https://doi.org/10.7840/kics.2023.48.2.249

행렬 곱 연산을 위한 분산 부호화 컴퓨팅에서 집단 검증 기반의 악의적 공격 탐지 기법

홍 상 우◆, 양 희 철°, 윤 영 석*, 이 정 우*

Byzantine Attack Identification via Group Testing Approach in Coded Distributed Computing for Matrix Multiplication

Sangwoo Hong◆, Heecheol Yang°, Youngseok Yoon*, Jungwoo Lee*

요 약

여러 컴퓨팅 노드를 활용하는 분산 컴퓨팅 환경에서 분산 부호화 컴퓨팅은 낙오 효과 감소, 데이터 보안 보장 등을 위해 활용되어 왔다. 그러나 많은 컴퓨팅 노드를 활용하는 분산 컴퓨팅의 특성상 컴퓨팅 노드 중 잘못된 연산 결과를 보내 전체 연산을 망치고자 하는 악의적 공격을 수행하는 비잔틴 워커가 존재할 수 있다. 본 논문에서는 행렬 곱 연산을 위한 분산 부호화 컴퓨팅에서 악의적으로 잘못된 결과를 반환하는 비잔틴 워커가 존재할 때, 이를 효과적으로 탐지하고 올바른 전체 연산 결과를 복호화하는 기법을 소개한다. 우선 악의적 공격 탐지를 위한 새로운 분산 부호화 컴퓨팅 기법을 소개하고, 이에 적합한 두 가지 집단 검증 기법을 제안한다. 또한, 분산 컴퓨팅 환경을 제공하는 아마존 웹 서비스에서 비잔틴 워커의 존재 하에 행렬 곱 연산을 수행하는 실험을 통해 비잔틴 워커 탐지에 소요되는 시간을 밝힌다.

키워드 : 분산 컴퓨팅, 분산 부호화 컴퓨팅, 비잔틴 공격
Key Words : Distributed computing, Coded distributed computing, Byzantine attack

ABSTRACT

In a distributed computing system utilizing multiple computing nodes, coded distributed computing has been proposed to reduce straggler effects, ensure data privacy, and so on. However, due to the characteristic of distributed computing systems where many computing nodes called workers are utilized, there may exist Byzantine workers that carry out malicious attacks by sending wrong computation results to contaminate the overall computation results. In this paper, for the case where some Byzantine workers that return wrong computation results exist in coded distributed computing for matrix multiplication tasks, we introduce a new coded distributed computing scheme for effective identifying Byzantine workers and decoding the overall computation results. First, we suggest a new coded distributed computing technique for identifying Byzantine workers, and propose two group testing approaches suitable for the proposed coded distributed computing scheme. In addition, we show the Byzantine worker identification time to identify Byzantine workers via experiments that perform matrix multiplication tasks in the presence of Byzantine workers in Amazon Web Service (AWS) distributed computing systems.

※ 본 연구는 한국연구재단 이공분야기초연구사업(2022R1C1C1005749) 지원으로 수행되었습니다.
◆ First Author : Seoul National University, Department of Electrical and Computer Engineering, tkddn0606@snu.ac.kr, 학생회원
° Corresponding Author : Chungnam National University, Division of Computer Convergence, hcyang@cnu.ac.kr, 종신회원
* Seoul National University, Department of Electrical and Computer Engineering, youngseok8@snu.ac.kr, 학생회원; junglee@snu.ac.kr, 종신회원
논문번호 : KICS2022-09-227, Received October 11, 2022; Revised December 8, 2022; Accepted December 8, 2022

Ⅰ. 서 론

분산 컴퓨팅은 다수의 컴퓨팅 노드 또는 워커(worker)를 활용하여 연산을 병렬적으로 처리하는 기법으로, 최근에는 머신러닝 또는 딥러닝에 필요한 행렬 곱 연산을 비롯한 대규모 연산을 효과적으로 처리하기 위해 사용된다. 분산 컴퓨팅 시스템은 주로 하나의 센터 노드 또는 마스터(master)가 다수의 워커에 연산을 할당하고 이에 대한 결과를 받아 최종적으로 원하는 연산 결과를 얻는다. 이러한 분산 컴퓨팅 시스템에서 마스터에게 할당받은 연산의 결과를 반환하지 못하거나 다른 워커보다 느리게 연산 결과를 반환하는 낙오 워커(straggler)가 존재하는 경우 시스템의 전체 연산이 지연되는 문제가 발생할 수 있다.[1] 이를 분산 컴퓨팅에서의 낙오 효과(straggler effects)라 한다. 낙오 효과를 완화하기 위해 최근 오류 정정 부호 기반의 분산 부호화 컴퓨팅 기법이 제안되었다.[2] 마스터가 워커에게 할당하는 연산의 인코딩에 오류 정정 부호를 활용하여 개별 워커에 할당되는 연산량은 증가하지만, 대신 낙오 워커의 연산 결과를 다른 워커의 연산 결과로부터 복구할 수 있어 낙오 효과를 감소시킬 수 있다. 특히, 행렬 곱 연산을 비롯한 선형 연산에 오류 정정 부호를 활용하여 낙오 효과를 감소함으로써 분산 컴퓨팅 시스템의 전체 연산 처리 성능을 향상시킬 수 있어 이에 대한 다양한 연구가 이루어지고 있다.[3-5]

