제36회 한국정보처리학회 추계학술발표대회 논문집 제18권 제2호 (2011. 11)

클라우드 컴퓨팅에서 비용-효율적 스팟 인스턴스를 위한 시간 문턱치 기반의 검사점 기법

정대용*, 유헌창*, 길준민**
*고려대학교 컴퓨터교육과
**대구가톨릭대학교 컴퓨터정보통신공학부
e-mail: karat@korea.ac.kr, yuhc@korea.ac.kr, jmgil@cu.ac.kr

A Time Threshold-based Checkpointing Scheme for Cost-Efficient Spot Instances in Cloud Computing

Daeyong Jung*, HeonChang Yu*, Joon-Min Gil**
*Dept. of Computer Science Education, Korea University
**School of Computer & Information Communication Engineering,
Catholic University of Daegu

요 약

클라우드 환경에서 스팟 인스턴스(spot instance)는 사용자가 제시한 입찰 가격으로 클라우드 내의 자원을 활용하도록 해 준다. 그러나 사용자의 입찰 가격이 클라우드 자원 가격보다 높으면 작업 실패가 발생하고 이로 인해 작업 완료 시간의 지연과 서비스 품질의 저하를 야기한다. 이 문제에 효과적으로 대처하기 위해, 본 논문에서는 시간 문턱치 기반의 검사점(time threshold-based checkpointing) 기법을 제안하고, 시뮬레이션을 통하여 작업 수행 시간과 비용 절감 관점에서 기존 기법과 비교분석한다.

1. 서론

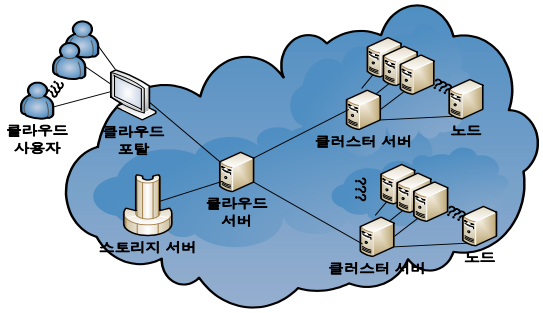
최근 클라우드 컴퓨팅에 대한 폭발적 관심으로 인해 많은 개발 프로젝트와 상업적 시스템이 구현되고 있다. 대다수의 클라우드 컴퓨팅 시스템에서는 가상머신(virtual machine; VM)을 의미하는 인스턴스를 비용-효율적 방법으로 사용자가 제공받도록 해준다. 일반적으로 인스턴스는 주문형 인스턴스(on-demand instance)와 스팟 인스턴스(spot instance)로 구분된다. 주문형 인스턴스는 사용자의 요구에 따라 자원을 가상화시켜 사용자에게 제공하는 것이고, 스팟 인스턴스는 사용자 입찰 가격에 근거하여 클라우드 내의 자원을 활용하게 해준다[1].

한편, 스팟 인스턴스 기반의 클라우드 환경에서는 사용자의 입찰 가격이 자원 가격보다 적게 제시될 때에만 VM을 활용할 수 있다. 그러나 자원 가격은 자원의 수요와 공급에 따라 주기적 혹은 비주기적으로 변동한다. 그래서 사용자 입찰 가격이 자원 가격보다 적게 제시되어 VM을 현재 활용하고 있더라도 자원 가격의 변동에 따라 사용자의 입찰 가격은 언제든지 초과될 수 있다. 이러한 상황에서는 수행 중인 VM은 즉각 중지되고 VM 내에서 동작하는 작업은 실패하게 된다[2].

이 문제에 대처하고자, 본 논문에서는 시간 문턱치 기반의 검사점(time threshold-based checkpointing) 기법을 제안한다. 제안 검사점 기법에서는 작업의 평균 수행 시간을 이용하여 VM의 실패 발생 시점을 예상하고, 이 시점에서 검사점을 수행한다. 따라서 자원 가격 변동에 따른 작업 실패에 적극적으로 대처하여 VM의 회복 시에 소요되는 회복 시간을 줄여 작업 완료 시간을 줄인다. 제안 검사점 기법의 성능평가를 위해 본 논문에서는 아마존 EC2에서 제시된 자원 가격 변동 데이터를 이용하여 시뮬레이션을 수행한다. 시뮬레이션 결과는 제안 검사점 기법이 기존 검사점 기법에 비해 작업 수행 시간과 비용 절감 관점에서 우수함을 보여준다.

