Mutual Cloud - 단일 장애점 없는 P2P 분산 자원 관리를 위한 탈중앙 제어 평면 프레임워크

Mutual Cloud - A P2P Decentralized Control Plane Framework for SPOF-Resilient Distributed Resource Management

구르미(09팀) : 금채원(2276029), 이서영(2276218), 송예린(2171023)

1. Team Info

1.1. 과제명

Mutual Cloud: 단일 장애점 없는 P2P 분산 자원 관리를 위한 탈중앙 제어 평면 프레임워크 (Mutual Cloud: A P2P Decentralized Control Plane Framework for SPOF-Resilient Distributed Resource Management)

1.2. 팀 정보

팀 번호	09
팀 이름	구르미

1.3. 팀 구성원

이름	학번	이메일
금채원(팀장)	2276029	sskeum2009@naver.com
이서영(팀원)	2276218	okcathy@ewhain.net
송예린(팀원)	2171023	sylin0820@ewha.ac.kr

2. Project-Summary

2.1. 문제 정의

~~산업 전반이 'Cloud First'를 거치며 퍼블릭 클라우드 의존도가 높아졌지만, 비용이 지속적으로 누적·상승하면서 온프레미스(on-premises) 회귀가 늘고 있다. 본 절은 여기에서 파생되는 두 가지 문제를 정의한다. 첫째, 온프레미스는 피크 트래픽을 기준으로 용량을 미리 확보하는 과다 프로비저닝(over-provisioning)을 유발해 대부분의 시간 동안 컴퓨팅 자원이 유휴 상태로 남는다. 퍼블릭 클라우드는 컴퓨팅 사용료, 데이터 전송, 스토리지 비용 등 운영비(OPEX)가 누적되어 장기 총소유비용(TCO)을 끌어올린다. 결과적으로 동일 워크로드 대비 과지출과 자산 회전율 저하가 발생하고, 확장 탄력성도 제약된다.~~

최근 클라우드 인프라는 대규모 데이터 처리와 AI 서비스의 핵심 기반으로 자리 잡았으나, 대부분의 시스템은 여전히 중앙집중형 제어 평면(Centralized Control Plane) 에 의존하고 있다. 이러한 구조는 제어 기능이 하나의 중앙 노드에 집중되어 있어, 단일 장애점(SPOF,

Single Point of Failure)과 제어 부하 집중(Bottleneck) 문제를 본질적으로 내포한다. 스케줄링, 메타데이터 관리, 상태 동기화 등 핵심 제어 로직이 중앙 서버에 집중될 경우, 제어기의 장애나 네트워크 단절이 발생하면 전체 스케줄링이 중단되고 복구 시간이 지연된다. 또한 리더 선출(Leader Election)과 상태 동기화(State Synchronization) 과정에서 지연이 누적되어 p95/p99 꼬리 지연(Tail Latency) 이 급증하며, 시스템 전반의 성능 오버헤드가 증가한다. ~~노드가 늘어날수록 장애 전파 범위와 성능 변동성이 확대되어 정상 시에는 느림, 장애 시에는 광범위한 중단이라는 양극단 리스크가 상시화된다.~~

~~종합하면, 비용 절감을 위해 온프레미스를 선택해도 과다 프로비저닝으로 인한 유휴 자원 낭비가 지속되고,~~ 결국 중앙집중형 구조에서는 확장성과 복원력(Resilience) 을 동시에 확보하기 어렵고, 시스템 규모가 커질수록 장애 전파 범위와 성능 변동성이 기하급수적으로 확대된다. 또한, 중앙집중형 구조 탓에 안정성·가용성·성능의 근본 한계인 단일 장애점을 피하기 어렵다. 따라서 본 과제는 중앙 제어기에 대한 의존을 제거하고, 각 노드가 자율적으로 상태를 공유·관리하는 분산 자율 제어 평면(Decentralized Control Plane)의 필요성과 타당성을 규명하는 것을 목표로 한다.

2.2. 기존 연구와의 비교

본 절에서는 Mutual Cloud가 기존의 중앙형·프라이빗 클라우드 및 DAO 기반 분산 오케스트레이션 구조, Distributed Cloud-Edge와 어떻게 차별화되는지를 제어 평면 구조, 장애 복원 방식, 확장성, 운영 신뢰 모델의 네 가지 측면에서 비교한다. 비교의 목적은 단순한 관리 편의성이나 성능 지표가 아니라, 제어면(Control Plane) 자체의 구조적 탈중앙화와 자율 복구(Self-healing) 가능성을 검증하는 데 있다. 즉, Mutual Cloud가 중앙 제어기를 제거하면서도 동일한 수준의 안정성과 연속성을 유지할 수 있는지를 중심으로 평가한다.

