클라우드 컴퓨팅 환경에서 가쉽 기반 비잔틴 합의 알고리즘

임종범*, 최희석*, 강인성*, 이대원**, 유헌창*§
*고려대학교 컴퓨터교육학과
**서경대학교
e-mail:jblim@korea.ac.kr

A Gossip-based Byzantine Consensus Algorithm in Cloud Computing Environments

JongBeom Lim*, HeeSeok Choi*, InSung Kang*, DaeWon Lee**,
HeonChang Yu*
*Dept of Computer Science Education, Korea University
**SeoKyeong University

요 약

합의는 분산 시스템 환경에서 해결해야 할 근본적인 문제 중 하나이다. 특히 노드 또는 프로세스의 임의적인 실패 즉, 비잔틴 실패가 발생하였을 때 합의 문제는 더 복잡해진다. 본 연구에서는 동적인 노드의 가입과 탈퇴가 자유로운 클라우드 환경에서 비잔틴 합의 문제를 해결하기 위한 가쉽 알고리즘을 제안한다. 제안하는 알고리즘에서 확장성과 결함 포용의 특성을 내재한 가쉽 알고리즘을 적용함으로써 클라우드 환경에서의 비잔틴 합의 문제를 확장적이고 결함 포용적으로 해결할 수 있다. 알고리즘의 성능을 분석하기 위해 성능 평가를 수행하였다.

1. 서론

프로세스 또는 노드들 간의 합의는 분산 시스템에서 해결해야 할 필수불가결한 문제 중 하나이다. 분산 시스템에서 합의 문제가 어려운 이유는 다음과 같다. 공유 메모리가 존재하지 않고, 글로벌 클럭(global clock)이 존재하지 않으며, 프로세스 또는 노드들 간의 통신은 메시지 전달에 의해서만 가능하기 때문이다.

합의 문제 중에서 비잔틴 합의 문제는 실패-중지(fail-stop) 실패가 발생하는 상황보다 더 어려운 문제로 분류할 수 있다. 실패-중지 실패 모델에서는 노드의 실패 발생 시 더 이상의 이상 행동을 발생하지 않는 반면, 비잔틴 실패 모델에서는 실패한 노드는 예상하지 못하는 비정상적인 행동을 계속적으로 수행하기 때문이다.

이러한 비잔틴 합의 문제는 클라우드 컴퓨팅 환경과 같이 노드들의 동적인 특성을 가지는 환경에서는 노드들의 가입과 탈퇴와 같은 추가적인 요구사항을 더 만족시켜주어야 한다. 하지만 비잔틴 합의를 위한 대부분의 기존 연구에서는 노드의 수가 정적, 즉, 노드의 가입과 탈퇴를 허용하지 않는 환경을 가정하였다.

본 연구에서는 동적인 특성을 가지는 클라우드 컴퓨팅 환경에서 비잔틴 합의 문제를 해결하기 위한 가쉽 알고리즘을 제안한다. 가쉽 알고리즘은 주기적으로 대상 노드를 임의적으로 선택하여 메시지를 주고받는 형태로, 시스템을 구성하는 각 노드가 전체 노드 정보를 알지 않더라도 높은 확률로 모든 노드가 메시지를 전달 받는다는 것을 보장한다[1]. 다시 말해, 가쉽 알고리즘에서는 시스템을 구성하는 각 노드가 시스템 내의 전체 노드 정보를 유지하기보다 부분 뷰(partial view)라는 시스템 내의 전체 노드 정보 중 일부만을 가지는 목록을 관리함으로써 확장성의 문제를 극복할 수 있다. 각 노드는 이렇게 만들어진 부분 뷰 내에서 가쉽 대상을 임의적으로 선택하여 메시지를 주고받을 수 있다. 이러한 특성을 가지는 가쉽 알고리즘은 또한 노드가 무수히 많은 환경에서도 확장성을 제공하고 천(churn)-노드의 가입과 탈퇴-과 실패가 잦은 환경에서도 회복력이 뛰어나다.

