고성능·비잔틴 내성을 위한 RAFT-PBFT 하이브리드 합의 구조의 설계 및 검증

(Design and Verification of a RAFT-PBFT Hybrid Consensus Architecture for High Performance and Byzantine Fault Tolerance)

이지운*, 서희석**
(Ji Woon Lee, Hee Suk Seo)

요 약

분산 시스템에서 높은 처리 성능과 비잔틴 장애 내성을 동시에 만족시키기 위한 하이브리드 합의(Hybrid Consensus) 구조를 제안한다. 제안된 구조는 다수결 기반의 RAFT 알고리즘을 이용하여 각 샤드(shard) 내에서 빠른 초기 합의(temporary agreement)를 수행하고, 소수의 대표 노드로 구성된 PBFT 전역 그룹에서 최종 승인(final approval)을 수행하는 2단계 파이프라인으로 이루어진다. 이를 통해 정상 상황에서는 RAFT 단독 실행과 유사한 높은 처리율과 저지연을 달성하면서, 악의적 노드가 존재할 때는 PBFT 단계가 무결성을 보장한다. 형식적 검증을 위해 Z3/PySMT를 이용한 형식적 검증을 통해 안전성(safety)과 활성(liveness)을 모두 충족함을 확인하였다. 본 연구는 RAFT와 PBFT의 장점을 통합한 새로운 합의 패러다임을 제시하고, 허가형 블록체인·금융 플랫폼·분산 데이터베이스 등 보안과 성능이 모두 중요한 응용 분야에 실용적 대안이 될 수 있음을 보였다.

■ 중심어 : 하이브리드 합의 ; 이중 합의 구조 ; 비잔틴 합의 ; 형식적 검증 ; 블록체인

Abstract

We propose a hybrid consensus framework that simultaneously achieves high throughput and Byzantine fault tolerance in distributed systems. The proposed structure operates as a two stage pipeline: it first uses a majority voting RAFT protocol within each shard to obtain a rapid temporary agreement, and then relies on a compact PBFT committee of representative nodes to conduct a global final approval. Under normal conditions, this approach delivers throughput and latency comparable to standalone RAFT, while in the presence of malicious nodes the PBFT stage enforces transaction integrity. Formal verification using Z3/PySMT confirms that both safety and liveness properties are satisfied. Our study introduces a new consensus paradigm that unifies the efficiency of RAFT with the security of PBFT, demonstrating its practicality for permissioned blockchains, financial platforms, and distributed databases where both performance and fault tolerance are critical.

■ keywords : Hybrid Consensus ; Dual Consensus Architecture ; PBFT; Formal Verification ; Blockchain

Ⅰ. 서 론

분산 컴퓨팅 환경에서 여러 노드들이 하나의 일관된 상태를 유지하려면 합의 알고리즘(consensus algorithm)이 필수적이다[1]. 합의 알고리즘은 네트워크 지연이나 장애가 발생해도 모든 정직한 노드들이 동일한 값(또는 로그)을 커밋 하도록 보장함으로써 데이터의 무결성과 정합성을 유지한다. 특히 블록체인, 분산 데이터베이스, 분산 파일 시스템 등에서는 장애 내성과 성능(처리율·지연 시간)을 모두

* 정회원, 한국기술교육대학교 미래융합학부 기술연구원
** 정회원, 한국기술교육대학교 컴퓨터공학부 교수
이 논문은 2025년도 한국기술교육대학교 교수 교육연구진흥과제 지원에 의하여 수행된 연구임.
접수일자 : 2025년 04월 24일
게재확정일 : 2025년 05월 21일
교신저자 : 서희석 e-mail : histone@koreatech.ac.kr

만족하는 합의 프로토콜에 대한 요구가 높다. 오늘날 널리 사용되는 합의 알고리즘으로 RAFT와 PBFT가 대표적이다. RAFT는 Crash Fault 환경(정상 노드 장애만 존재하는 환경)에서 다수결 기반 리더 선출과 로그 복제를 통해 높은 처리율과 낮은 지연을 달성하는 프로토콜로, 구현이 간단하고 성능이 우수해 여러 분산 시스템(예: etcd, Consul 등)에 폭넓게 채택되고 있다[3]. 반면 PBFT(Practical Byzantine Fault Tolerance)는 비동기 네트워크에서 최대 f개의 비잔틴 장애(노드가 악의적이거나 임의로 잘못된 동작을 하는 경우)를 허용하면서도 안전성(safety)과 활성(liveness)을 보장하는 합의 알고리즘이다[1,4]. PBFT는 $3f+1$개의 노드 중 최대 f개가 악의적으로 행동해도 올바른 합의 결과를 도출하지만, 노드 수가 증가할수록 통신 오버헤드가 크게 늘어나 처리율이 급격히 저하되는 단점이 지적된다[1,5]. 다시 말해, RAFT는 성능 면에서 뛰어나지만 악의적인 노드가 있을 경우 로그 무결성을 보장하지 못하는 한계가 있고, PBFT는 높은 보안성을 제공하지만 확장성 면에서 제약이 있다. 이러한 상반된 특성 때문에 두 알고리즘은 성능 대 보안성 측면에서 트레이드 오프 관계를 가진다. 이 연구에서는 위 두 알고리즘의 장점을 결합하는 방안을 탐구한다. 구체적으로, RAFT의 빠른 합의 형성 능력은 유지하면서 PBFT의 비잔틴 장애 대응 능력을 접목하여, “정상 상황”에서는 RAFT에 준하는 속도를 내면서도 “비정상 상황”에서는 PBFT 수준의 안전성을 확보할 수 있는지 질문을 던진다. 이를 위해 RAFT를 이용해 노드 간 로그를 빠르게 복제하고 임시 합의를 얻은 뒤, PBFT를 통해 추가적인 서명 검증과 최종 승인(final commit)을 수행하는 하이브리드 합의 구조를 제안한다. 제안 방식에서는 RAFT가 우선 로그 합의를 완료하면 PBFT가 별도의 단계에서 해당 로그의 유효성을 검증하여 승인하는 2단계 파이프라인으로 동작한다. 이러한 하이브리드 합의(Dual Consensus) 절차를 통해 정상 시에는 RAFT 단독 실행 시와 거의 동일한 처리율을 유지하면서, 비잔틴 노드가 존재하는 경우에도 PBFT 단계가 안전장치 역할을 수행하여 시스템 전체의 무결성을 지킬 것으로 기대된다.

II. 관 련 연 구

RAFT 알고리즘은 전통적인 Crash Fault 환경을 가정한 합의 프로토콜로, 핵심 특징은 다음과 같다.

