*, 2009.

[13] Globus. [Online]. Available: http://www.globus.org/

[14] OpenNebula. [Online]. Available: http://opennebula.org/

[15] e-AIRS 2.0. [Online]. Available: http://eairs.kisti.re.kr/gridsphere

[16] Dhrystone MIPS, [Online]. Available: http://en.wikipedia.org/wiki/Dhryston

고 정 인

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

강 혜 정

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

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