2. 시스템 아키텍처

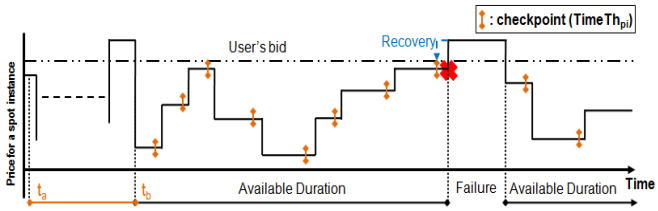
(그림 1)은 본 논문에서 가정하는 클라우드 컴퓨팅 환경을 보여준다. 이 그림의 클라우드 컴퓨팅 환경은 크게 클라우드 서버, 클러스터 서버, 스토리지 서버, 클라우드 사용자로 이루어져 있다. 클라우드 서버는 클러스터 서버와 스토리지 서버로 연결되어 있으며, 클러스터 서버는 수많은 컴퓨팅 노드로 구성된다. 클라우드 사용자는 클러스터 서버 내의 컴퓨팅 노드를 이용하기 위해 클라우드 포탈을 통해 클라우드 서버로 접근한다. 그러므로 클라우드 서버는 클라우드 사용자의 요구사항을 만족하는 가상 자원을 생성하고 작업 관리의 역할을 한다. 본 논문에서는 클러스터 서버 내의 컴퓨터 노드에서 수행되는 스팟 인스턴스에서 비용-효율적으로 작업이 수행되도록 스팟 인스턴스의 검사점 수행과 관리에 초점을 맞춘다. 스팟 인스턴스의 검사점 수행은 각각의 컴퓨팅 노드에서 수행되며, 검사점 관리는 클라우드 서버에서 수행된다.



(그림 1) 클라우드 컴퓨팅 환경

3. 시간 문턱치 기반의 검사점 기법

스팟 인스턴스 환경에서 수행 중인 작업의 실패는 자원 가격이 사용자의 입찰 가격을 초과할 때 발생한다. 이 절에서는 이러한 문제를 해결하기 위해 결함포용 기법으로서 시간 문턱치 기반의 검사점 기법을 제안한다.



(그림 2) 제안하는 검사점 기법의 예시

(그림 2)는 시간 문턱치 기반의 검사점 기법에 대한 예시를 보여준다. 제안 기법은 기본적으로 자원 가격의 변동 정보를 바탕으로 계산된 예상 수행 시간을 사용한다. t_a와 t_b를 예상 수행 시간의 시작점과 끝점이라 하자. 그러면, 시간 문턱치 $TimeTh_{p_i}$는 다음과 같이 계산된다.

$$TimeTh_{p_i} = AvgTime(t_a, t_b) \times (1 - F_{p_i}) \quad (1)$$

여기서, F_{p_i}는 자원 가격 p_i에 대한 실패 발생 확률을 나타내며, $AvgTime_{p_i}(t_a, t_b)$는 자원 가격 p_i에 대해서 t_a와 t_b의 구간 내에서의 평균 작업 수행 시간을 나타낸다. 수식 (1)를 통하여 계산된 시간 문턱치를 이용하여 본 논문의 검사점 기법은 시간 문턱치가 현재 자원 가격을 초과할 때마다 검사점을 수행한다.

(그림 3)은 제안 기법에서의 검사점과 회복 알고리즘을 보여준다. 이 알고리즘에서 작업 실패의 발생 여부를 나타내는 flag는 초기에 false 값으로 설정된다. 검사점 처리 과정은 모든 작업이 완료될 때까지 반복 수행된다. 작업 수행이 정상적일 때(즉, flag가 false일 때), 스케줄러는 작업 실패에 대비하기 위해 검사점 처리를 수행하며(2-17 줄), 회복 과정은 flag가 true일 때 수행된다(4-7 줄). 작업 수행 시간이 시간 문턱치보다 크다면, 스케줄러는 검사점 연산을 수행한다(9-11). 작업 실패가 발생했을 때, flag는 회복 과정의 수행을 위해 true로 설정된다(14-16 줄). 이 알고리즘에서 18-21 줄과 22-25 줄은 각각 검사점과 회복에 대한 자세한 처리 과정을 보여준다.

```
1: Boolean flag = false
2: while (!task execution finishes) do
3:   if (spot prices < user's bid) then
4:     if (flag) then
5:       Recovery();
6:       flag = false;
7:     end if
8:     if (!flag) then
9:       if (Time Threshold < execution time) then
10:          Checkpoint();
11:      end if
12:    end if
13:  end if
14:  if (failure occurs) then
15:    flag = true;
16:  end if
17: end while
18: Function Checkpoint()
19:   take a checkpoint on the spot instance;
20:   save the checkpoint in the storage;
21: end Function
22: Function Recovery()
23:   rollback the checkpoint to the storage;
24:   restart task execution;
25: end Function
```