1) 중앙형 클라우드와의 비교

~~중앙형 클라우드는 대기업이 하드웨어를 소유·운영하고 사용자는 이를 임대하는 구조다. 초기에는 확장성과 관리형 서비스를 바로 쓸 수 있는 장점이 있어 널리 쓰인다. 그러나, 시간이 지날수록 특정 밴더의 API·요금 체계에 묶이는 Vendor Lock-in이 심화되고, 최대 수요를 가정해 자원을 넉넉히 확보하는 over-provisioning 관행 때문에 유휴 자원이 쌓이면서 비용이 상승한다. 더 나아가 데이터 주권과 규제, 리전 제한으로 인해 데이터를 원래 위치에서 처리하기 어려워 불필요한 데이터 이동과 지연, 추가 비용이 발생한다.~~

중앙형 제어 평면은 네트워크 내 다수의 장치를 논리적으로 하나의 중앙 컨트롤러가 제어하는 구조를 따른다. 이러한 방식은 전역 최적화(Global Optimization)와 관리의 단순성을 제공하지만, 제어 기능이 중앙에 집중되어 있어 단일 장애점과 그로 인한 확장성 제약을 필연적으로 내포한다.

이 구조에서는 제어 평면의 물리적 분산을 통해 가용성과 부하 분산을 확보하려는 시도가 이루어졌지만, 여전히 논리적으로 중앙화된 제어 모델을 유지한다. 예를 들어, 참고문헌[1]의 경우, SDN(Software Defined Networking) 환경에서 제어 평면을 물리적으로 여러 컨트롤러로 분산하면서, 이들 간의 일관성을 유지하기 위해 데이터 플레인 자체를 트랜잭셔널 공유 메모

리(transactional shared memory) 로 사용하는 인밴드(In-band) 동기화 방식을 제안하였다. 이 접근법은 OpenFlow의 Bundling 기능과 Compare-and-Swap(CAS) 연산을 활용해 컨트롤러 간 정책 갱신을 원자적으로 처리함으로써, 충돌(conflict)을 방지하고 네트워크 상태의 일관성을 보장한다는 점에서 의미가 있다. 그러나 이러한 트랜잭션 중심 구조는 컨트롤러 간의 지속적인 합의(Consensus)와 동기화 절차를 필요로 하며, 제어 평면의 확장성 및 복원력 측면에서 본질적 제약이 존재한다.

이에 비해 Mutual Cloud는 중앙 제어기의 존재를 제거하고, DHT(Distributed Hash Table) 기반의 리더 없는 자율 제어 평면(Leaderless Decentralized Control Plane) 을 구성한다. 각 노드는 자신의 상태를 DHT 상에서 직접 기록·탐색하며, 전역 합의나 트랜잭션 동기화 없이도 상태 일관성을 유지할 수 있다. 이를 통해 Mutual Cloud는 중앙형 구조의 단일 장애점 문제를 구조적으로 제거하고, 노드 단위의 자율 복원력(Self-healing Resilience) 을 확보한다.
~~본 시스템은 참여자 각자가 가진 자원을 함께 쓰는 구조이다. 새로 사지 않아도 이미 갖고 있는 것을 효율적으로 공유해 추가 비용을 최소화하고, 어느 조직의 자원을 어디에 쓸지에 대한 정책 통제권을 참여자가 유지한다. 작업은 데이터가 있는 곳·네트워크가 유리한 곳으로 보내어 데이터 이동 비용을 줄이고, 장애나 중단은 자동 회수·재배치 규칙으로 복구한다. 비용은 실제 사용 기록에 근거해 후불로 투명하게 정산한다.~~

2) 프라이빗 클라우드와의 비교

기존 프라이빗 클라우드(Private Cloud) 는 단일 조직 내부 자원을 중앙 관리 노드(vCenter, Kubernetes 등)가 통합 제어하는 구조로, 가상화와 자동화 기술을 결합해 내부 인프라를 클라우드처럼 운용할 수 있도록 한다. ~~다만 확장은 물리 자원과 예산(CAPEX)에 좌우되고, 외부 조직과 자원을 정책적으로 공유하기가 어렵다. 유휴 자원은 조직 울타리 안에 묶여 남거 쉬워 Overprovisioning 비용이 누적되며, 정산은 대개 내부 차지백/쇼백으로 처리된다.~~ 이러한 중앙집중형 모델은 높은 보안성과 통제력을 제공하지만, 모든 자원이 단일 관리 계층에 종속되어 자율 분산 제어가 불가능하며, 확장성과 복원성 측면에서도 구조적 제약이 존재한다.