① 리더 기반 합의: 하나의 노드를 리더로 선출하여 클라이언트 명령을 대표로 처리하고 다른 팔로워(follower) 노드에 전파함으로써 합의를 수행한다. 로그 복제 과정에서 과반수 노드의 응답이 있으면 해당 로그를 커밋(committed) 처리하여 모든 노드에 반영하고, 리더 장애 시에는 재빠른 리더 교체(재선거)로 합의를 지속한다.

② 높은 성능과 활용도: 구현이 단순하고 직관적이라 개발 및 디버깅이 용이하며, etcd, Consul, TiKV 등 다양한 분산 시스템에 채택될 만큼 높은 처리율과 안정성을 보인다. 다만 RAFT는 비잔틴 환경(노드가 악의적으로 거짓 정보를 전파하는 상황)을 고려하지 않으므로, 시스템 내 신뢰할 수 없는 노드가 존재할 경우 합의 결과의 무결성을 담보하지 못한다는 한계가 있다. 즉, 악의적인 노드가 로그를 변조하거나 잘못된 블록을 전파하면 RAFT만으로는 이를 막을 수 없다. PBFT 알고리즘은 Castro와 Liskov의 고전 연구로 제안된 비잔틴 내성 합의(BFT) 프로토콜이다[4]. PBFT의 동작은 크게 3단계 메시지 교환으로 구성되는데, 한 노드가 프라이머리(Primary)가 되어 클라이언트 요청을 다른 노드들과 합의하는 Pre-prepare, Prepare, Commit 과정을 순차적으로 진행한다. 이를 통해 네트워크에 참가한 $3f+1$개의 노드 중 최대 f개가 악의적이거나 비정상이어도 올바른 합의 결과를 얻을 수 있도록 설계되었다. PBFT는 합의 단계마다 다수의 디지털 서명 검증을 수행하고 노드 간 모든-to-모든 브로드캐스트 통신을 요구하므로, 시스템 규모가 커질수록 통신 부하가 폭증하여 처리율이 급격히 떨어지고 지연 시간이 늘어난다는 단점을 가진다[1,4]. 실제로 노드가 10개를 넘어서 수십 개 수준에 달하면 PBFT 단독으로는 합의 성능이 크게 저하된다는 보고가 있으며, 이러한 이유로 PBFT는 주로 노드 규모가 제한된 컨소시엄 블록체인 등에서만

실용적인 것으로 평가된다. RAFT와 PBFT를 결합하여 두 알고리즘의 장점을 동시에 얻으려는 다양한 연구 시도가 존재해 왔다. 예를 들어, 계층적인 BFT(Hierarchical BFT) 구조를 통해 하위 그룹에서는 RAFT 합의를 수행하고 상위 그룹에서는 PBFT로 최종 합의를 하는 방식을 제안하였다[7,8]. 이를 통해 노드 수 증가 시 PBFT 단계에서 발생하는 병목을 완화하고자 하였다. 또한 대부분의 트랜잭션이 정상적으로 처리된다"는 가정 하에, RAFT로 평상시 합의를 진행하다가 주기적으로 PBFT 노드들이 RAFT 로그를 재검증하도록 하는 구조를 설계하였다. 일반적인 상황에서는 RAFT 속도로 처리하면서, 특정 시점에만 PBFT의 검증을 거치는 체크포인트 방식이라 볼 수 있다. 그러나 기존 연구들의 상당수는 알고리즘 개념 제안이나 제한적인 프로토타입 구현에 그치며, 노드 수, 비잔틴 노드 비율, 네트워크 지연 등 다양한 변수에 따른 체계적인 실험 분석이나 안전성 검증이 미흡하다는 지적이 있다.. 다시 말해, 정량적 평가와 실용적 검증이 부족하여 실제 시스템에 적용하기 위한 신뢰성을 담보하기엔 한계가 있었다. 본 논문에서는 이러한 연구들을 발전시켜, RAFT와 PBFT의 하이브리드 합의 구조를 명시적으로 구현하고 다양한 실험을 통해 그 효용성을 검증하고자 한다.

Ⅲ. 연구방법 및 결과

1. 전체 구조

본 연구에서 제안하는 시스템은 다중 RAFT 샤드(RAFT shards)와 글로벌 PBFT 그룹(PBFT committee)을 결합한 형태로 구성된다. 하위 계층에는 여러 개의 RAFT 클러스터(샤드)가 병렬로 존재하여 로그 복제 및 임시 합의(temporary log agreement)를 수행하고, 상위 계층에는 소규모의 PBFT 합의 그룹이 있어서 각 샤드에서 제출된 트랜잭션에 대해 최종 승인(final approval)을 내리는 역할을 담당한다. 각 RAFT 샤드는 독립적인 RAFT 합의 인스턴스로 동작하며 자체적인 리더 선출과 로그 저장소를 가진다. 시스템에 새로운 트랜잭션이 들어오면, 사전에 정의된 방식에 따라 하나 또는 모든 RAFT 샤드의 리더에게 전달되어 병렬로 처리된다. RAFT 단계에서 Crash Fault로 인한 일부 노드 장애가 발생하더라도 해당 샤드 내에서 리더 재선거를 통해 빠르게 복구할 수 있다. 한편 PBFT 그룹은 전체 시스템 관점에서 합의위원회 역할을 수행하는 노드들의 집합으로, 일반적으로 $Npbft = 3f+1$개의 노드로 구성되며 이 중 1개 노드가 Primary로 지정된다. PBFT 노드들은 RAFT 샤드와는 분리되어 병렬로 동작하며, PBFT 그룹 내에서는 노드 최대 f개까지 비잔틴 장애를 허용한다. PBFT 그룹의 크기는 합의 단계의 병목을 줄이기 위해 소규모(예: 4~7개 노드)로 유지한다. 이렇게 하면 PBFT의 높은 통신 복잡도를 완화하면서도 필요한 비잔틴 내성을 확보할 수 있다. RAFT 샤드들과 PBFT 그룹 간에는 인터페이스 역할을 하는 네트워크 계층이 존재하여, RAFT 단계가 완료된 트랜잭션의 메타정보(트랜잭션 ID, 포함된 샤드 ID, RAFT 로그 인덱스 등)를 PBFT 단계로 전달하고 그 결과를 다시 각 샤드에 알려주는 통신을 담당한다. 모든 통신 메시지는 지연, 손실, 중복 등의 네트워크 이벤트가 발생할 수 있다고 가정하며, PBFT 메시지는 디지털 서명으로 검증되고 RAFT 메시지는 RPC 방식으로 신뢰성을 확보한다.