구체적으로 가쉽 알고리즘을 이용하여 비잔틴 합의 문제를 해결하고 값에 대한 의견 불일치 발생 시 이를 검증하기 위해 위조-증명(fail-stop) 서명 기법[2]을 가쉽 알고리즘에 통합 적용하였다.

본 논문의 나머지 구성은 다음과 같다. 2장에서는 비잔틴 합의 문제와 구체적인 시스템 모델을 명시하고, 3장에서는 위조-증명 서명이 적용된 가쉽 알고리즘을 제안한다. 4장에서는 제안하는 가쉽 알고리즘의 성능 평가 결과를 보여주고, 마지막으로 5장에서는 결론을 통해 본 논문을 마친다.

본 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. 2012-0007429)
§ 교신저자

2. 비잔틴 합의 문제와 시스템 모델

2.1 비잔틴 합의 문제

비잔틴 합의 문제는 일반적으로 크게 합의(consensus) 문제와 비잔틴 장군(Byzantine generals) 문제로 나누어서 생각할 수 있다. 본 논문에서는 비잔틴 합의 문제를 비잔틴 실패가 있는 환경에서의 합의 문제를 다루도록 한다. 단, 합의를 이루기 위한 초기 값은 비잔틴 장군 문제처럼 정해져 있다고 가정한다. 따라서 비잔틴 합의 문제는 아래와 같은 특성을 갖는다.

- 비잔틴 실패가 아닌 프로세스 P_i는 자신의 초기 값을 다른 노드에게 전달한다.
- 비진틴 실패인 프로세스 P_i는 초기 값이 아닌 다른 값을 다른 노드에게 전달한다.

추가적으로 비잔틴 합의 문제는 완전 연결 네트워크를 가정하고 있으므로, 상대적으로 간단한 오버레이 네트워크를 가지는 가쉽 알고리즘 상에서 합의 문제를 해결하기 위해 비잔틴 합의 문제는 상호 일관성(interactive consistency)문제로 대체하여 생각할 수 있다. 상호 인관성 문제는 각 프로세스에 대한 값을 벡터에 저장하고 그 벡터 값에 대한 합의를 이루기 위한 문제이며 다음과 같은 요구사항을 갖는다.

- 종료(termination): 모든 정상적인 프로세스는 결국에는 결정 값을 정한다.
- 합의(agreement): 모든 정상적인 프로세스들의 결정 벡터 값은 같다.
- 유효성(validity): 만약 P_i가 정상적이라면, 모든 정상적인 프로세스는 자신의 벡터 v_i에 P_i가 정한 값으로 결정해야 한다.

2.2 시스템 모델

클라우드 인프라스트럭처를 구성하는 다수의 프로세스 또는 노드들이 존재한다: P_i (i = 1, 2, ..., N). 앞으로 프로세스와 노드 용어는 교체사용 가능한 것으로 간주한다. 각 노드는 자기 자신의 공개키와 개인키를 가지며 이것으로 메시지 서명을 수행한다. 공유 메모리, 글로벌 클럭이 존재하지 않는다. 따라서 노드들 간의 통신은 메시지 전달에 의해서만 가능하다. 통신 채널은 신뢰적이지만 선입선출 형식으로 한정하지는 않는다. 노드들은 실패가 발생할 수 있으며, 실패 모델은 비잔틴 실패이다.

비잔틴 실패에 대한 공격자 모델은 다음과 같이 조건을 만족하는 것으로 가정한다.

- 무한한 계산 능력을 가진 공격자는 다른 정상적인 노드의 공개키를 만족하는 메시지를 생성할 수 있다. 구체적으로 공격자가 정상적인 노드의 서명 $s = SG_x(m)$과 메시지 m 그리고 공개키 K를 안다고 가정할 때, 공격자는 $x^* \in K_{s,m}$을 만족하는 모든 가능한 키를 만들 수 있다고 가정한다. (여기서 $K_{s,m}$은 $s = SG_{x*}(m) = SG_x(m)$을 만족하는 x^*의 집합이다.)
- 공격자는 다른 정상적인 노드의 개인키 x를 모르므로 $K_{s,m}$으로부터 개인키를 추측할 수밖에 없다. 따라서 만약 공격자가 다른 메시지 m^*를 서명한다면 ($m^* \neq m$), $s^* = SG_{x*}(m^*) \neq SG_x(m)$ 을 만족한다.
- 위와 같은 속성으로 공격자는 다른 정상적인 노드를 위장하여 메시지(“공격”, “후퇴” 혹은 ‘1’, ‘0’ 등)를 다른 노드들에게 전달할 수 있다.