(그림 3) 검사점과 회복 알고리즘

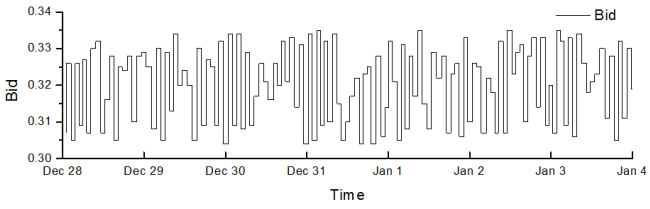
4. 성능 평가

4.1 시뮬레이션 환경

본 논문에서는 아마존 EC2로부터 얻은 자원 가격의 변동 데이터를 이용하여 시뮬레이션을 수행하였다. (그림 4)는 <표 2>에서 보여주는 c1.xlarge 인스턴스 유형에 대해 2010년 12월 28일부터 2011년 1월 4일까지 총 8일 동안의 자원 가격에 대한 변동 추이를 보여준다. 2010년 12월 31일 이전의 데이터는 예상 수행 시간과 실패 발생 확률을 추출하기 위해 이용되었으며, 추출된 예상 수행 시간과 실패 발생 확률에 기반으로 2011년 1월 1일 이후의 자원 가격 데이터가 성능 평가를 위해 적용되었다[3].

<표 2> c1.xlarge 자원 유형

인스턴스명	계산 유닛	가상 코어	메모리	스토리지	플랫폼
c1.xlarge	8 EC2	4 core (2 EC2)	15GB	1690GB	64-bit

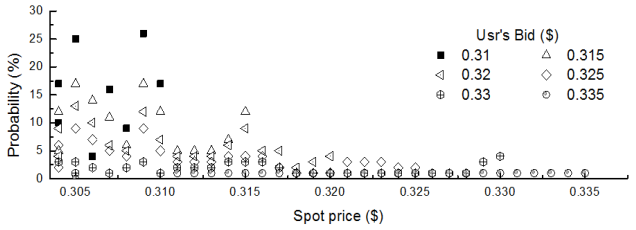


(그림 4) c1.xlarge 자원의 가격 변동 추이

<표 3>은 시뮬레이션을 위해 사용되는 파라미터와 값들을 보여주며, (그림 5)는 c1.xlarge 인스턴스에 대해서 사용자의 제시 비용이 주어졌을 때 자원 가격별 실패 발생 확률을 보여준다.

<표 3> 시뮬레이션 파라미터와 값

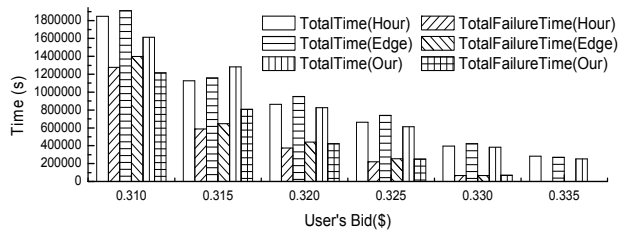
시뮬레이션 파라미터	작업 시간(초)	최대/최소 입찰가($)	평균 입찰가($)	검사점 시간(초)	회복 시간(초)
값	259,200	0.336/0.304	0.319	300	300