하이브리드 합의 구조에서 트랜잭션 처리 흐름은 대략 다음과 같은 5단계 파이프라인으로 이뤄진다.

```
############################################################
# HybridConsensus
############################################################
class HybridConsensus(NetworkDelayMixin):
    def __init__(self, config=None, **kwargs):
        if config is None:
            config = ConsensusConfig()
        for key, value in kwargs.items():
            config.set(key, value)

        self.config = config
        self.stop_flag = False
        self.net_delay_range = config.get("net_delay_range", (0, 0))

        self.shard_count = config.get("shard_count", 3)
        self.shard_size = config.get("shard_size", 2)
        self.shards = []
        self.raft_nodes = {}
        for s in range(self.shard_count):
            shard_ids = []
            for j in range(self.shard_size):
                node_id = f"RaftShard{s}_N{j}"
                shard_ids.append(node_id)
            self.shards.append(shard_ids)
        for s in range(self.shard_count):
            for node_id in self.shards[s]:
                node = RaftNode(node_id, s, self.shards[s], config=config)
                self.raft_nodes[node_id] = node
```

그림 1. 제안하는 합의 구조의 구현체

① 트랜잭션 제출: 클라이언트나 시스템 내 상위 모듈이 새로운 트랜잭션을 발생시키면, 사전에 정한 샤드의 RAFT 리더 노드에 해당 트랜잭션을 전달한다. (모든 샤드에 동일 트랜잭션을 제출하는 방식이나, 트랜잭션 유형에 따라 특정 샤드에만 제출하는 샤딩 전략을 선택할 수 있다.)

② RAFT 로그 복제: 각 RAFT 샤드의 리더는 트랜잭션을 자신의 로그에 추가하고, 팔로워 노드들에게 AppendEntries RPC 메시지를 전송한다. 과반수 이상의 팔로워 노드로부터 응답이 오면 해당 트랜잭션을 샤드 차원에서 커밋 가능(commit-able) 상태로 인정한다. 만약 어떤 샤드에서라도 합의 실패(예: 리더 노드 다운이나 네트워크 단절로 과반 응답 실패)가 발생하면, 그 트랜잭션은 전체 시스템에서 중단 또는 재시도되고 더 이상 진행되지 않는다.

③ PBFT 요청 생성: 모든 RAFT 샤드에서 해당 트랜잭션의 로그 복제가 성공하면, 시스템은 PBFT 합의를 시작한다. 한 가지 구현 방식으로, 미리 지정된 대표 노드(예: 샤드 1의 리더)가 PBFT 그룹의 Primary 노드에게 “트랜잭션이 모든 샤드에 복제 완료됨”을 알리는 요청(request) 메시지를 보낸다. 이 요청에는 트랜잭션 ID, 관련된 샤드 정보, RAFT 단계의 로그 인덱스/임기(term) 등이 포함되어 PBFT 단계에서 해당 트랜잭션을 고유하게 식별하고 검증하는 데 사용된다.

④ PBFT 합의 수행: PBFT Primary 노드는 요청을 받으면 해당 트랜잭션에 대한 PBFT 합의 절차(Pre-Prepare → Prepare → Commit)를 개시한다. 먼저 다른 PBFT 노드들에게 Pre-prepare 메시지를 전송하고, 각 노드는 이를 검증하여 정당하다고 판단하면 Prepare 메시지를 브로드캐스트한 후, 충분한 동의가 모이면 Commit 메시지를 교환하여 최종 합의를 시도한다. PBFT Primary가 응답하지 않거나 악의적이라면 뷰 변경(view change)이 발생하여 새로운 Primary를 선출하고 합의를 재개한다. 장애가 없을 경우 PBFT 그룹 내 최소 $2f+1$개의 노드가 Commit 단계에 참여하여 해당 트랜잭션을 승인한다.

⑤ 최종 확정 및 반영: PBFT 단계에서 Commit 메시지 교환에 성공하면, 이는 곧 해당 트랜잭션이 비잔틴 검증을 통과했음을 의미하므로 시스템 전역에서 최종 커밋된 것으로 간주한다. RAFT 단계에서 각 샤드가 임시 저장해 두었던 트랜잭션 로그는 확정 상태로 승격되어 불가역적인 합의 결과가 된다. 반대로 PBFT 단계에서 해당 트랜잭션이 충분한 서명을 얻지 못해 합의 실패로 판정되면, RAFT 샤드에 저장되었던 로그도 최종적으로 폐기하거나 무효화된다. 본 구현에서는 PBFT 승인을 얻지 못한 트랜잭션은 모든 샤드에서 롤백(rollback) 처리하도록 하여, PBFT에서 승인된 트랜잭션만이 최종 상태로 유지되도록 보장하였다.

이상의 2단계 합의 파이프라인을 통해, RAFT가 이미 수행한 로그 복제를 바탕으로 PBFT는 추가 검증만 진행하면 되므로 PBFT 단독 대비 통신 및 연산 부하를 상당 부분 상쇄할 수 있다. 또한 RAFT와 PBFT 단계가 분리되어 병렬 동작하기 때문에 한쪽 단계에 장애가 발생해도 다른 쪽 단계가 즉각 영향을 받지 않는다. 예를 들어 RAFT 샤드의 리더 노드가 장애로 다운되면 해당 샤드 내에서만 리더 재선출이 일어나고 로그 합의를 재시도하면 되며, PBFT Primary 노드가 비잔틴 행위를 하면 PBFT 그룹 내부에서 뷰 변경을 통해 새로운 Primary를 선출하여 합의를 계속 진행하면 된다. 이처럼 RAFT와 PBFT의 장애 처리 메커니즘이 독립적으로 동작함으로써, 둘 중 하나의 장애가 전체 시스템을 마비시키지 않고도 부분적으로 복구되어 시스템의 활성(liveness)을 유지할 수 있다. 한편 비잔틴 보안 측면에서는, PBFT 검증을 통해 최종 무결성을 확보하는 전략을 취한다. 설령 RAFT 단계에서 한 노드가 악의적으로 잘못된 트랜잭션을 로그에 넣어두더라도 PBFT 단계에서 다수의 올바른 노드로부터 서명된 승인을 얻지 못하면 해당 트랜잭션은 최종 커밋에 이르지 못한다. 결과적으로 RAFT의 높은 처리율에 PBFT의 무결성 검증을 결합함으로써, 평상시에는 단일 RAFT와 거의 동일한 속도로 합의를 진행하면서도 비잔틴 장애가 발생할 경우 PBFT 수준의 안전성을 확보하는 효과를 얻을 수 있다. 더 나아가 제안 구조에서는 다수의 샤드를 병렬 운영함으로써 RAFT 단계의 처리율을 선형적

으로 확장할 수 있고, PBFT 그룹은 소수 정예로 구성하여 합의 지연을 최소화할 수 있으므로 확장성과 안전성을 모두 충족할 수 있다[6]. 이러한 특징들로 인해 본 하이브리드 합의 구조는 노드 규모가 매우 크지 않은 허가형 블록체인이나 분산 데이터 저장소 환경에서 실용적인 대안이 될 수 있다. (노드 수가 수백 개 이상으로 매우 커지는 환경에서는 PBFT 단계가 여전히 병목이 될 수 있으므로, 제안 방식은 주로 수십 개 내외의 노드로 구성된 시스템에서 적합하다.)