구체적인 서명 방법은 부록 A와 동일한 방법을 사용하는 것으로 간주한다.

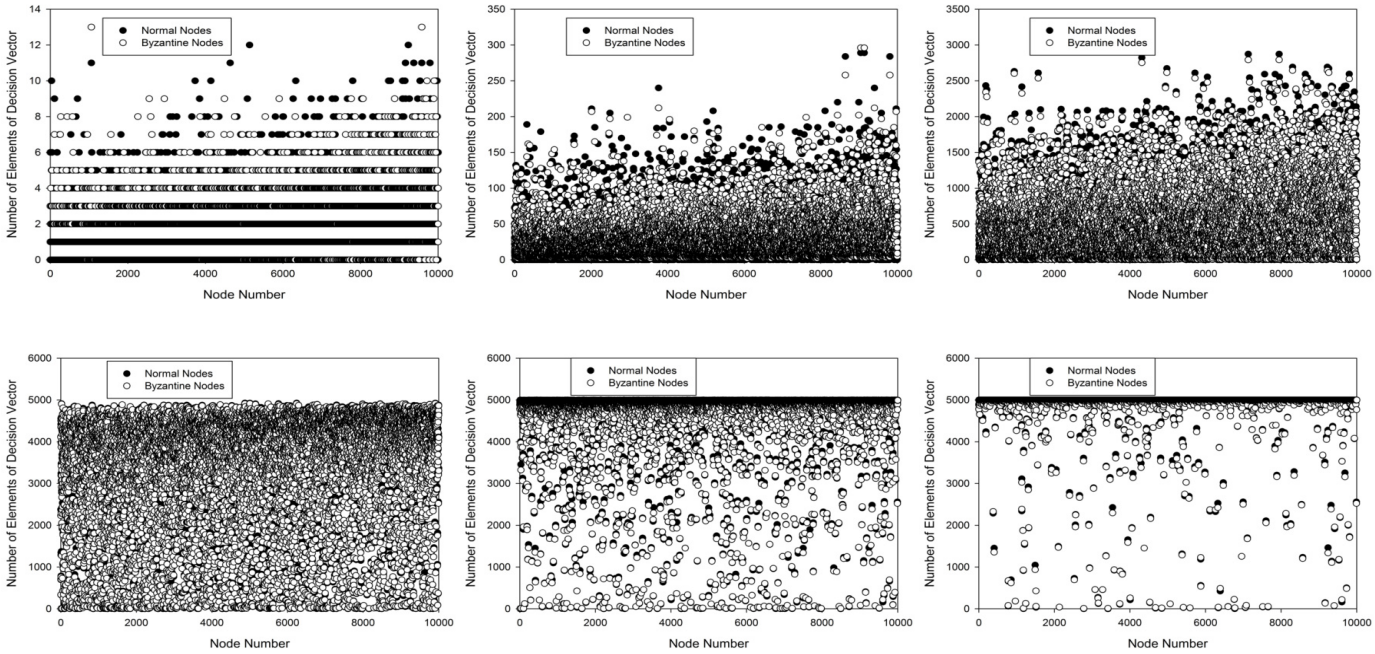
3. 제안하는 알고리즘

(그림 1)은 가쉽 알고리즘을 이용한 비자틴 합의 알고리즘을 보여준다.

```
1: • Initual state for Pi
2:   - vector_i[k] = null, ∀k ∈ {1 ··· N}
3:   - state_i = getStateFromCommander()
4: • During gossiping: Pi selects Pj as target
5:   - SG_x(state_i)
6:   - vector_i[i] = SG_x(state_i)
7:   - sendVector(j, vector_i)
8:   - receiveVector(j, vector_j)
9:   - updateVector(vector_i, vector_j)
10:  - checkConsensus(vector_i)
11: • updateVector() for Pi
12:  - if vector_i[i] ≠ vector_j[i]
13:    - call checkForge(vector_j[i])
14:  - for each element of vector_i[k] and vector_j[k]
15:    - if vector_i[k] == null && vector_j[k] ≠ null
16:      - vector_i[k] == vector_j[k]
17:    - if vector_i[k] ≠ null && vector_j[k] == null
18:      - vector_j[k] == vector_i[k]
19: • checkForge() for Pi
20:  - PROOF(vector_j[i])
21:  - if PROOF() reveals forgery
22:    - stop gossiping and mark Pj as adversary
23: • checkConsensus() for Pi
24:  - for each element of vector_i except for adversary
25:    - check if all elements are non-null and the same
26:    - if line 25 is true
27:      - consensus is reached
28:    - else
29:      - consensus is not reached
```