이에 반해 Mutual Cloud는 중앙 제어기에 의존하지 않고 각 노드가 독립적으로 자원을 관리·공유하는 탈중앙화 제어 평면(Decentralized Control Plane)을 채택한다. 노드는 동시에 자원의 공급자와 소비자로 동작하며, 분산 해시 테이블(DHT)을 통해 외부 자원을 탐색하고 연합한다. ~~중앙 컨트롤러에 의존하지 않고 분산 주소록과 상태 공유로 조건에 맞는 외부 노드를 자동 선택해 작업을 맡기며, 작업은 데이터가 있는 곳이나 네트워크가 유리한 곳으로 보내 이그레스 비용과 지연을 줄인다.~~ 제어 트래픽은 네트워크 규모에 선형적으로 증가하지 않아 확장성이 높고, 장애 발생 시 자율 복구(Self-Healing) 메커니즘을 통해 중앙 스케줄러 없이도 시스템 연속성을 유지한다. ~~어떤 자원을 어디에 쓸지는 소유자가 허용 워크로드·데이터 범위·자원 한도 등 정책으로 통제한다. 사용 기록에 근거해 후불·투명 정산이 이뤄져 CAPEX 부담을 낮추면서 필요 시 탄력적으로 확장할 수 있다. 규제·보안상 내부 실행이 필요한 영역은 프라이빗에 두고, 이동 가능한 배치·AI·ETL·버스트 트래픽은 제안하는 모델로 분산하는 병행 전략이 실용적이다.~~ 프라이빗 클라우드가 단일 조직 내 중앙집중형 관리에 기반한 폐쇄적 구조라면, Mutual Cloud는 다기관 간 상호 신뢰 기반의 개방형 자율 분산 모델이다. 전자는 내

부 통제력은 우수하나 확장성과 복원력이 제한되고, 후자는 중앙 의존을 제거함으로써 단일 장애점 없는 복원력(Resilience) 과 선형적 확장성(Scalability) 을 확보한다.

3) DAOnetes와의 비교

[1]DAOnetes는 블록체인(Solana) 기반의 스마트컨트랙트로 워크그룹을 만들고, 그 안에 승인된 기기들을 묶어 컨테이너 작업을 실행하는 모델이다. 신원과 권한은 블록체인으로 관리되고, 노드 간 통신은 WireGuard 기반 P2P VPN으로 구성되어 공개망과 분리된다. 온체인 규칙을 중심으로 운영되는 프라이빗 분산 실행 체계인 DAOnetes는 초기 설정과 운영이 다소 무거운 반면, 본 시스템은 블록체인을 사용하지 않고 분산 주소록으로 적합 노드를 찾아 매칭 후 배치한다. 상태 기반 자동 교체를 내장해 도입이 가볍고 기존 DevOps와 맞물린다. 따라서 불변 기록과 온체인 거버넌스가 중요한 환경에는 DAOnetes가, 신속한 도입·비용 절감·유휴 자원 활용 극대화가 중요한 환경에는 본 연구에서 제안하는 모델인 Mutual Cloud가 더 적합하다.

4) Distributed Cloud-Edge 인프라와의 비교

분산 클라우드-엣지 인프라를 위한 탈중앙화 SDN 제어 평면 관련 논문[2]에서는 IoT, NFV, Mobile Edge Computing과 같은 분산 서비스 환경에서 다수의 소규모 데이터센터(Edge Site) 를 연계하여 클라우드 서비스를 제공하는 Cloud-Edge 분산 인프라를 제안하였다. 해당 구조는 중앙 데이터센터 대신, 백본 네트워크 내 여러 지점(Point-of-Presence, PoP)에 리소스를 분산 배치함으로써 지연(Latency)과 처리 부하를 줄이는 것을 목표로 한다. 그러나 이러한 시스템 역시 SDN(Software Defined Networking) 기반의 중앙 Controller가 네트워크 정책과 라우팅을 통합 관리하기 때문에, 제어 평면이 완전히 분산되지는 않는다.

즉, 해당 논문[3]은 물리적 인프라 측면에서는 분산(Distributed)되어 있지만, 제어 구조는 여전히 SDN Controller 중심의 중앙집중형 논리 제어(Logic Centralization) 를 유지한다. 각 사이트의 Resource Manager는 독립적으로 동작하지 않으며, Controller 간 동기화나 정책 전파가 필수적이다. 따라서 시스템 규모가 커질수록 제어 트래픽 증가 및 컨트롤러 장애 시 복구 지연 문제가 발생한다.

반면, Mutual Cloud는 제어 자체를 SDN Controller에서 완전히 분리하고, 모든 노드가 동일한 권한으로 참여하는 DHT 기반 자율 제어 평면(Self-Managing Control Plane) 을 구현한다. 노드는 중앙 명령 없이도 자원의 등록, 탐색, 임대(Lease), 복구를 수행할 수 있으며, XOR 거리 기반 라우팅을 통해 제어 트래픽을 O(log N) 수준으로 억제한다. 따라서 Mutual Cloud는 Cloud-Edge 모델이 가진 “물리적 분산은 있으나 제어는 중앙화된 구조적 한계”를 넘어서는 논리적 탈중앙화(Full Decentralization) 를 실현한다.