2. 형식적 검증

RAFT와 PBFT를 연계한 하이브리드 합의 구조에 대해 형식적 검증(formal verification)을 수행하여, 시스템이 핵심 속성인 안전성(Safety)과 활성(Liveness)을 충족하는지 확인하였다. RAFT와 PBFT 개별 알고리즘은 각각 안전성과 활성에 대해 이론적으로 보장되지만, 두 단계를 연쇄 결합했을 때에도 동일한 보장이 유지되는지는 명확히 증명되지 않았다. 따라서 본 절에서는 상태 기반 모델을 통해 하이브리드 합의 구조의 동작을 논리적으로 기술하고, 안전성과 활성 속성이 항상 성립하는지 검증한다. 여기서 안전성이란 동일한 시퀀스 번호에 두 개 이상의 다른 트랜잭션이 커밋되지 않으며, 승인된 트랜잭션이 중복으로 최종 커밋되지 않는 특성을 의미한다. 활성이란 충분한 수의 정상 노드가 있을 경우 모든 유효 트랜잭션이 결국 유한 시간 내에 최종 커밋에 도달한다는 특성을 뜻한다.

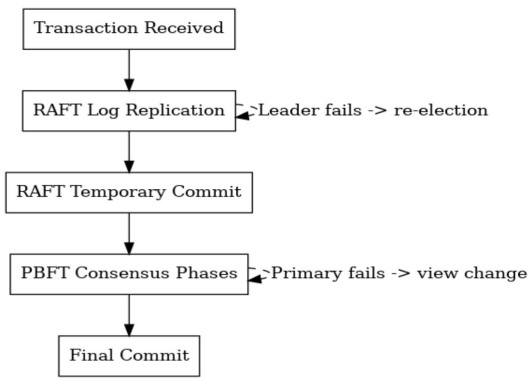

그림 2. 하이브리드 합의 구조의 순서도

형식적 검증은 상태 전이 모델에 기반한 유계 모델 검증(bounded model checking) 기법으로 수행하였다. 하이브리드 합의 알고리즘의 RAFT 단계와 PBFT 단계를 아우르는 상태 머신을 정의하고, Python 기반 SMT 솔버인 Z3와 PySMT를 활용하여 논리 검증을 진행하였다[9,10].

가. SMT 불변식 및 검증 조건

형식적 모델에서는 안전성(Safety)과 활성 보장(Liveness)을 검증하기 위해 아래와 같은 논리 불변식을 명세하였다. SMT 기반의 검증 도구(Z3/PySMT)를 통해, 모든 가능 상태 전이 경로에서 이 불변식들이 유지되는지를 확인한다.

```
from z3 import *

# --- Example RAFT Shard Model (3-node shard) ---
num_nodes = 3
is_leader = [Bool(f'is_leader_{i}') for i in range(num_nodes)]

# RAFT invariant: There can be at most one leader in a shard
raft_invariant = Sum([If(is_leader[i], 1, 0) for i in range(num_nodes)]) <= 1

# --- Example PBFT Model (4 nodes, f=1) ---
pbft_node_count = 4
# For each PBFT node, a variable indicating whether a commit signature is present
pbft_commit = [Bool(f'commit_{i}') for i in range(pbft_node_count)]
# In a normal environment (without considering Byzantine nodes),
# it is assumed that an honest node does not perform double-signing.
# Normal condition: When only honest nodes sign, at least 3 signatures are required for consensus (2f+1 = 3)
pbft_commit_count = Sum([If(pbft_commit[i], 1, 0) for i in range(pbft_node_count)])
pbft_invariant = pbft_commit_count >= 3

# --- Hybrid Interface: Transactions committed in RAFT are passed to PBFT ---
# Simply link the RAFT commit variable 'tx_commit' with the final approval condition in PBFT.
tx_commit = Bool('tx_commit')  # Indicates whether the transaction (tx) is committed in RAFT
pbft_approved = pbft_invariant  # PBFT final approval condition for the transaction (tx)

# Hybrid invariant: If a transaction (tx) is committed in RAFT, it must also be approved in PBFT.
hybrid_invariant = Implies(tx_commit, pbft_approved)

# Adding constraints to the SMT solver
s = Solver()
s.add(raft_invariant, pbft_invariant, hybrid_invariant)

# For example, set the state to indicate that the transaction (tx) is committed in RAFT
s.add(tx_commit == True)

# Perform verification
if s.check() == sat:
    print("All invariants hold. (Safety and progress confirmed)")
    print(s.model())
else:
    print("Invariant violation detected! (Counterexample model exists)")
```

그림 3. Z3 솔버를 이용한 구현체

① Safety 1: 동일한 로그 인덱스에 서로 다른 트랜잭션이 커밋되지 않는다. 즉, 한 번 특정 log_index에서 어떤 tx_id가 committed로 확정되면, 다른 트랜잭션이 동일한 log_index에 커밋 상태로 기록되는 경우는 발생할 수 없다. 이 불변식은 RAFT와 PBFT 결합 프로토콜의 기본 일관성 조건으로, 분산 로그의 단조(total order) 특성을 의미한다. 이속성 덕분에 두 노드가 같은 위치에 서로 다른 명령을 확정짓는 충돌 커밋 상황이 방지된다.

② Safety 2: 하나의 트랜잭션 ID는 시스템에서 최대 한 번만 committed 될 수 있다. 중복 트랜잭션 처리가 없음을 나타내는 속성으로서, 이미 커

밋된 tx_id를 다시 커밋하려 시도하지 않는 것을 보장한다. 이를 통해 트랜잭션의 Exactly-Once (정확히 한 번 처리)한다라고 할 수 있다. 만약 동일 트랜잭션이 두 번 커밋되는 시나리오가 있다면 불변식 위반으로 검출된다.