(그림 1) 제안하는 알고리즘

각 프로세스들은 상호 일관성 문제를 위해 시스템에 존재하는 프로세스의 수만큼의 벡터를 가진다. 또한 합의를 위한 초기 값을 가져온다(줄: 1-3). 만약 프로세스 P_i가 가쉽 메시지 교환을 위해 프로세스 P_j를 선택했다면 다음을 수행한다. 자신의 상태에 대한 메시지를 서명하고 서명 메시지를 자신의 결정 벡터에 저장한다(줄: 5-6). 비잔틴



(그림 2) 사이클이 진행함에 따른 결정 벡터 요소의 수
(• : 정상적인 노드, ◦ : 비잔틴 노드, 상위 왼쪽부터 사이클 1, 4, 7, 10, 13, 16)

합의를 위한 가쉽 알고리즘이 푸쉬-풀(push-pull) 모드를 사용한다면 P_i는 자신의 메시지를 P_j에게 보내고 P_j의 메시지 또한 가져온다(줄: 7-8). 결정 벡터의 업데이트를 마친 후에는 합의 검사를 실시한다(줄: 9-10).

결정 벡터를 업데이트하기 위한 구체적인 알고리즘은 다음과 같다. 먼저 $vector_i[i]$와 $vector_j[i]$가 같은지 검사한다(줄: 12-13). 이 검사가 먼저 이루어지는 이유는 P_j가 비잔틴 실패 상태일 가능성이 있기 때문이다. (본 예에서는 P_i는 정상적인 프로세스라고 가정한다.) 만약 $vector_i[i]$와 $vector_j[i]$가 같지 않다면 비잔틴 실패의 정의에 의해 P_j는 비잔틴 실패 상태임에 틀림없다. 반대로 두 결정 벡터가 같다면 두 프로세스는 모두 정상적인 상태이다. 만약 두 결정 벡터 값이 같았다면 $vector_i$와 $vector_j$를 요소별로 검사하여 벡터 업데이트를 수행한다(줄: 14-18).

만약 두 결정 벡터 값이 다르다면 위조 검사를 수행한다. 부록 A에 정의된 것과 같은 방법으로 해당 서명 값에 대한 위조를 판단한 결과 값이 위조로 판명되었다면 프로세스 P_j를 공격자로 판단하고 j번째 결정 벡터에 공격자로 표시를 해둔다(줄: 19-22).

마지막 단계로 만약 프로세스 P_i가 합의를 검사할 때에는 다음을 수행한다. 자신의 결정 벡터 $vector_i$에서 각 요소를 검사하여 *null*인지 아닌지 여부와 모든 요소가 같은 값을 가지는지 여부를 판단한다. (결정 벡터 $vector_i$에서 공격자로 판단되는 요소의 값은 무시한다.) 만약 모든 요소의 값이 *null*이 아니고 같은 값을 가지고 있다면 P_i는 합의에 도달했다고 판단한다. 만약 그렇지 않다면 합의에 도달하지 않은 것으로 판단한다(줄: 23-29).