---

1) https://github.com/workbenchapp/daonetes
2) D. E. Sarmiento, A. Lebre, L. Nussbaum, and A. Chari, "Decentralized SDN Control Plane for a Distributed Cloud-Edge Infrastructure: A Survey," IEEE Communications Surveys & Tutorials, vol. 23, no. 1, pp. 256-281, Jan. 2021.
3) 2)와 동일 논문

구분	중앙형 클라우드	프라이빗 클라우드	DAO 기반 분산 오케스트레이션 (DAOnetes)	Cloud–Edge 분산 인프라	Mutual Cloud
제어 구조 (Control Plane Architecture)	단일 중앙 컨트롤러가 모든 노드 제어 (Master–Worker 구조)	조직 내부 단일 관리 계층이 자원 통합 제어	블록체인(Solana) 기반 온체인 규칙 중심 제어, 노드는 WireGuard VPN으로 연결	SDN Controller 중심의 중앙 논리 제어(Logical Centralization)	DHT 기반 리더 없는 자율 제어 평면 (Leaderless Decentralized Control Plane)
장애 복구 방식 (Fault-Recovery Mechanism)	리더 선출 및 복제 기반 복원 (Master 장애 시 재선출)	내부 HA 구성으로 일정 수준 복원 가능하나 제어기 장애 시 전체 영향	온체인 규칙과 스마트컨트랙트 기반 승인·재실행, 속도 느림	Controller 간 정책 동기화 및 장애 시 수동 복구 필요	Lease–Heartbeat–Fencing 기반 자율 복구(Self-healing), 중복 실행 방지
확장성 (Scalability)	컨트롤러 부하 급증, 제어 트래픽 O(N) 증가	물리 자원/조직 경계 내 확장 제한 (CAPEX 종속)	온체인 트랜잭션 병목으로 확장 시 지연 발생	Edge Site 확장 가능하나 제어 트래픽은 중앙 집중	O(log N) 수준의 제어 트래픽, 노드 증가 시 지연 완만 (Scalable Resilience)
구조적 특징 (Structural Property)	전역 최적화 용이하나 구조적 SPOF 내재	폐쇄적 구조로 외부 자원 연합 어려움	불변 기록 및 거버넌스 투명, 설정 복잡	물리적 분산이나 제어 논리 중앙화 유지	완전한 논리적 탈중앙화(Full Decentralization), SPOF 제거, 자율 복원(Self-managing)

[표1] 타 클라우드/자원 공유 시스템과 비교

2.3. 제안 내용

본 연구에서는 P2P 기반의 프라이빗/커뮤니티 클라우드(Private/Community Cloud) 형태의 자원 공유 시스템인 'Mutual Cloud'를 제안한다. Mutual Cloud는 중앙집중형 클라우드의 가장 근본적인 한계인 단일 장애점 문제를 해결하기 위해 설계된 분산 자원 공유 시스템으로, 각 참여 노드가 자신의 유휴 컴퓨팅 자원을 공용 자원 풀에 제공하고 상호 간 작업을 분산 처리한다. 시스템의 주요 공유 자원은 CPU 및 GPU와 같은 컴퓨팅 자원이며, RAM과 로컬 스토리지는 작업 수행에 필요한 임시 메모리 및 스크래치 공간으로 활용된다. 본 시스템은 중앙 스케줄러나 마스터 노드 없이, 모든 제어 기능을 노드 간에 분산시켜 단일 장애점을 근본적으로 제거하도록 설계되었다. 이를 위해 DHT(Distributed Hash Table)를 활용한 리더 없는 자율 제어 평면(Leaderless Decentralized Control Plane)을 핵심 메커니즘으로 채택하였다.

시스템 아키텍처는 제어면(Control Plane)과 데이터면(Data Plane)으로 분리된다. 제어면은 각 노드가 자신의 상태와 작업 메타데이터를 DHT에 직접 기록·조회함으로써 중앙 제어기 없이 분산된 상태 관리와 스케줄링을 수행한다. 데이터 면은 실제 작업 데이터 및 결과물을 노드 간 P2P 방식으로 직접 전송하여 전송 효율성을 극대화한다. 이러한 구조는 특정 제어 노드의 장애가 전체 시스템의 중단으로 이어지는 중앙집중형 구조의 취약점을 구조적으로 제거하며, 제어와 데이터 경로를 분리하여 높은 가용성과 연속성을 확보한다.

본 시스템의 핵심 목표는 다음과 같다. ~~첫째, 온프레미스(On-premise) 환경의 유휴 자원의 활용 극대화한다.~~ 첫째, 중앙 제어기(Control Node)의 장애나 네트워크 단절이 발생하더라도, 작업의 탐색·할당·복구가 중단되지 않고 지속될 수 있는 완전 분산 제어 구조를 구현한다. 이를 위해 DHT 기반의 노드 탐색, 분산 리스(Lease) 관리, 상태 복제를 적용하여 제어 기능을 네트워크 전반으로 분산시킨다.