③ Liveness: Crash 또는 비잔틴 노드가 허용된 최대치인 f 이하로 존재하는 한, 모든 트랜잭션은 결국 유한 단계 내에 커밋에 도달한다. 이 활성(liveness) 조건은 프로토콜이 일정 수준의 장애를 견디면서도 진행성(progress)을 잃지 않음을 의미한다. 예를 들어 일부 노드가 장애가 생겨도 남은 정상 노드들이 합의를 이어나가 최종적으로 각 트랜잭션을 커밋함을 보장해야 한다. 본 모델에서는 타임아웃 후 재시도, 뷰 변경 등의 메커니즘을 포함하여 $2f$+1 이상의 노드가 합의에 참여할 수 있는 한, 트랜잭션이 무한정 지연되거나 누락되지 않고 처리됨을 불변식으로 규정하였다. 단, 네트워크 파티션과 같이 일시적으로 활성 조건을 저해하는 상황은 영구적이지 않다고 가정하며 (일정 시간 후 복구), 영구적인 f 초과 장애는 고려 대상에서 제외한다.

나. 상태 요소 정의

하이브리드 합의 프로토콜의 시스템 상태는 다음과 같은 요소들로 구성된다.

① 리더 ID (raft_leader): 현재 RAFT 리더 노드의 식별자. RAFT 리더 선출에 따라 변경되며, 로그 복제를 조율한다.

② 현재 임기 (term): RAFT에서 사용되는 현재 임기 번호로, 리더 선출 시 증가한다. 새로운 리더가 선출되면 가장 높은 임기로 갱신된다.

③ 로그 인덱스 (log_index): 트랜잭션이 기록되는 RAFT 로그 위치를 나타낸다. 각 샤드별로 증가하며, PBFT 최종 승인이 되면 해당 인덱스의 로그 항목이 확정된다.

④ 로그 값 (log_value): 특정 로그 인덱스에 저장된 트랜잭션 내용 또는 명령값이다. 로그 값은 RAFT AppendEntries RPC로 전파되며, 일단 기록되면 리더 교체 후에도 삭제되지 않는다.

⑤ 로그 커밋 플래그 (commit_flag): 로그 항목의 커밋 여부를 표시한다. RAFT 단계에서 과반수 복제를 이루면 임시 커밋 가능 상태로 설정되고, PBFT 단계에서 최종 Commit에 성공하면 최종 커밋으로 확정된다.

⑥ PBFT 단계 상태 (pbft_stage): PBFT 합의 프로토콜의 현재 단계를 나타내는 값으로, {None, PrePrepare, Prepare, Commit} 중 하나를 갖는다. 예를 들어 PBFT Primary가 새로운 요청을 브로드캐스트하면 pbft_stage가 PrePrepare로 전이되고, 이후 노드들이 Prepare 메시지를 교환하면 Prepare 단계로, 최종 Commit 메세지 교환 시 Commit 단계로 변경된다. 각 PBFT 노드는 이 값을 통해 현재 자신이 어느 단계에 있는지 추적한다.

⑦ 트랜잭션 ID (tx_id) 및 트랜잭션 상태 (tx_status): 처리 중인 트랜잭션의 식별자와 현재 상태를 나타낸다. 트랜잭션 상태는 pending (합의 진행 중), committed (최종 합의 완료), discarded (합의 실패로 폐기) 등의 값을 가진다. 새로운 트랜잭션이 시스템에 접수되면 pending으로 등록되고, RAFT와 PBFT 단계를 거쳐 최종 합의되면 committed로 바뀐다. 만약 PBFT 단계에서 부결되면 해당 트랜잭션은 discarded 처리된다.

⑧ 노드 상태 (node_status): 각 노드의 동작 상태를 나타내며, {정상, 정지(crash), 비잔틴} 중 하나이다. 정상 노드는 프로토콜을 준수하여 동작하고, 정지(crash) 상태가 되면 더 이상 메시지를 보내거나 처리하지 못한다. 비잔틴 상태인 노드는 임의의 혹은 악의적인 동작을 보이는 노드로서, 잘못된 메시지 전송, 서명 위조, 메시지 드롭 등 프로토콜 위반 행동을 할 수 있다. 시스템은 최대 f개의 노드까지 비잔틴 상태를 허용하도록 구성된다(PBFT 그룹에서 $Npbft = 3f+1$ 구성을 가정한다).

다. 전이 조건

정의된 상태 요소들은 RAFT와 PBFT 프로토콜 동작, 그리고 장애(event)에 따라 다음과 같이 변화한다.

① RAFT 단계 전이: RAFT 알고리즘에서 리더 선출과 로그 복제 관련 사건들로 상태가 변한다. 예를 들어 한 노드가 타임아웃되어 후보자(candidate)가 되면 다른 노드들에게 RequestVote RPC를 전송하고 임기를 증가시킨다. 이 후보자가 과반수로부터 투표를 획득하면 해당 노드는 리더로 승격되며 raft_leader가 그 노드 ID로 변경되고, term이 새로운 임기로 설정된다. 반대로 기존 리더가 새로운 리더의 AppendEntries RPC를 수신하면 자신의 역할을 팔로워로 강등하고 raft_leader를 갱신한다. 리더는 새로운 트랜잭션을 수신할 때마다 log_index를 증가시키고 해당 위치에 log_value를 기록한 뒤, 팔로워들에게 AppendEntries를 보내 로그를 복제한다. 팔로워 노드가 AppendEntries를 받으면 자신의 로그에 해당 항목을 추가하고 log_index 및 log_value 상태를 갱신한다. 과반수의 팔로워에게서 응답이 오면 해당 로그 항목의 commit_flag를 임시 커밋 가능으로 설정한다.

② PBFT 단계 전이: RAFT 단계에서 모든 샤드의 로그 복제가 완료되면, PBFT Primary가 해당 트랜잭션 ID와 로그 정보를 가지고 PBFT 합의를 개시한다. Primary 노드는 PrePrepare 메시지를 생성하여 PBFT 백업 노드들에 전송하고, 이를 받은 백업 노드들은 메시지의 유효성을 검사한 뒤 자신의 pbft_stage를 PrePrepare에서 Prepare 단계로 전이시키고 Prepare 메시지들을 브로드캐스트한다. 각 PBFT 노드가 $2f+1$개의 유효한 Prepare 메시지를 수신하면 해당 트랜잭션에 대한 준비 완료(prepared) 상태를 만족하게 되고, 자신의 pbft_stage를 Commit 단계로 바꾸어 Commit 메시지를 전송한다. 이어서 $2f+1$개의 Commit 메시지가 모이면 노드는 해당 트랜잭션을 Commit 확정할 수 있게 되며, 이에 따라 해당 트랜잭션 ID의 tx_status를 committed로 설정하고 RAFT의 commit_flag를 최종 커밋으로 표시한다. 이 시점에서 pbft_stage는 다시 None 상태로 리셋되어 새로운 트랜잭션 합의를 받을 준비를 한다. 만약 Primary 노드가 비잔틴으로 PrePrepare 메시지를 잘못 보낼 경우 백업 노드들이 이를 감지하고 일정 시간 응답이 없으면 뷰 변경(view change) 절차를 개시한다. 뷰 변경이 일어나면 PBFT의 현재 뷰 숫자가 증가하고 새로운 Primary로 raft_leader와 별개로 primary 노드 ID가 교체되며, pbft_stage가 기존 진행 중이던 합의를 초기화하고 새 Primary 기준으로 PrePrepare 단계부터 다시 진행된다.