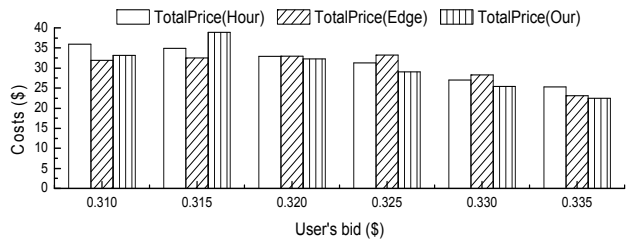
(그림 5) c1.xlarge에 대한 실패 발생 확률

4.2 시뮬레이션 결과

(그림 6)은 사용자 입찰 가격별 제안 검사점 기법과 기존 검사점 기법(hour-boundary 검사점 기법[4]과 rising edge-driven 검사점 기법[5])의 작업 수행 시간과 실패 시간의 비교를 보여준다. 이 그림의 결과로부터 제안 검사점 기법은 hour-boundary 검사점 기법에 비해 평균 4.3%의 작업 수행 시간의 절감과 rising edge-driven 검사점 기법에 비해서는 평균 8.6%의 작업 수행 시간의 절감을 볼 수 있다.



(그림 6) 검사점 기법에 대한 작업 수행 시간과 실패 시간의 비교



(그림 7) 검사점 기법에 대한 총 비용의 비교

(그림 7)은 사용자 입찰 가격별 제안 검사점 기법과 기존 검사점 기법의 총 비용을 보여준다. 이 그림의 결과는 제안 검사점 기법이 hour-boundary 검사점과 rising edge-driven 검사점 기법에 비해 평균 3.67$과 0.62$의 비용 절감 효과가 있음을 알 수 있다.

(그림 6)과 (그림 7)의 결과는 본 논문에서 제안하는 검사점 기법이 사용자 입찰 가격에 상관없이 실패 시간을 줄일 수 있어서 기존 검사점 기법에 비해 훨씬 적은 비용을 사용하였음에도 불구하고 작업 수행 시간을 줄일 수 있음을 나타낸다. 이는 본 논문의 검사점 기법이 스팟 인스턴스의 가격 변동 데이터를 적극적으로 고려하여 실패 발생 확률을 추출하고 이를 통해 검사점 수행 여부를 판단하는 시간 문턱치 값을 유도하였기 때문이다.

5. 결론 및 향후 연구

본 논문에서는 스팟 인스턴스 기반의 클라우드 컴퓨팅 환경에서 작업 수행의 안정성을 제공하기 위해 자원 가격 변동을 적극적으로 고려한 시간 문턱치 기반의 검사점 기법을 제안하였다. 제안 검사점 기법에서는 자원의 가격 변동에 의해 발생하는 실패에 효과적으로 대처하기 위해 비용 효율적 방법으로 검사점 위치를 결정하였다. 따라서 제안 검사점 기법은 기존 검사점 기법에 비해 실패 시간을 자원의 가격 변동에 상관없이 줄일 수 있어 전반적으로 작업 수행 시간을 줄인다. 아마존 EC2의 실제 자원의 가격 변동 데이터를 활용하여 얻어진 시뮬레이션 결과는 사용자 입찰 비용에 상관없이 훨씬 적은 비용으로도 작업 수행 시간을 줄일 수 있음을 보여주었다.

본 논문에서는 제안 검사점 기법을 컴퓨팅 연산을 위한 x1.large 인스턴스에 적용하였다. 향후 연구로서 제안 검사점 기법의 활용성을 보이기 위해서 메모리 연산 등 다양한 유형의 인스턴스에 제안 검사점 기법을 적용할 예정이다.

참고문헌

[1] I. Foster, Z. Yong, I. Raicu, and S. Lu, “Cloud Computing and Grid Computing 360-Degree Compared. Proc. of 2008 Grid Computing Environments Workshop, pp. 1-10, 2008.

[2] Amazon EC2 spot Instances, http://aws.amazon.com/ec2/spot-instances (2011)

[3] Cloud exchange, http://cloudexchange.org, (2011)

[4] S. Yi, J. Heo, Y. Cho, and J. Hong “Taking Point Decision Mechanism for Page-level Incremental Checkpointing based on Cost Analysis of Process Execution Time,” J. of Information Science and Engineering, vol. 23, no. 5, pp. 1325 - 1337, 2007.

[5] S. Yi, D. Kondo, and A. Andrzejak, “Reducing Costs of Spot Instances via Checkpointing in the Amazon Elastic Compute Cloud” Proc. of the 2010 IEEE 3rd Int. Conf. on Cloud Computing, pp. 236-243, 2010.