둘째, 중앙 제어기 부재 상황에서도 각 노드가 스스로 장애를 감지하고, 작업을 재할당하거나 이어받을 수 있는 자율 복원(Self-recovery) 구조를 확보한다. 이를 통해 장애 상황에서도 서비스 연속성과 데이터 일관성을 유지한다. ~~셋째, 신뢰할 수 없는 환경에서도 보안과 격리를 유지하고, 제어기의 부재 상황에서도 작업 제출부터 결과 반환까지 일관성 있는 사용자 경험을 제공한다.~~ 결과적으로, 본 시스템은 중앙 제어기 의존으로 인한 병목과 유휴 자원 방치를 제거하여 전체 자원 활용률을 극대화한다.

2.3.2. on-prem 유휴 자원 & 퍼블릭 비용 비효율 → P2P 가상 자원 풀 + 분산 매칭

온프레 노드를 P2P 오버레이로 묶어 하나의 가상 자원 풀로 보이게 하고, 중앙 스케줄러 없이 DHT 기반 분산 매칭으로 작업을 배분한다. 수요서버는 원장(System of Record)에서 대기 작업을 감지해 DHT에 경량 메타데이터로 공지하고, 에이전트는 이를 조회해 자율 선점 후 실행한다. 이로써 조직 내/간 자원을 수평적으로 풀링하여 유휴를 흡수하고, 운영은 제어면(control plane)=메타데이터(DHT) / 데이터면=P2P 전송의 분리 원칙으로 단순화한다. (범위: 주 공유=CPU/GPU, 부가=RAM·SSD/HDD의 실행 시 임시 용도)

2.3.3. 중앙집중형 아키텍처 리스크(SPOF·병목) → 중앙 없는 제어면 + P2P 데이터면

매칭·상태 관리를 DHT의 분산 상태로 이관해 중앙 스케줄러를 제거하고, 에이전트는 리스/TTL/하트비트 규칙으로 자기치유(장애 시 자동 재선점)를 수행한다. 무거운 데이터(이미지·대용량 입력/산출)는 노드 간 P2P로 직접 전송·캐시하고 중앙 구성요소는 어떤 데이터가 어디 있는가만 기록한다. 그 결과, SPOF와 중앙 병목이 구조적으로 배제되고, 노드가 늘어날수록 매칭 부하와 데이터 트래픽이 수평 분산된다.

2.3.4. 운영·보안·검증 개요

운영은 네임스페이스 분리로 환경 간 간섭을 줄이고, 부트스트랩 피어(복수) 를 상시 유지해 DHT 합류를 안정화한다. 보안은 오버레이 암호화(WireGuard) 와 실행 격리(Kata Containers) 로 강화하고, 자격증명·비밀은 DHT가 아닌 안전 저장소에 둔다. 품질 관리는 상태 전이, 리스 만료/재선점, 대기·처리 지연 등의 핵심 지표를 관측하며, 검증은 단일 서버 대비 P2P 가상 풀에서의 처리 흐름·지연·장애 복구 동작과 중앙 네트워크/스토리지 부하 변화를 비교 관찰한다.

2.4. 기대 효과 및 의의

본 연구에서 제안하는 모델 'Mutual Cloud'는 중앙 집중형 제어 구조의 단일 장애점을 근본적으로 제거하여 시스템 안정성을 달성하는 것을 목표로 한다. 뮤츄얼 클라우드는 조직의 유휴 자원을 파악하고 최적의 노드를 탐색하여 수요 자원을 배치하는 지능형 메커니즘을 활용한다. 이를 통해 조직은 퍼블릭 클라우드 이용료나 추가적인 온프레미스 서버 증설 비용 없이, 기존에 소유한 자산을 활용하여 컴퓨팅 파워를 확보한다. 특히, 본 연구는 유휴 자원 탐지-배치 메커니즘을 통해 퍼블릭 클라우드 대체 및 보완 시 총소유비용(TCO)을 동일 워크로드 기준으로 ΔC≈40% 감소시키는 것을 목표로 한다. 기존 클라우드 인프라는 모든 제어 로직이 중앙 제어기에 집중되어 있어 장애 발생 시 전체 서비스가 중단되거나, 확장 시 제어 부하가 병목으로 작용하는 구조적 한계를 가진다. Mutual Cloud는 이러한 제어 평면의 중앙 의존성을 분산 제어 구조로 대체함으로써, 장애·확장·운영 측면의 근본적 취약점을 해소한다. 이러한 경제적 효과는 개별 연구원이나 소규모 연구실이 로컬 서버의 한계를 넘어 필요 시 즉시 더 높은 컴퓨팅 자원을 활용하게 하여 연구 및 개발 속도를 향상시킨다. 이는 비싼 퍼블릭 클라우드와 비효율적인 온프레미스라는 양자택일의 딜레마에서 벗어나, 지속 가능한 자체 컴퓨팅 인프라를 구축할 수 있는 새로운 모델을 제시한다. 궁극적으로, 조직이 소유한 자원을 직접 통제하고 활용함으로써 기술적, 경제적 자원 주권(Resource Sovereignty)을 확보할 수 있는 가능성을 실증한다.