③ 장애 및 특수 이벤트: 시스템 동작 중 장애(fault)가 발생하면 관련 상태가 즉시 변화하거나 추가 전이가 필요하다. 노드 Crash 발생 시 해당 노드의 node_status를 정지(crash)로 표시하고, RAFT 샤드의 리더가 죽은 경우 해당 샤드에서는 새로운 임기의 선거가 촉발되어 (term 증가) 다른 노드를 리더로 선출하게 된다. 이 과정에서 일시적으로 로그 복제가 지연되나, 새로운 리더가 이전 리더의 로그를 인계받아 지속성을 유지한다. 비잔틴 장애의 경우 PBFT 단계에서만 고려되는데, 예를 들어 PBFT Primary가 악의적으로 서로 다른 노드에 동일 시퀀스 번호로 서로 다른 PrePrepare를 보내는 경우, 백업 노드들이 일치된 시퀀스 번호에 합의할 수 없으므로 해당 뷰는 실패하고 뷰 변경이 일어난다. 이때 기존 뷰에서의 미확정 트랜잭션은 새로운 뷰로 넘어가 재시도를 기다리거나 폐기된다. 또한 메시지 드롭(drop)이 네트워크층에서 발생할 수도 있는데, 일부 AppendEntries나 Prepare/Commit 메시지가 유실되면 해당 메시지를 받지 못한 노드는 타임아웃이나 추가 재전송을 통해 프로토콜을 계속 진행한다. 예를 들어 PBFT에서 한 노드가 Commit 메시지를 못 받았더라도, 다른 노드들이

$2f$+1 Commit을 확보하면 합의는 완료되며, 유실된 노드는 나중에 다른 노드의 상태를 통해 해당 트랜잭션이 커밋된 것을 알게 된다. 이러한 네트워크 불확실성은 상태 전이 모델에서 non deterministic drop이나 지연으로 추상화되며, 그로 인해 필요한 경우 추가 전이(예: 재전송 전이)가 발생할 수 있다.

라. 상태-전이 흐름의 세부

정의한 상태와 전이 규칙을 토대로, RAFT+PBFT 하이브리드 합의의 상태 변화 흐름을 도식화하면 [그림4]와 같다. 초기 상태에서는 각 샤드마다 RAFT 리더가 선출되어 있고(PBFT Primary도 별도로 존재), 모든 노드가 정상 상태이며 pbft_stage=None 이다.

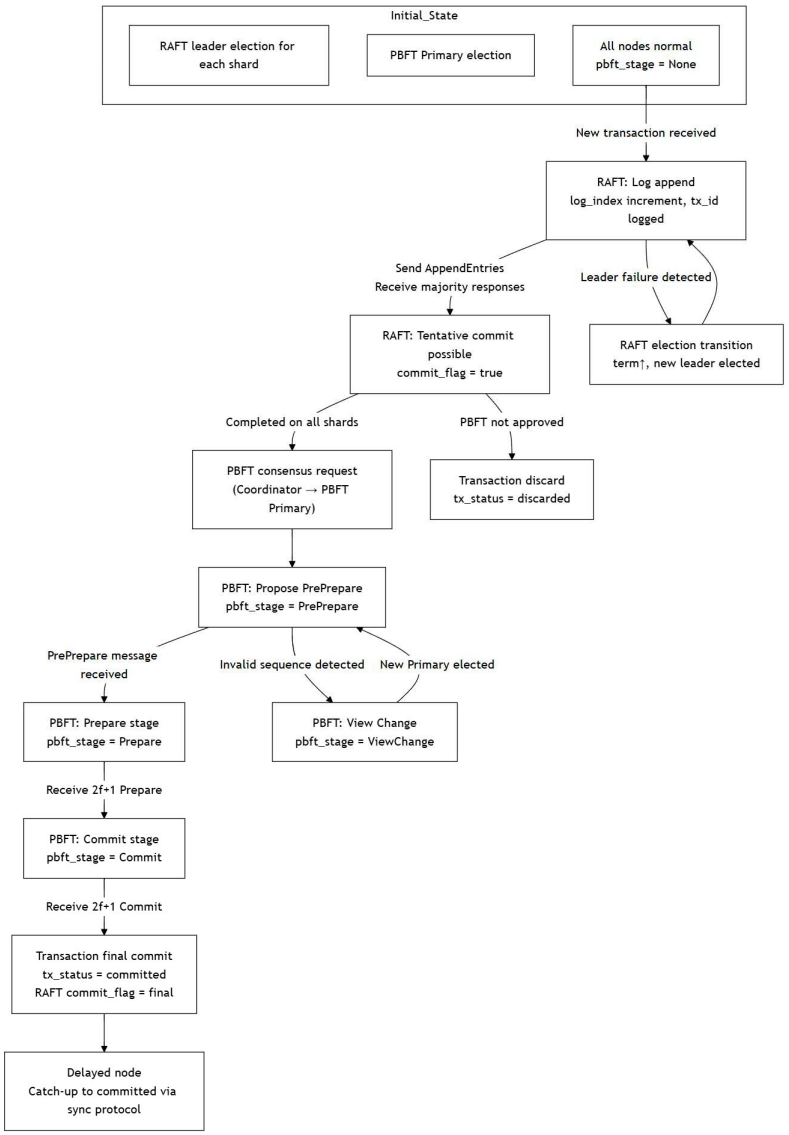

그림 4. 상태-전이 흐름도

새 트랜잭션이 들어오면 우선 RAFT 리더들의 log_index가 증가하고 해당 tx_id를 포함한 로그 값이 추가된다. AppendEntries 전송과 과반수 응답을 통해 각 샤드에서는 해당 tx_id가 임시 커밋 가능 상태(commit_flag=true 임시 설정)로 전환된다. 이 과정에서 term(임기)은 기존 리더 유지 시 변화가 없

지만, 만약 리더 장애가 발생했다면 새로운 리더 선출로 raft_leader와 term이 바뀐 후 로그가 복제된다.