4. 성능 평가

제안하는 비잔틴 합의 알고리즘의 유효성을 평가하기 위해 Peersim[3]을 사용하여 실험 환경을 구성하였다. Peersim은 분산 환경에서 다양한 프로토콜의 확장적인 성능 실험을 위해 자바(Java) 언어로 구현된 시뮬레이터로, 본 연구에서 사용한 실험 환경 매개변수는 <표 1>과 같다.

<표 1> 실험에 사용한 환경 변수 및 설정 값

시스템 매개변수	설정 값
n (전체 노드의 수)	10,000
k (부분 뷰가 가지는 노드의 수)	20
Cycle	30
Fanout	1
Gossip Mode	Push-pull
p (노드의 비잔틴 실패 확률)	0.5

실험에서 전체 노드의 수는 10,000, 부분 뷰의 사이즈는 20으로 정하였으며, 가쉽 모드는 푸쉬-풀 모드를 사용하였다. 노드의 비잔틴 실패 확률은 0.5로 설정하였다. 이에 따라 전체 10,000 노드 중 5,001 노드의 수가 정상 노드, 4,999 노드의 수가 비잔틴 노드가 됨을 보였다.

(그림 2)는 가쉽 사이클이 진행함에 따라 전체 노드가 가지는 결정 벡터의 수를 나타낸다. 그림에서 • 는 결정 벡터 중 정상 노드로 판단된 요소의 수를 나타나며, ◦ 는 결정 벡터 중 비잔틴 노드로 판단된 요소의 수를 나타낸

다. (그림 2)에서는 상위 세 개의 그래프는 사이클 1, 4, 7에서의 결정 벡터 요소의 수를 나타내며, 하위 세 개의 그래프는 사이클 10, 13, 16에서의 결정 벡터 요소의 수를 나타낸다.

(그림 1)의 알고리즘은 기본적인 가쉽 기반 메시지 전파 알고리즘과 다음과 같은 상이점을 가지고 있다. 정상적인 노드가 정상적인 노드를 가쉽 타겟으로 선택한 경우에는 일반적인 가쉽 알고리즘과 비슷한 방법으로 메시지 교환을 하는 반면 정상적인 노드가 비잔틴 노드를 선택한 경우에는 가쉽 타겟이 비잔틴 노드로 판단한 후에는 정상적인 노드는 더 이상 비잔틴 노드와 메시지 교환을 하지 않는다.

반대로 비잔틴 노드가 가쉽 타겟으로 정상적인 노드를 선택한 경우에는 비잔틴 노드는 정상적인 노드와 메시지 교환을 하되 가쉽 타겟으로 선택된 정상적인 노드는 메시지 교환을 하지 않는다. 또한 비잔틴 노드가 가쉽 타겟으로 비잔틴 노드를 선택한 경우에는 두 노드 모두 서로 비잔틴 노드인 것을 확인한 이후 메시지 교환을 하지 않는다.

따라서 일반적인 가쉽 기반의 메시지 교환 알고리즘과 대조한다면 본 논문에서 제시한 비잔틴 합의 알고리즘이 일반적인 가쉽 메시지 교환 알고리즘보다 요구하는 사이클의 수가 더 많음을 짐작할 수 있다.

클라우드 컴퓨팅 환경과 같이 노드의 가입과 탈퇴가 자유로운 동적인 환경에서 가쉽 알고리즘을 사용을 때 나타날 수 있는 장점은 다음과 같다. 브로드캐스트를 사용하는 기존의 알고리즘에서 발생하는 병목현상이 발생하지 않으며 따라서 노드의 수에 대하여 확장적이다. 추가적으로 제안하는 알고리즘에서는 비잔틴 노드의 판단을 위해 위조-증명 서명을 사용하여 $N \geq 3f + 1$을 만족해야 하는 요구사항을 충족시키지 않아도 된다. (여기서 N은 전체 노드의 수, f는 비잔틴 노드의 수를 의미한다.)