1. 구조적 안정성과 고가용성 확보

Mutual Cloud는 P2P + DHT(Distributed Hash Table) 기반의 리더 없는 제어 평면(Leaderless Control Plane)을 구성하여, 중앙 제어기의 장애에도 전체 서비스가 지속되도록 단일 장애점 없는 구조를 실현한다. 노드 간 상태 정보가 DHT에 분산 복제되어 저장되므로, 특정 노드가 오프라인이 되더라도 다른 노드가 동일한 정보를 즉시 조회하고 복구할 수 있다. 이러한 구조는 중앙 서버가 없는 환경에서도 서비스 성공률 ≥99.99%, 평균 복구시간(MTTR) 0.1초 이하 수준의 고가용성을 보장하도록 설계되었다. 또한 각 노드는 주기적인 하트비트 및 상태 해시(State Hash)를 교환하여 장애를 자율 감지하고, 체크포인트(Checkpoint)를 통해 작업을 이어받음으로 Control Node가 부재한 상황에서도 자율 복구(Self-healing)가 가능하다. 이를 통해 중앙 복구 절차나 리더 선출 과정 없이 평균 복구 시간(MTTR) ≤2분 수준의 신속한 복원력을 확보한다.

2. 확장성 및 제어 부하 분산

기존 중앙 제어 구조에서는 노드 수나 작업량이 증가할수록 제어기에 트래픽과 스케줄링 요청이 집중되어 병목이 발생했다. Mutual Cloud는 DHT 기반의 분산 탐색·매칭 메커니즘을 적용해 제어 로직 자체를 네트워크 전반으로 분산시킴으로써, 제어 평면의 부하를 구조적으로 분산시킨다. 이로써 노드 수가 N → 10N으로 증가하더라도 O(log N) 수준의 탐색 복잡도를 유지하고, 평균 탐색 지연은 ≤5ms , 스케줄링 대기시간 ≤ 10ms 범위 내에서 억제된다. 결과적으로 시스템 규모가 커져도 제어 부하가 한 지점에 집중되지 않으며, 중앙 병목 없이 선형적 확장성을 보장한다.

3. 자가 복원(Self-healing Resilience) 및 일관성 유지

Mutual Cloud의 자가 복원 메커니즘은 중앙 제어기가 없는 상태에서도 시스템이 스스로 장애를 감지하고 복구할 수 있도록 설계되었다. 각 노드는 lease 만료 및 하트비트 단절을 감시하며 장애를 탐지하고, 작업의 중간 상태를 체크포인트(Checkpoint)로 저장하여 다른 노드가 즉시 이어받아 실행할 수 있다. 이를 통해 노드 장애율이 총 노드 수의 절반인 50%에 달하는 환경에서도 전체 작업 성공률 ≥99.9%, 복구 지연 시간 ≤ 0.1초 수준의 자율 복원력을 달성할 것으로 기대된다. 즉, 단일 장애점 제거는 단순한 가용성 향상을 넘어 시스템이 스스로 복구(Self-recovery)하고, 중앙 통제 없이 자율적으로 운영되는 구조적 복원력(Self-healing Resilience)을 제공한다. ~~동시에, NAT 환경, 데이터 로컬리티 실패 시 비용 증가, 조직 정책 충돌과 같은 현실적 한계를 병기하여 적용 범위를 명확하게 규정한다. 뮤츄얼 클라우드는 P2P와 DHT 기술이 실제 조직의 IT 인프라 문제를 해결할 수 있는지에 대한 구체적인 적용 사례를 제시하며, 차세대 온프레미스 아키텍처의 청사진을 제안한다는 점에서 중요한 기술적 의의를 가진다.~~

2.5. 주요 기능 리스트

본 논문에서 제안하는 Mutual Cloud는 ~~경제적 효율성과 운영 안정성이라는 두 가지 목표를 달성하기 위해 상호 유기적으로 동작하는 네 가지 핵심 기능 모듈로 구성된다~~ 단일 제어점을 제거하고, 자율 복원(Self-healing) 및 확장 가능한 분산 제어 구조를 구현하기 위해 설계된 시스템이다.

이 시스템은 중앙 제어기에 의존하지 않고 상호 자율적으로 동작하는 세 가지 핵심 기능 모듈로 구성된다. 각 기능은 P2P 통신과 DHT(Distributed Hash Table)를 기반으로 하여 제어 기능을 네트워크 전반에 분산시킴으로써, 기존 중앙형 시스템이 가진 구조적 병목과 단일 장애점을 근본적으로 해소한다.