한편, 분산 컴퓨팅 시스템은 다양한 유형의 보안 위협에 처할 수 있다. 특히 본 논문에서는 분산 컴퓨팅 시스템에 대한 공격을 위해 악의적 의도를 가지고 워커로 참여하거나 일부 워커에 대한 접근을 통해 잘못된 연산 결과를 반환하는 공격을 수행하는 비잔틴 공격(Byzantine attack)을 고려하였다. 할당받은 연산에 대해 의도적으로 잘못된 결과를 반환함으로써 전체 연산 결과의 왜곡을 가져오도록 비잔틴 공격을 수행하는 워커를 비잔틴 워커(Byzantine worker)라 칭한다. 비잔틴 워커가 존재할 경우 분산 컴퓨팅 시스템의 신뢰성이 저하되고 올바른 연산 결과를 얻기 위한 성능 저하로 이어질 수 있다. 따라서, 이러한 비잔틴 공격에 대응하기 위해 일부 비잔틴 워커가 존재하더라도 나머지 워커의 올바른 연산 결과를 통해 전체 연산 결과를 정확히 복구할 수 있도록 하는 연산 인코딩 기법이 연구되고 있고, 한편으로는 다른 워커의 연산 결과와 비교하여 비잔틴 워커의 잘못된 연산 결과를 찾아내 비잔틴 워커를 탐지할 수 있도록 하는 연산 인코딩 기법과 비잔틴 워커 탐지 기법이 연구되고 있다. [6-10]

본 논문에서는 행렬 곱 연산을 수행하는 분산 컴퓨팅 시스템에서 비잔틴 워커를 탐지하는 기법에 대해 다룬다. 분산 컴퓨팅 시스템에 존재하는 비잔틴 워커들이 의도적으로 잘못된 연산 결과를 반환할 때, 다른 워커들의 연산 결과와의 비교를 통해 효과적으로 비잔틴 워커를 탐지하기 위해 효율적인 연산 업무 인코딩 기법을 소개한다. 또한, 제안된 연산 업무 인코딩 기법에 적합한 비잔틴 워커 탐지 기법을 두 가지 제안한다. 제안한 비잔틴 워커 탐지 기법의 성능을 검증하기 위해 행렬 곱 연산을 수행하는 분산 컴퓨팅 시스템을 가정하고 비잔틴 워커를 탐지하기 위해 소요되는 시간을 실험을 통해 측정하였다.

본 논문의 주요 성과는 두 가지로 아래와 같다.

- 본 논문에서는 효율적으로 비잔틴 워커를 탐지하기 위해 여러 연산 결과를 한번에 검증하는 집단 검증 기법을 소개한다. 이를 위해 집단 검증이 가능한 새로운 연산 업무 인코딩 기법에 적용될 수 있는 부호화 코드, 즉 부분적 검증 가능 코드를 제시한다. 부분적 검증 가능 코드를 활용하였을 때 선형적 복잡도로 비잔틴 워커를 찾아낼 수 있음을 밝히고, 기존에 제안된 Lagrange coded computing[5]에 비해 비잔틴 워커의 존재에 대해 두 배 강인한 시스템 성능을 보여줌을 밝힌다.
- 분산 컴퓨팅 시스템에서 워커에게 연산 업무를 할당하는 마스터가 시스템에 존재하는 비잔틴 워커의 수를 정확히 파악하고 있는 경우와 비잔틴 워커의 수를 모르고 있는 경우로 나누어 2가지 경우에 적용할 수 있는 집단 검증 스케줄링 기법을 제시한다. 또한, 2가지 집단 검증 스케줄링 기법의 성능을 실험적으로 검증한다. Amazon Web Service (AWS)의 Elastic Cloud Computer (EC2) 환경에서 여러 개의 가상 컴퓨팅 노드를 사용하여 행렬 곱 연산을 처리하는 분산 컴퓨팅 시스템을 설정하였다. 이 때 다양한 환경에서 집단 검증 기법의 성능을 파악하기 위해 분산 컴퓨팅 시스템에 존재하는 워커의 수, 비잔틴 워커의 수, 연산의 대상이 되는 행렬의 크기, 행렬을 분할하는 파라미터를 다르게 하여 비잔틴 워커를 탐지하는데 소요되는 시간을 측정하였다. 그 결과, 마스터가 비잔틴 워커의 수를 알고 있는 경우와 모르는 경우 모두 비잔틴 워커를 모두 탐지할 수 있음을 보이고, 비잔틴 워커의 수를 모르는 경우에도 적절한 집단 검증 스케줄링 기법을 사용하여 큰 성능 손실없이 비잔틴 워커를 탐지할 수 있음을 확인하였다.

본 논문에서는 우선 논문에서 가정하는 시스템 모

델을 밝히고, 집단 검증 기법을 활용하기 위한 연산 업무 인코딩 기법의 하나로 부분적 검증 가능 코드를 소개한다. 그리고 비잔틴 워커를 효율적으로 탐지하기 위해 부분적 검증 가능 코드에 적합한 2가지 집단 검증 스케줄링 기법을 제안한다. 마지막으로 집단 검증 스케줄링 기법의 성능을 파악하기 위해 행렬 곱 연산을 수행하는 다양한 분산 컴퓨팅 환경에서 비잔틴 워커 탐지에 소요되는 시간을 밝힌다.