모든 샤드에서 로그 복제가 완료되면, 시스템은 PBFT 단계로 전환한다. 하나의 샤드 또는 별도의 코디네이터가 PBFT Primary에게 트랜잭션 합의 요청을 보내면, PBFT Primary 노드는 해당 tx_id를 새 합의 시퀀스로 제안하며 자신의 pbft_stage를 PrePrepare로 설정한다. 곧바로 PrePrepare 메시지가 PBFT 전체에 퍼지고, 각 백업 노드들은 이를 수신한 순간 자신의 상태를 PrePrepare에서 Prepare로 변경하고 Prepare 메시지를 보낸다. 상태 변화: Prepare 단계에서 각 PBFT 노드는 pbft_stage=Prepare이며, 내부적으로 해당 트랜잭션의 tx_status는 아직 pending이지만 prepared 조건을 만족했는지 추적한다. 충분한 Prepare 메시지를 받아 prepared 상태가 충족되면 (2f+1 Prepare 수신) pbft_stage를 Commit으로 변경하고 Commit 메시지를 전송한다. Commit 메시지 또한 $2f$+1개 모이면 각 노드는 tx_status를 committed로 바꾸고, RAFT의 해당 로그 항목 commit_flag를 최종 커밋으로 설정한다. 이때 상태 변화의 핵심은 모든 정직한 노드들의 로그와 트랜잭션 상태가 일관되게 committed로 맞춰진다는 점이다. 한편, 한 노드가 Commit 단계에 들어갔으나 다른 일부 노드의 Commit 메시지가 지연되더라도, 합의 정족수를 채운 노드들이 최종 커밋을 진행하기 때문에 시스템 전체적으로 해당 트랜잭션은 커밋된 것으로 간주된다. 지연된 노드도 이후 동기화 프로토콜이나 다른 노드의 정보를 통해 자신의 상태를 committed로 따라잡게 된다.

만약 이 과정 중 복합 장애 상황이 발생한다면 각 단계별로 대응하는 추가 상태 전이가 일어난다. 예를 들어 RAFT 로그 복제 중 한 샤드 리더가 Crash되면, 그 샤드의 node_status가 Crash로 설정됨과 동시에 해당 샤드에서 새로운 리더를 뽑는 선거 전이가 발생한다 (term 증가, 새로운 raft_leader 설정). 그런 다음 남은 단계들을 진행하며 PBFT에 진입한다. 또 PBFT 단계에서 Primary가 비잔틴 동작을 하면 (예: 잘못된 시퀀스 할당), 일정 시간 후 백업 노드들이 뷰 변경을 개시하여 PBFT 합의를 재시작한다. 뷰 변경 동안 pbft_stage는 None 또는 특별한 뷰전이 상태로 두고, 새로운 Primary 선출이 완료되면 다시 PrePrepare 단계부터 진행한다. 이러한 장애 시나리오에서도 상태 불일치가 없도록, RAFT 단계에서 임시 커밋되었던 트랜잭션은 PBFT 최종 승인이 나지 않으면 최종 커밋으로 전이되지 않으며, 필요시 tx_status를 discarded로 변경하여 로그에서 무효화 한다. 그 결과 RAFT와 PBFT 어느 한 쪽 단계에서만 커밋되고 다른 쪽에서는 취소되는 모순 상태를 방지한다. 요컨대, 정의된 상태-전이 모델은 정상 시나리오부터 Crash/비잔틴 혼합 상황까지 모든 경우의 상태 변화를 추적하며, 각 전이 마다 관련된 상태 변수들(leader, term, pbft_stage, commit_flag 등)이 어떻게 변하는지를 명시적으로 기술한다.

마. 형식적 검증의 결과

정의된 상태 전이 모델과 불변식을 바탕으로 SMT 기반의 검증을 수행한 결과, 제한된 상태 공간 내에서 모든 안전성 조건이 충족됨을 확인하였다. 구체적으로, 노드 수 5~9개 구성에서 최대 6단계의 상태 전이를 포괄하는 bounded model checking을 수행한 바, 앞서 명시한 Safety 불변식들을 위배하는 반례(counterexample)가 존재하지 않음을 확인하였다. 예를 들어, 두 개의 다른 트랜잭션이 하나의 로그 인덱스에 커밋되는 시나리오나, 동일 트랜잭션이 중복 커밋되는 시나리오를 탐색했지만, 지정된 제한 내에서는 그러한 경우가 발생하지 않았다. 또한 Crash 노드와 Byzantine 노드가 동시에 존재하는 복합 장애 상황에서도 합의 프로토콜의 상태 무결성이 유지됨을 보였다. 이는 RAFT+PBFT 하이브리드 합의 구조가 설계된 대로 안전성(safety)을 확보하고 있음을 시사하며, 아울러 f 이하여서 발생하는 장애에 대해서는 활성(liveness) 속성도 지켜짐을 의미한다. 요약하면, 형식적 검증을 통해 제안한

하이브리드 합의 모델이 작은 규모의 검증 가능한 범위 내에서 모순 없는 상태 전이를 보임을 확인하였고, 이는 본 프로토콜의 타당성을 뒷받침하는 근거가 된다.