1. 탈중앙화 노드 탐색 및 상태 복제 (Decentralized Node Discovery and State Replication)

본 모듈은 중앙 제어기(Control Node)의 장애가 네트워크 전체의 자원 탐색 기능에 영향을 주지 않도록 설계된 단일 장애점 없는 구조의 핵심 구성요소이다. 중앙의 관리 서버 없이 네트워크에 참여하는 모든 노드의 상태와 자원 정보를 DHT를 통해 관리하고, 새로운 노드의 참여나 이탈을 자율적으로 처리한다. 각 노드는 자신의 상태(예: 유휴/작업 중)와 자원 사양(CPU, GPU, Memory)을 주기적으로 DHT에 '발행(Publish)'하고, 다른 노드는 작업 요청 시 특정 조건(예: 'GPU=True, Status=Idle')으로 '질의(Query)'를 수행하여 후보 노드 목록을 확보한다. 이때 모든 상태 정보는 DHT 상에서 다중 노드에 복제(Replicated) 되어 저장되므로, Control Node가 일시적으로 중단되더라도 탐색 기능은 지속적으로 유지된다. 이러한 구조는 중앙 관리자에게 집중되던 자원탐색 및 상태 관리 기능을 네트워크 전체로 완전히 분산시켜, 제어 평면에서의 단일 장애점을 근본적으로 제거한다. 또한 노드 수가 증가하더라도 O(log N) 수준의 탐색 복잡도를 유지함으로써, 시스템 확장 시에도 안정적 성능을 보장한다. 이를 검증하기 위해, 노드 수를 100개에서 10,000개까지 점진적으로 증가시키는 시뮬레이션 환경을 구축하였다. 각 실험에서는 임의의 노드가 특정 자원 조건을 가진 다른 노드를 1,000회 탐색하도록 구성하고, 평균 조회 지연 시간과 조회 성공률을 측정하도록 설계하였다.

~~2. 작업 매칭 및 할당 (Task Matching and Assignment)~~

~~사용자가 요청한 작업의 필수 요구사항과 DHT를 통해 탐색된 유휴 노드들의 자원 상태를 비교하여, 실행 가능한 후보 노드 풀(Pool)을 동적으로 생성하고 작업을 최종 할당한다. 이 기능은 사용자의 작업 요구사항(예: GPU 필수, 최소 RAM 16GB)을 만족하는 유휴 노드 목록을 DHT를 통해 필터링하여 후보 풀을 생성한다. 작업 할당은 이 후보 목록 내에서 사전에 정의된 단순 정책(예: 랜덤 선택, 최초 응답 노드 우선)에 따르거나 사용자에게 선택권을 위임하는 방식으로 이루어진다. 즉, 본 시스템은 어떤 정책이든 적용할 수 있는 유연한 '매칭 프레임워크'를 탈중앙화된 방식으로 구현하는 데 초점을 맞춘다. 이때, 자원 활용률 개선 효과를 검증하고자, 50개 노드의 가상 연구실 환경에 실제와 유사한 워크로드(주간-바쁨, 야간-한가)를 적용한다. 이후 24시간 동안 뮤츄얼 클라우드 시스템을 적용했을 때와 적용하지 않았을 때의 전체 시스템 평균 자원 활용률(ΔU≈60%p 향상)과 총 작업 처리량을 비교 분석한다.~~

2. 분산 작업 매칭 및 임대 관리 (Distributed Task Matching and Lease Management)

본 모듈은 중앙 스케줄러 없이 각 노드가 자율적으로 작업을 탐색하고 할당하는 분산 스케줄링 구조를 구현한다. 기존 중앙형 오케스트레이션 시스템에서는 모든 작업이 중앙 제어기의 스케줄러를 통해 분배되기 때문에, 제어기의 장애나 과부하가 전체 시스템의 병목으로 이어진다. 이에 반해 Mutual Cloud는 DHT 기반의 분산 매칭 알고리즘을 적용하여 제어기(Control Node)가 존재하지 않더라도 작업 탐색과 임대(Lease) 갱신이 노드 간 상호 협상 방식으로 이

루어진다. 작업 요청이 발생하면 요청 노드(Consumer)는 DHT에 등록된 자원 메타데이터를 기반으로 실행 가능한 후보 노드 풀(Pool)을 구성하고, 후보 중 하나를 선택하여 임대 요청을 전송한다. 임대가 승인되면 해당 노드는 작업을 점유(Claim)하고, lease가 유지되는 동안 Control Node의 개입 없이 자체적으로 상태를 갱신한다. Control Node가 일시적으로 중단되더라도 lease 정보는 DHT에 분산 저장되어 있으므로, 임대 만료나 재할당은 다른 노드에 의해 자동으로 처리된다. 이를 통해 단일 제어점에 집중되던 작업 배분 로직이 네트워크 전반으로 분산되어, 제어 평면의 단일 장애점 문제를 근본적으로 제거하였다.