Ⅱ. 본 론

2.1 시스템 모델

본 논문에서는 행렬 곱 연산을 수행하는 분산 부호화 컴퓨팅 시스템을 고려하였다. 마스터는 행렬 곱 연산 $C = AB$의 연산 결과를 얻기 위해 W개의 워커를 활용하며, 이를 위해 연산 업무를 개별 워커에게 할당한다. 각각의 워커 W_i는 할당된 연산 $\tilde{C}_i = \tilde{A}_i \times \tilde{B}_i$을 계산한다고 표현한다. 이 때, $\tilde{A}_i$와 $\tilde{B}_i$는 마스터가 워커 W_i에게 연산 업무를 할당하기 위해 입력 행렬 A와 B, 그리고 부호화 함수 p_A와 p_B를 사용하여 인코딩한 부호화 행렬이다. 마스터는 여러 워커로부터 충분한 수의 연산 결과를 받은 후에 비잔틴 워커 탐지를 시행한다. 비잔틴 워커를 모두 탐지하면 비잔틴 워커의 연산 결과를 제외하고 일반적인 워커의 정상적인 연산 결과를 이용해 복호화 과정을 수행하고, 이를 통해 최종 연산 결과인 $C = AB$를 도출해낸다. 구체적인 시스템 모델은 그림 1을 통해 묘사하였다.

본 논문에서는 분산 컴퓨팅 시스템에서 마스터가 활용하는 워커 중 비잔틴 워커가 존재하며, 마스터는 워커로부터 연산 결과를 받기 전까지 비잔틴 워커를 탐지하지 못한다고 가정한다. 또한, 마스터가 시스템 내에 존재하는 비잔틴 워커의 수를 알고 있는 경우와

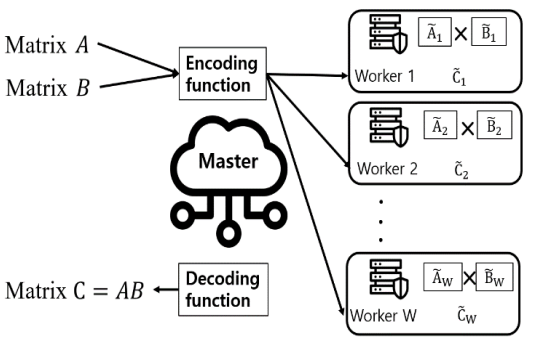

Fig. 1. Coded distributed computing for matrix multiplication tasks

사전 정보가 전혀 없어 비잔틴 워커의 수를 모르는 경우를 모두 가정한다. 비잔틴 워커는 최종 연산 결과를 왜곡하거나 전체 분산 컴퓨팅 시스템 성능을 저하시키기 위해 할당받은 연산 업무에 대해 의도적으로 잘못된 연산 결과를 반환한다. 또한 최악의 경우를 가정하여 비잔틴 워커들이 분산 컴퓨팅 시스템의 프로토콜과 연산의 대상이 되는 입력 행렬 A와 B에 대해 알고 있다고 가정한다. 한편, 비잔틴 워커들은 능동적인 공격을 위해 서로 공모할 수는 없다고 가정하여 자신들에게 할당된 연산 업무, 즉 할당된 부호화 행렬에 대한 정보를 서로 공유하지 않는다고 가정한다.

본 논문에서는 효율적인 비잔틴 워커 탐지를 위한 연산 업무 인코딩 기법, 즉 마스터가 부호화 행렬을 생성하기 위해 사용하는 부호화 함수 p_A, p_B를 소개하고, 부호화 함수 p_A, p_B를 통해 생성된 연산 업무에 대한 연산 결과로부터 전체 행렬 곱 연산의 결과인 $C = AB$를 도출하는 과정을 밝힌다. 또한, 빠르게 연산 결과를 도출하기 위해서는 비잔틴 워커를 효율적으로 탐지하는 것이 중요하므로 비잔틴 워커 탐지 시간을 최소화할 수 있는 연산 결과 검증 스케줄링 기법을 고안하는 것에 초점을 맞춘다.

2.2 부분적 검증 가능 코드

본 장에서는 마스터가 워커에게 연산 업무를할당할 때, 연산 업무를 인코딩하기 위해 사용하는 부분적 검증 가능 코드를 소개한다. [8] 부분적 검증 가능 코드는 우선 입력 행렬 A와 B를 아래와 같이 각각 행 방향과 열 방향으로 m개의 부행렬과 n개의 부행렬로 나누어 표현한 후에 부호화 함수에 입력으로 사용한다.

$$\mathrm{A} = \begin{bmatrix} A_1 \\ \vdots \\ A_m \end{bmatrix}, \qquad \mathrm{B} = [B_1 \quad \cdots \quad B_n]$$

또한, 행렬 곱 연산 인코딩을 위해 아래와 같은 조건을 만족하는 다항식 f(x)을 사용한다.

i) f(x)는 (m + 1)차 다항식이다.

ii) f(x)는 $s \in \{1,2,\ldots,\frac{W}{m+1}\}$에 대해 $f(\alpha_{s,1}) = \cdots = f(\alpha_{s,m+1})$을 만족하는 (m + 1)개의 점 집합 $\frac{W}{m+1}$개를 가진다.