제안하는 알고리즘의 또 다른 장점은 트리와 같은 구조를 사용하는 구조적인 알고리즘에 비해 결함 포용적이라는 것이다. 다시 말해 시스템 구성을 트리와 같은 구조로 구성하였다면 노드의 실패나 탈퇴가 발생하였을 때 트리 구조를 재구성해야 하며 이로 인한 오버헤드가 발생하게 된다. 하지만 가쉽 알고리즘은 비구조적인 알고리즘으로 노드의 가입과 탈퇴 그리고 노드 실패로 인하여 토폴로지 구조를 재구성할 필요가 없다.

결과적으로 기존의 비잔틴 합의 알고리즘에서 노드의 가입과 탈퇴 그리고 노드 실패가 발생하였을 때에는 비잔틴 합의 알고리즘 같은 경우에는 알고리즘을 재시작 해야 하는 단점으로 인해 총괄적인 메시지 복잡도가 증가하는 반면 제안하는 비잔틴 합의 알고리즘은 결함 포용적 특성을 내재하고 있는 가쉽 알고리즘을 사용함으로 인해 동적인 환경에서도 알고리즘을 재시작하지 않아도 되는 이점이 있다.

5. 결론

본 연구에서는 동적인 특정을 지닌 클라우드 컴퓨팅 환경에서 비잔틴 합의 문제를 해결하기 위한 가쉽 알고리즘을 제안하였다. 제안하는 알고리즘은 완전 연결 네트워크를 가정하지 않고 상대적으로 단순한 오버레이를 갖는 환경에서 가쉽 알고리즘을 통해 비잔틴 합의를 이룰 수 있으며, 따라서 기존 알고리즘에 비해 확장적이며 병목현상을 가지지 않는다. 성능 평가에서 제안하는 알고리즘이 효율적으로 비잔틴 합의에 도달할 수 있음을 보였다.

참고문헌

[1] Ganesh, A. J., Kermarrec, A. M., & Massoulie, L. (2003). Peer-to-peer membership management for gossip-based protocols. Computers, IEEE Transactions on, 52(2), 139-149.

[2] B. Pfitzmann and M. Waidner. (1991). Fail-stop signatures and their applications. In Proceedings of SECURICOM'91, 338-350.

[3] Montresor, A., & Jelasity, M. (2009). PeerSim: A scalable P2P simulator. Paper presented at the Peer-to-Peer Computing, 2009. P2P '09. IEEE Ninth International Conference on.

부록 A

TTP(trusted third party)는 $p-1 = 2q$를 만족하는 소수 p와 $g \in Z_p$ 그리고 임의의 정수 $r \in Z^*_q$ 을 선택한다. (여기서 q는 소수이다.) 다음으로 $R = g^r$을 계산하고 (p, q, g, R)을 노드 A에게 보낸다. (r은 TTP에서 보관한다.)

노드 A는 $x = (a_1, a_2, b_1, b_2) \in Z_q$를 선택하고 다음을 계산한다.

$R \equiv g^r \pmod p$,

$A \equiv g^{a1}R^{a2} \pmod p$,

$B \equiv g^{b1}R^{b2} \pmod p$

노드 A는 $K = (g, p, R, A, B)$를 자신의 공개키로 두며, x는 개인키로 둔다.

- 서명: 메시지 m에 대하여 다음을 생성한다.
  $s = SG_x(m) = (\beta_1, \beta_2)$
  (여기서 $\beta_1 \equiv a_1 + mb_1 \pmod q$이고, $\beta_2 \equiv a_2 + mb_2 \pmod q$ 이다.)
- 증명: 공격자는 특정 노드의 서명 $s = (\beta_1, \beta_2)$와 메시지 m 그리고 공개키 K를 획득한 후 다음을 검사한다.
  $VER_K(m,s) = (AB^m \equiv g^{\beta1}R^{\beta2} \pmod p)$
- 위조 증명: 특정 노드가 자신의 위조된 서명 $s' = (\beta'_1, \beta'_2)$을 발견하였을 때 다음을 계산한다.
  $PROOF(s') \equiv (\beta_1 - \beta'_1)(\beta'_2 - \beta_2)^{-1} \pmod q$