Ⅳ. 결 론

본 연구는 분산 시스템에서 높은 처리 성능과 비잔틴 장애 내성을 동시에 달성하기 위한 새로운 합의 구조를 모색하였다. 기존 RAFT와 PBFT 프로토콜의 트레이드 오프에 주목하여, RAFT의 신속한 합의 형성 능력과 PBFT의 강력한 비잔틴 장애 대응 능력을 결합함으로써 하나의 시스템에서 성능과 보안성을 모두 확보하고자 하였다. 이러한 문제의식을 바탕으로 RAFT와 PBFT를 연계하는 하이브리드 합의 구조를 제안하였으며, 이를 통해 Crash Fault 환경에서도 뛰어난 성능을 유지하면서 Byzantine Fault 환경에서도 안전성을 보장할 수 있는 합의 기제의 가능성을 탐색하였다. 제안한 하이브리드 합의 구조는 RAFT 알고리즘의 초기 합의 단계와 PBFT 알고리즘의 최종 승인 단계를 결합하여 설계되었다. 구체적으로, 다수결 투표 기반의 RAFT를 활용해 각 샤드(shard) 내에서 로그 복제와 임시 합의를 빠르게 수행한 뒤, 소수의 대표 노드로 구성된 PBFT 전역 그룹에서 서명 검증을 거쳐 최종 커밋을 확정한다. 이러한 2단계 절차를 통해 정상적인 상황에서는 RAFT에 준하는 속도로 합의가 이루어지고, 비정상 상황에서는 PBFT의 추가 검증을 통해 시스템 전반의 무결성이 확보된다. 다시 말해, RAFT 단계의 병렬 처리를 통한 처리율 향상과 PBFT 단계의 최종 무결성 보장을 조화롭게 결합함으로써, 성능 저하 없이도 비잔틴 노드에 대비할 수 있는 합의 과정을 구현하였다. 본 하이브리드 합의 메커니즘은 기존의 RAFT-PBFT 결합 연구들과 비교하여 몇 가지 중요한 차별점과 이론적 기여를 가진다. 선행 연구들이 RAFT와 PBFT를 제한적으로 조합하거나 개념 수준에서 제안된 경우가 많았던 반면, 본 연구는 합의 전 과정에 걸쳐 두 알고리즘을 통합함으로써 확장성과 안정성을 동시에 추구하였다. 예를 들어, 일부 기존 접근법이 PBFT의 리더 선출만 RAFT로 대체하거나 샤딩 환경에 국한된 반면, 본 설계는 RAFT 단계 자체를 다중 샤드로 병렬화하고 PBFT를 글로벌 검증자로 활용하는 통합적 구조를 제시하였다. 이를 통해 Crash Fault 기반 시스템과 Byzantine Fault 기반 시스템의 장점을 하나로 접목하는 새로운 합의 패러다임을 구축하였으며, 성능-보안성 균형이라는 분산 시스템 분야의 난제에 대한 이론적 해법을 제시한 점에서 의의가 있다고 할 수 있다. 또한 본 연구에서는 제안된 하이브리드 합의 프로토콜의 유효성을 뒷받침하기 위해 형식 검증 측면에서 다양한 기술적 성과를 도출하였다. 아울러 Z3/PySMT 등을 활용한 형식적 검증을 통해 RAFT의 임시 합의와 PBFT 최종 승인 사이에 발생할 수 있는 복잡한 시나리오(예: 리더 장애와 PBFT 뷰 변경의 동시 발생)에 대해서도 시스템의 안전성 속성이 유지됨을 확인하였다. 이러한 구현 실험과 논리 검증 작업은 하이브리드 합의 구조가 이론상 뿐만 아니라 실제 환경에서도 효과적으로 동작할 수 있음을 보여주며, 제안한 합의 프로토콜의 신뢰성과 견고성을 한층 강화한다. 마지막으로, 본 연구의 결과는 향후 분산 시스템 설계에 실용적인 시사점을 제공한다. 제안된 RAFT-PBFT 하이브리드 합의 구조는 노드 수가 비교적 제한적이지만 높은 보안 요구사항이 있는 허가형 블록체인, 금융 거래 플랫폼, 분산 데이터베이스 등에 적용할 수 있을 것으로 기대된다. 본 구조를 도입함으로써 기존 시스템에서도 RAFT와 PBFT의 장점을 동시에 활용하여 높은 보안성과 성능을 갖춘 서비스를 구현할 수 있으며, 비잔틴 장애에 대비한 안정적인 운영이 가능해진다. 향후 본 연구를 바탕으로 대규모 네트워크 환경에 대한 최적화, 다양한 비잔틴 공격 시나리오에 대한 추가적인 검증, 그리고 프로토콜의 표준화 및 실 환경 적용과 같은 과제가 이어진다면, 제안된 하이브리드 합의 메커니즘은 차세대 분산 시스템을 위한 실용적이고 신뢰할 수 있는 합의 솔루션으로 자리매김할 것으로 기대된다.

REFERENCES

[1] 민연아, "프라이빗 블록체인 기반의 사용자 환경을 고려한 수정된 PBFT 연구," *스마트미디어저널*, 제9권, 제1호, 9-15쪽, 2020년

[2] Leslie Lamport, "The part-time parliament", *ACM Transactions on Computer Systems,* vol. 16, issue 2, pp. 133 - 169, 1998.

[3] Ongaro, Diego, and John Ousterhout, "In search of an understandable consensus algorithm", *2014 USENIX annual technical conference.* pp. 305-319, 2014.

[4] Castro, Miguel, and Barbara Liskov, "Practical byzantine fault tolerance" *OsDI(Operating systems design and implementation),* Vol. 99. pp. 173-186. 1999.

[5] Zhong, Weiyu, Ce Yang, Wei Liang, Jiahong Cai, Lin Chen, Jing Liao, and Naixue Xiong, "Byzantine Fault-Tolerant Consensus Algorithms: A Survey" *Electronics(MDPI)* Vol. 12, no. 18: 3801.

[6] Wu, X., Jiang, W., Song, M. et al, "An efficient sharding consensus algorithm for consortium chains" *Scientific Reports(Nature)* vol. 13, Article Number: 20, 2023.

[7] Rongxin Guo, Zhenping Guo, Zerui Lin and Wenxian Jiang, "A hierarchical byzantine fault tolerance consensus protocol for the internet of things", *High-Confidence Computing,* Vol 4, Issue 3: 100196, 2024.

[8] Team, ByStack, "*BBFT: a Hierarchical Byzantine Fault Tolerant Consensus Algorithm.*" 2019

[9] G. Doussot, "Software Verification and Analysis Using Z3," *NCC Group Research Blog*(2021), https://www.fox-it.com/nl-en/research-blog/software-verification-and-analysis-using-z3/ (accessed Mar., 14, 2025).

[10] Gario, Marco, and Andrea Micheli, "PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms." *SMT workshop,* Vol. 2015, 2015.

저 자 소 개

이지운(정회원)

2016년서강대학교 정보통신대학원 석사 졸업

2021년 한국기술교육대학교 컴퓨터공학과 박사 졸업

2021년 한양대학교 산학협력단(서울) 연구원

2022년 ~ 현재: 한국기술교육대학교 미래융합학부 기술연구원

<주관심분야 : 블록체인, 클라우드 컴퓨팅, 정보보안>

서희석(정회원)

2000년 성균관대학교 산업공학과 학사 졸업

2002년 성균관대학교 전기전자및컴퓨터공학부 석사 졸업

2005년 성균관대학교 전기전자및컴퓨터공학부 박사 졸업

2005년 ~ 현재: 한국기술교육대학교 컴퓨터공학부 교수

<주관심분야 : 시스템보안, 클라우드 보안>