이 때, 전체 워커를 $\frac{W}{m+1}$개 그룹으로 나누어 각 워커를 $W_{s,t}, s \in \left\{1,2,\ldots,\frac{W}{m+1}\right\}, t \in \{1,2,\ldots,m+1\}$

로 표기한다. 각 워커가 할당받은 부호화 행렬은 아래와 같이 표현된다.

$$\tilde{A}_{s,t} = \sum_{v=0}^{m-1} A_{v+1}\alpha_{s,t},$$

$$\tilde{B}_{s,t} = \sum_{z=0}^{n-1} B_{z+1} f^{z}(\alpha_{s,t}) = \sum_{z=0}^{n-1} B_{z+1}\beta_{s}^{z}.$$

각 워커가 계산하는 $\tilde{C}_{s,t} = \tilde{A}_{s,t} \times \tilde{B}_{s,t}$는 아래 다항식 p($x$)의 $x = \alpha_{s,t}$에서의 값과 같다.

$$p(x) = \sum_{v=0}^{m-1}\sum_{z=0}^{n-1} A_{v+1}B_{z+1}\alpha_{s,t}^{v} f^{z}(\alpha_{s,t}).$$

마스터는 워커로부터 받은 연산 결과인 $\tilde{C}_{s,t}$를 사용하여 다항식 p(x)를 보간(interpolation)하고, p(x)의 계수로부터 전체 연산 결과인 C의 부분블록행렬 $A_1B_1 \sim A_mB_n$을 추출할 수 있다. 따라서, 부분적 검증 가능 코드를 통해 마스터는 원하는 전체 연산 결과 $C = AB$을 복원할 수 있다. 이 때, 다항식 p(x)는 mn-1 차수의 다항식으로 보간을 위해서는 mn개의 결과값 $\tilde{C}_{s,t}$이 필요하다. 따라서 부분 검증 가능 코드를 활용할 때, 마스터가 전체 연산 결과를 얻기 위해 워커로부터 반환받아야 하는 최소 연산 결과 임계값은 mn이며, 이는 부분적 검증 가능 코드와 같은 방식으로 입력 행렬을 나누었을 때 최적의 값으로 밝혀져 있다.[3]

2.3 집단 검증 기법을 활용한 비잔틴 워커 탐지

2.3.1 집단 검증 기법

집단 검증 기법은 수학적 검출 기법으로써 큰 수의 집단에서 이상이 있는 구성원을 찾고자 할 때에 적은 수의 검증 횟수만으로 이상이 있는 구성원을 탐지할 수 있는 스케줄링 기법이다. 집단에 있는 구성원들은 개인적으로, 혹은 다른 구성원들과 함께 검증을 받을 수 있다. 대표적인 집단 검증 기법의 예시를 그림 2에 표현하였다. 이 예시에서는 8개의 구성원 중에 1개의 이상이 있는 구성원이 존재하는 경우를 가정하였고, 4개의 구성원이 한 번에 집단 검증을 받을 수 있다고 가정하였다. 만약 집단 검증 결과가 음성이면 해당 집단에는 이상이 있는 구성원이 존재하지 않음을 의미

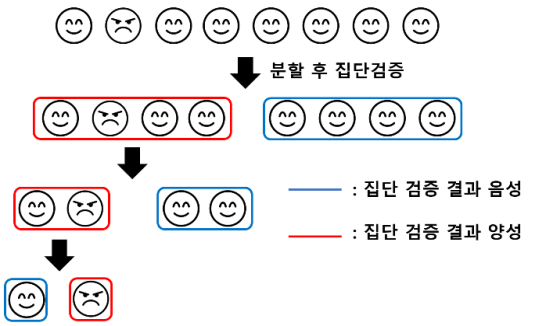

Fig. 2. An example for group testing

하며, 집단 검증 결과가 양성이면 해당 집단에 이상이 있는 구성원이 존재함을 의미한다. 이 경우, 이상이 있는 구성원을 찾기 위해 추가적인 검증이 필요하며 2개의 구성원을 집단으로 묶어 집단 검증을 수행한다. 그 결과 집단 검증 결과가 양성인 집단에 대해서는 개별적으로 검증을 실시하고 최종적으로 이상이 있는 구성원을 탐지할 수 있다. 이러한 집단 검증 스케줄링 기법을 통해 이상이 있는 구성원을 탐지할 경우 총 6번의 검증 횟수가 필요하다.

2.3.2 분산 부호화 컴퓨팅에서 집단 검증 기법을 통한 비잔틴 워커 탐지

분산 부호화 컴퓨팅 시스템에서는 마스터가 적절한 부호화 함수를 사용하여 개별 워커에게 할당할 연산 업무에 해당하는 부호화 행렬을 생성한다. 이러한 부호화 행렬의 역행렬을 집단 검증 행렬로 정의하는 경우 집단 검증 기법을 실시하기 위해서는 최소한 마스터가 전체 연산 결과를 복구하기 위해 필요로 하는 최소 연산 결과 임계값보다 많은 연산 결과를 워커로부터 반환받아야 하는 것이 알려져 있다.[7] 이 경우, 이와 같이 집단 검증 기법을 실시하기 위한 최소한의 집단 크기가 존재하는 경우 더 작은 수의 집단에 대한 검증이 불가하여 효율적인 비잔틴 워커 탐지를 수행하지 못하게 된다. 하지만 부분적 검증 가능 코드를 사용하여 부호화 행렬을 생성하는 경우 다항식 f(x)가 만족하는 조건 ii)에 따라 $\alpha_{s,1} \sim \alpha_{s,m+1}$을 사용하여 만든 Vandermonde 행렬의 역행렬을 복호화 행렬로 사용함으로써 $m+1$개 이상의 연산 결과에 대해 집단 검증 기법을 실시할 수 있다.[8] 즉, 부분적 검증 가능 코드를 사용할 때 마스터에서 전체 연산 결과를 복구하기 위해 필요로하는 최소 연산 결과 임계값 mn보다 작은 수의 연산 결과에 대해 집단 검증 기법을 실시할 수 있어 전체 검증 횟수를 감소시킬 수 있고 이에 따

라 집단 검증의 효율성을 높일 수 있다. 집단 검증을 수행할 경우 같은 집단에 속한 비잔틴 워커들이 서로 공모할 수 있다면 반환하는 연산 결과를 서로 합의하여 집단 검증 결과는 음성이지만 잘못된 개별 연산 결과를 반환할 수 있는 여지가 있다. 하지만, 본 논문에서는 비잔틴 워커들이 서로 공모할 수 없다고 가정하며, 이에 따라 집단 검증을 수행하여 연산 결과를 반환한 워커 중 비잔틴 워커를 항상 검출할 수 있는 상황을 가정한다.

본 논문에서는 마스터가 분산 컴퓨팅 시스템에 존재하는 비잔틴 워커의 수를 사전에 알고 있는 경우와 모르는 경우 두 가지에 적용할 수 있는 집단 검증 기법을 소개한다.

1) 마스터가 비잔틴 워커의 수를 알고 있는 경우의 집단 검증 기법[8]

마스터가 비잔틴 워커의 수를 알고 있는 경우 정보이론적으로 Dorfman의 방법이 가장 최적이라는 것이 알려져 있다.[11] 따라서, 부분적 검증 가능 코드 역시 비잔틴 워커의 수가 알려져 있을 경우 Dorfman의 방법을 일부 차용하여 사용한다. Dorfman의 방법을 일부 차용한 집단 검증 스케줄링 기법은 두 단계로 진행되는데, 첫번째 단계에서는 전체 워커가 W개이고 비잔틴 워커의 수가 l개일 때 전체 워커를 $\sqrt{W/l}$개의 작은 집단으로 나누어서 검증한다. 그리고, 두번째 단계에서는 첫번째 단계에서 집단 검증 결과가 양성인 집단에 대해 개별 워커의 연산 결과를 하나씩 검증하는 과정을 거친다.

2) 마스터가 비잔틴 워커의 수를 모르는 경우의 집단 검증 기법[9]

마스터가 비잔틴 워커의 수를 모르는 경우에는 Dorfman의 방법을 사용할 수 없기에 효율적인 집단 검증 스케줄링 기법 중 하나인 Binary splitting 기법을 사용할 수 있다.[12] Binary splitting 기법에서는 전체 집단에 대해 검증을 수행한 후, 그 결과 비잔틴 워커가 존재하는 것으로 판단되면 집단을 같은 크기의 두 소집단으로 분할하여 각 소집단에 대한 검증을 실시한다. 해당 과정을 반복 수행하여 검증 대상 집단의 크기가 1이 되거나 모든 검증 결과가 음성이 될 때 까지 반복한다. 이를 통해 비잔틴 워커의 수를 모르더라도 모든 비잔틴 워커를 탐지할 수 있다.

Ⅲ. 실 험

본 장에서는 논문에서 소개하는 집단 검증 기반 비잔틴 워커 탐지 기법과 기존에 제시되었던 분산 부호화 컴퓨팅에서의 비잔틴 워커 탐지 기법의 이론적 성능을 비교한다. 또한, 실제 행렬 곱 연산을 수행하는 분산 컴퓨팅 시스템에서 비잔틴 워커 탐지를 위해 소요되는 시간을 실험을 통해 밝힌다.

집단 검증 기반 비잔틴 워커 탐지 기법과 기존에 제안된 Lagrange coded computing 기법의 비잔틴 워커의 공격에 대한 강인함과 탐지 계산 복잡도를 표 1에 나타내었다.

일반적으로 비잔틴 공격이 존재할 때 이에 대한 강인함을 비교하는 척도로 마스터가 올바른 전체 연산 결과를 얻기 위해 필요로 하는 추가적인 연산 결과의 수를 사용한다. 마스터가 필요로 하는 추가적인 연산 결과의 수가 많을 경우 더 많은 워커의 연산 결과를 얻어야 하기 때문에 그보다 적은 연산 결과를 얻을 경우 올바른 전체 연산 결과를 얻을 수 없어 비잔틴 공격에 대해 강인하지 않다고 판단한다. 본 논문에서 소개하는 집단 검증 기반의 비잔틴 워커 탐지 기법은 기존 분산 부호화 컴퓨팅 시스템에서 비잔틴 공격에 강인하고 비잔틴 워커를 탐지하기 위해 제안된 Lagrange coded computing과 비교해 비잔틴 공격에 더 강인하다고 할 수 있다. 집단 검증 기반의 비잔틴 워커 탐지 기법에서는 마스터가 비잔틴 워커의 존재하에 정상적인 전체 연산 결과를 얻기 위해 비잔틴 워커의 수와 같은 수의 일반적인 워커의 연산 결과를 추가로 필요로 한다. 반면 Lagrange coded computing에서는 마스터가 정상적인 전체 연산 결과를 얻기 위해서는 비잔틴 워커보다 2배 더 많은 수의 일반적인 워커의 연산 결과를 추가로 필요로 하여, 많은 수의 비잔틴 워커가 존재할 경우 정상적인 전체 연산 결과를

Table 1. Comparison between Lagrange coded computing and group testing based Byzantine attack identification

	비잔틴 공격에 대한 강인함	비잔틴 공격 탐지를 위한 계산 복잡도
Lagrange coded computing [1]	L 비잔틴 워커에 대해 2L개의 추가적인 논-비잔틴 워커의 연산 결과 필요	$O(W^2)$
집단 검증 기반 비잔틴 워커 탐지	L 비잔틴 워커에 대해 L개의 추가적인 논-비잔틴 워커의 연산 결과 필요	$O(W)$ (when $W \gg 1$)

얻기 위해 더 많은 워커의 연산 결과를 기다려야 함을 알 수 있다. 또한, Lagrange coded computing에서는 비잔틴 워커 탐지를 위한 계산 복잡도가 전체 워커의 수의 제곱에 비례하여 증가하는 반면 집단 검증 기반의 비잔틴 워커 탐지 기법에서는 전체 워커의 수에 비례하여 계산 복잡도가 증가해 워커의 수가 많은 분산 컴퓨팅 시스템에서 더 적합하다고 할 수 있다. 또한 2장에 따르면 부분적 검증 가능 코드를 연산 부호화에 적용함으로써 집단 검증을 위한 최소 검증 단위를 낮출 수 있다. 이는 집단 검증을 통해 모든 비잔틴 워커를 탐지하기 위한 검증 횟수를 크게 줄이는 것에 기여할 수 있고, 검증 횟수 감소에 의한 성능 향상 효과는 실험에서 확인할 수 있다. 추가적으로 입증가능한 컴퓨팅 (verifiable computing) 기반의 비잔틴 워커 탐지를 비교 대상으로 고려할 수 있다. 대표적으로 입증가능한 컴퓨팅에서 낙오 효과를 고려하기 위해 제안된 adaptive verifiable coded computing(AVCC) 기법이 있다. [13] AVCC의 경우 입증 키(verification key)를 기반으로 개별 워커의 연산 결과에 대해 비잔틴 공격 여부를 판별할 수 있어 집단 검증 기반의 비잔틴 워커 탐지 기법과 마찬가지로 마스터가 정상적인 전체 연산 결과를 얻기 위해 비잔틴 워커의 수와 같은 수의 일반적인 워커의 연산 결과를 추가로 필요로 한다. 따라서, AVCC도 비잔틴 공격에 강인한 특성을 가진다고 할 수 있다. 하지만, AVCC의 경우 워커에 연산 업무를 할당하기 전에 입증 키를 생성하고 워커의 연산 결과에 대해 입증 키를 통한 검증 과정을 추가로 필요로 해 본 논문에서 소개한 기법과의 직접적인 정량적 비교는 다루지 않는다.

다음으로 본 논문에서 소개한 집단 검증 기반의 비잔틴 워커 탐지 기법의 성능을 확인하기 위해 비잔틴 워커 탐지에 소요되는 시간을 측정하는 실험을 수행하였다. 실험은 분산 컴퓨팅 환경을 제공하는 AWS EC2에서 진행되었고 t2.2Xlarge 노드를 마스터로, t2.micro 노드들을 워커로 사용하였다. 마스터에서 워커로 부분적 검증 가능 코드를 통한 연산 업무의 할당은 분산 컴퓨팅 구현에서 주로 사용되는 MPI4py를 이용하였다.[14] 총 4가지의 실험 파라미터 변화에 따른 비잔틴 워커 탐지 소요 시간을 측정하였으며 4개의 실험에 대한 구체적인 파라미터 세팅은 표 2와 같다.

그림 3은 분산 컴퓨팅 시스템에 존재하는 전체 워커 수의 변화에 따른 비잔틴 워커 탐지 시간을 표시하였다. (실험 1) 전체 워커의 숫자가 적을 때에는 마스터가 비잔틴 워커의 수를 사전에 알지 못할 때 활용하는 binary splitting 알고리즘 (이하 Binary)과 마스터

Table 2. Parameter settings

	m	n	l	W	A	B
실험 1	4	4	3	-	3200 × 3200	3200 × 3200
실험 2	4	4	-	30	3200 × 3200	3200 × 3200
실험 3	4	4	3	30	3200 × 3200	3200 × 3200
					6400 × 1200	1200 × 6400
					2000 × 8000	8000 × 2000
실험 4	-	-	5	60	300 × 300	300 × 300

가 비잔틴 워커의 수를 알고 있을 때 활용하는 Dorfman의 방법 (이하 Dorfman)이 비슷한 탐지 소요 시간을 가지는 것을 확인할 수 있다. 하지만 전체 워커의 수가 늘어남에 따라 Dorfman이 Binary에 비해 더 짧은 탐지 소요 시간을 가지는 것을 볼 수 있다. 이는 전체 워커의 수가 많을수록 정보이론적으로 최적의 집단 검증 스케줄링을 수행하는 Dorfman과 일반적으로 우수한 성능을 보여줄 수 있는 Binary 사이의 성능 격차가 커지는 것으로 이해할 수 있다. 또한 한 가지 살펴볼 점은 전체 워커의 수가 늘어남에 따라 Binary와 Dorfman 모두 비잔틴 워커 탐지에 소요되는 시간이 감소한다는 것이다. 이는 분산 부호화 컴퓨팅에서 집단 검증 기법을 사용하기 위해 존재하는 최소 검증 단위에 의한 것으로 집단 검증의 경우 전체 워커 중 비잔틴 워커의 비율이 낮을수록 비잔틴 워커 탐지에 소요되는 시간이 줄어들기 때문으로 해석할 수 있다.

그림 4는 분산 컴퓨팅 시스템에 존재하는 비잔틴 워커 수의 변화에 따른 비잔틴 워커 탐지 시간을 표시하였다. (실험 2) 비잔틴 워커의 수가 늘어남에 따라 비잔틴 워커를 탐지하기 위해 소요되는 시간이 증가하는 것을 볼 수 있다. 또한, 비잔틴 워커의 수가 늘어

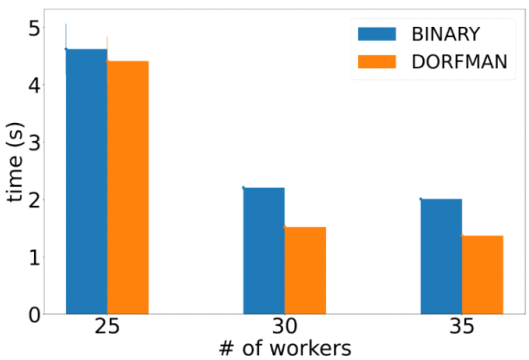

Fig. 3. Byzantine worker identification time for different number of workers

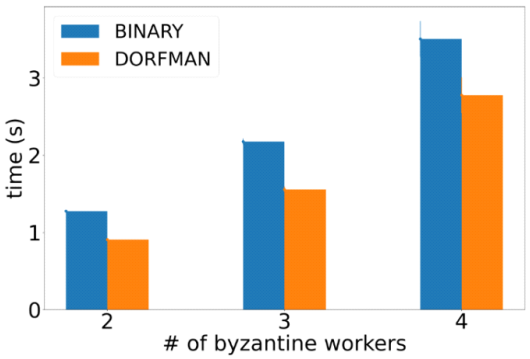

Fig. 4. Byzantine worker identification time for different number of Byzantine workers

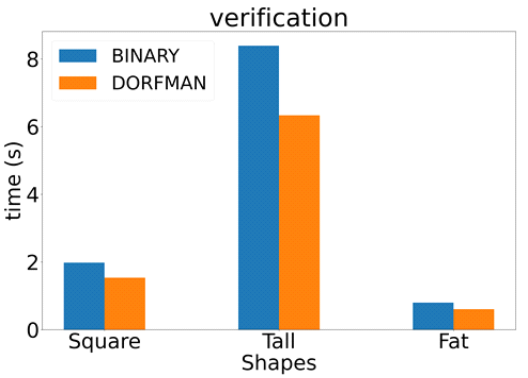

Fig. 5. Byzantine worker identification time for different shapes of input matrices

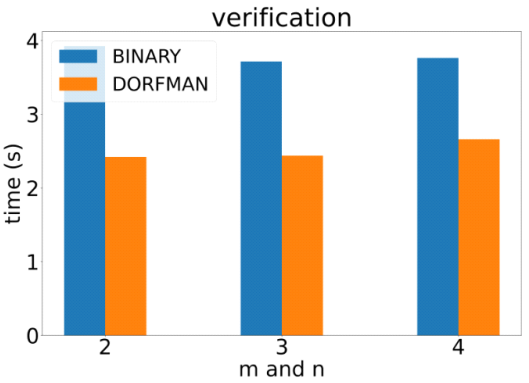

Fig. 6. Byzantine worker identification time for different matrix partitioning parameters

날수록 Binary와 Dorfman의 성능 격차 역시 커지는 것을 확인할 수 있다.

그림 5는 입력 행렬 A와 B의 형태에 따른 비잔틴 워커 탐지 소요 시간을 표시하였다. (실험 3)모든 경우에 대해 입력 행렬의 형태와 상관없이 Dorfman이 Binary에 비해 우수한 성능을 보이는 것을 확인할 수 있다. 2장에 따르면 비잔틴 워커를 탐지하기 위한 집단 검증 기법을 수행할 때, 워커의 연산 결과를 이어 붙인 행렬의 역행렬을 복호화에 사용하는데, 이에 필요한 연산량에 차이가 있어 행렬의 형태에 따라 비잔틴 워커를 탐지하는데 소요되는 시간이 큰 차이가 나는 것을 볼 수 있다. 비잔틴 워커를 탐지하는데 소요되는 시간이 큰 입력 행렬의 형태를 가질수록 Dorfman과 Binary의 성능 격차 또한 커지는 것을 확인할 수 있다.

마지막으로 그림 6은 입력 행렬 A와 B에 대한 행렬 분할 파라미터 m과 n의 차이에 따른 비잔틴 워커 탐지 소요 시간을 표시하였다. (실험 4) 앞선 실험과 마찬가지로 모든 경우에 대해 Dorfman이 Binary보다 우수한 성능을 보이는 것을 알 수 있다. 행렬 분할 파라미터 m과 n이 작을수록 개별 워커가 할당받은 부호화 행렬의 크기가 커서 워커의 개별 연산량이 많다고 할 수 있고, 대신 마스터의 복구 가능 임계치가 낮아 더 적은 수의 워커 연산 결과로부터 전체 연산 결과를 얻을 수 있다. 행렬 분할 파라미터 m이 작으면 부분적 검증 가능 코드를 활용할 때의 집단 검증 최소 단위 또한 작아 집단 검증 횟수가 낮아질 수 있다. 하지만 행렬 분할 파라미터가 변함으로써 마스터의 복구 가능 임계치를 비롯하여 비잔틴 워커 탐지 소요 시간에 영향을 줄 수 있는 다양한 값들이 변할 수 있다. 이에 따라 3가지 다른 행렬 분할 파라미터에 대해 비잔틴 워커 탐지에 소요되는 시간은 큰 변화가 없는 것을 확인할 수 있다.

Ⅳ. 결 론

본 논문에서는 행렬 곱 연산을 수행하는 분산 부호화 컴퓨팅 환경에서 악의적으로 잘못된 결과를 반환하는 비잔틴 워커를 탐지하기 위한 집단 검증 기법을 다루었다. 기존에 분산 부호화 컴퓨팅에서 비잔틴 워커를 탐지하기 위해 제안된 기법에 비해 비잔틴 공격에 더 강인하고 낮은 계산 복잡도를 가지는 것을 확인하였다. 나아가 실제 분산 컴퓨팅 환경에서의 다양한 실험을 통해 비잔틴 워커 탐지에 소요되는 시간을 확인하였다. 마스터가 시스템에 존재하는 비잔틴 워커의 수를 알고 있는 경우와 모르고 있는 경우 각각에 활용할 수 있는 집단 검증 스케줄링 기법을 소개하고 각각에 대한 비잔틴 워커 탐지 성능을 실험을 통해 보였다.

References

[1] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74-80, Feb. 2013. (https://doi.org/10.1145/2408776.2408794)

[2] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Trans. Inf. Theory*, vol. 64, no. 3, pp. 1514-1529, Mar. 2018. (https://doi.org/10.1109/TIT.2017.2736066)

[3] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Polynomial codes: An optimal design for high-dimensional coded matrix multiplication," *32th Annu. Conf. NIPS*, pp. 4403-4413, Long Beach, CA, Dec. 2017. (https://doi.org/10.5555/3294996.3295194)

[4] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: fundamental limits and optimal coding," *IEEE Trans. Inf. Theory*, vol. 66, no. 3, pp. 1920-1933, Mar. 2020. (https://doi.org/10.1109/TIT.2019.2963864)

[5] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. A. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," *The 22nd Int. Conf. Artificial Intell. and Statistics* (AISTATS), pp. 1215-1225, Okinawa, Japan, Apr. 2019. (https://proceedings.mlr.press/v89/yu19b.html)

[6] M. Soleymani, R. E. Ali, H. Mahdavifar, and A. S. Avestimehr, "List-decodable coded computing: Breaking the adversarial toleration barrier," *IEEE ISIT*, pp. 1206-1211, Melbourne, Australia, Jul. 2021. (https://doi.org/10.1109/ISIT45174.2021.9518044)

[7] A. Solanki, M. Cardone, and S. Mohajer, "Non-colluding attacks identification in distributed computing," *IEEE ITW*, pp. 1-5, Visby, Sweden, Sep. 2019. (https://doi.org/10.1109/ITW44776.2019.8989337)

[8] S. Hong, H. Yang, and J. Lee, "Hierarchical group testing for byzantine attack identification in distributed matrix multiplication," *IEEE J. Sel. Areas in Commun.*, vol. 40, no. 3, pp. 1013-1029, Mar. 2022. (https://doi.org/10.1109/JSAC.2022.3142364)

[9] S. Hong, H. Yang, and J. Lee, "Byzantine attack identification in distributed matrix multiplication via locally testable codes," *IEEE ISIT*, pp. 560-565, Espoo, Finland, Jun. 2022. (https://doi.org/10.1109/ISIT50566.2022.9834271)

[10] S. Hong, H. Yang, W. Shin, and J. Lee, "Byzantine attack identification via group testing approach in coded distributed computing," in *Proc. KICS Winter Conf.*, pp. 1-2, Feb. 2022.

[11] R. Dorfman, "The detection of defective members of large populations," *The Annals of Math. Statist.*, vol. 14, no. 4, pp. 436-440, Apr. 1943. (https://doi.org/10.1214/aoms/1177731363)

[12] M. Sobel and P. A. Groll, "Binomial group-testing with an unknown proportion of defectives," *Technometrics*, vol. 8, no. 4, pp. 631-656, Apr. 1966. (https://doi.org/10.2307/1266636)

[13] T. Tang, R. E. Ali, H. Hashemi, T. Gangwani, S. Avestimehr, and M. Annavaram, "Adaptive verifiable coded computing: Towards fast, secure and private distributed machine learning," in *Arxiv, arXiv:2107.12958v2*, Mar. 2022. (https://doi.org/10.48550/arXiv.2107.12958v2)

[14] L. Dalcín, R. Paz, and M. Storti, "MPI for Python," *J. Parall. and Distrib. Computing*, vol. 65, no. 9, pp. 1108-1115, Sep. 2005. (https://doi.org/10.1016/j.jpdc.2005.03.010)

홍 상 우 (Sangwoo Hong)

2020년 2월 : 서울대학교 전기정보공학부 졸업
2020년 3월~현재 : 서울대학교 전기정보공학부 석박 통합과정
<관심분야> 머신러닝, 분산컴퓨팅
[ORCID: 0000-0002-0270-2781]

양 희 철 (Heecheol Yang)

2013년 2월 : 서울대학교 전기정보공학부 졸업
2018년 2월 : 서울대학교 전기컴퓨터공학부 박사
2021년 3월~현재 : 충남대학교 컴퓨터융합학부 조교수
<관심분야> 무선통신, 머신러닝, 분산컴퓨팅
[ORCID:0000-0002-2802-2143]

윤 영 석 (Youngseok Yoon)

2020년 2월 : 서울대학교 전기정보공학부 졸업
2021년 3월~현재 : 서울대학교 전기정보공학부 석사과정
<관심분야> 검출 및 추정이론, 신호이론, 인공지능 소사이어티

이 정 우 (Jungwoo Lee)

1988년 : 서울대학교 전기공학 학사
1990년 : Princeton University 전기공학 석사
1994년 : Princeton University 전기공학 박사
2002년~현재 : 서울대학교 전기정보공학부 교수
<관심분야> 무선통신, 정보이론, 머신러닝