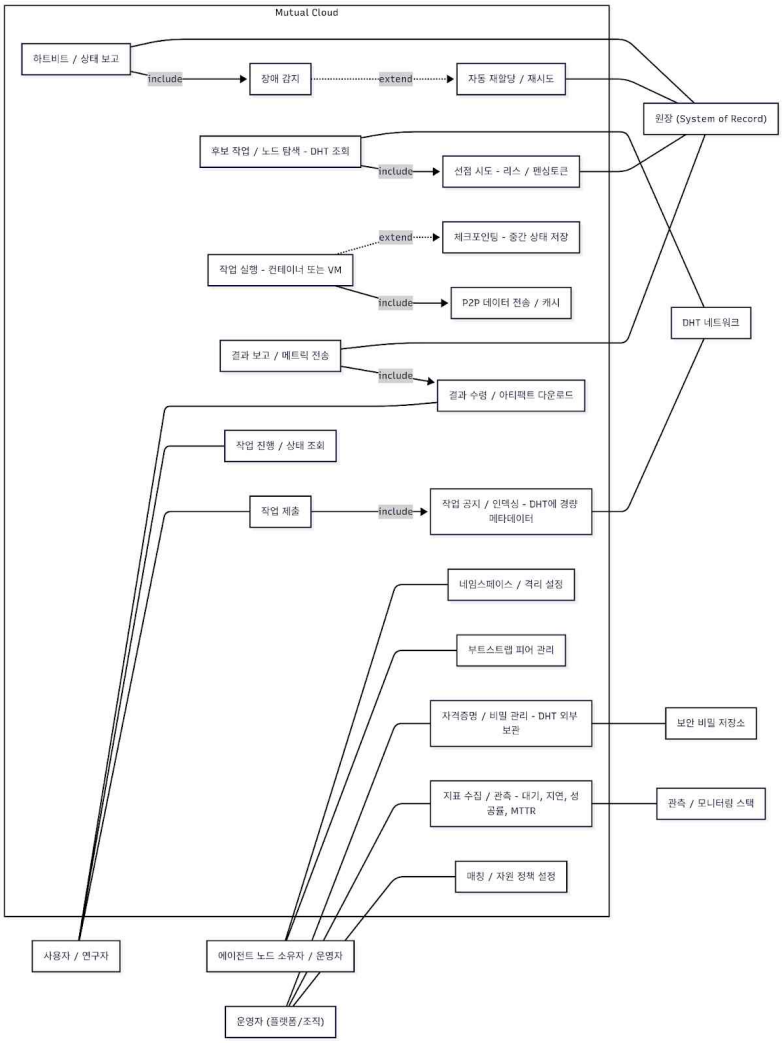
3. 자가 치유 기반 장애 감지 및 복구 (Self-healing Failure Detection and Recovery)
본 모듈은 중앙 제어기 없이도 노드 장애를 자동으로 감지하고 복구하는 자가 치유 구조를 담당한다. Mutual Cloud의 각 노드는 주기적으로 하트비트(Heartbeat)와 상태 해시(State Hash)를 교환하여 장애 발생 여부를 감시하며, 특정 노드의 하트비트가 일정 주기 이상 응답하지 않으면 해당 노드를 장애 상태로 간주한다. 장애가 감지되면, 해당 노드가 수행 중이던 작업은 즉시 '실패(Failed)' 상태로 전환된 후 DHT 상의 자원 테이블을 통해 다른 유휴 노드에 재할당된다. 작업의 중간 결과는 체크포인트(CheckPoint)로 저장되어, 새로운 노드가 Control Node의 개입 없이 중단된 시점부터 작업을 이어받아 실행을 재개할 수 있다. 이를 통해 중앙 복구 절차나 재시작 신호 없이도 전체 시스템이 자율적으로 복원되며, 제어기 장애 및 노드 장애가 동시에 발생하더라도 서비스 연속성이 유지된다. ~~어러한 내결함성 매커니즘의 실효성을 평가하기 위해, 100개의 노드가 1,000개의 작업을 처리하는 환경을 구축한다. 작업이 활발히 처리되는 도중, 무작위로 작업 수행 노드의 10%를 강제 종료하는 장애 주입(Fault Injection) 실험을 수행한다. 이때 작업 성공률과 평균 복구 시간을 측정하여 최종적으로 성공한 작업의 비율과 해당 작업이 다른 노드에서 재시작되기까지 걸리는 평균 시간을 도출한다.~~

상기 세 모듈은 각각 제어, 스케줄링, 복구 단계에서의 단일 장애점을 제거함으로써, Mutual Cloud가 중앙 제어기 부재 환경에서도 독립적으로 동작 가능한 완전 분산 제어 구조(SPOF-Resilient Control Plane)를 실현하였다. 본 구조는 제어 로직, 상태 정보, 복구 절차를 모두 네트워크 전반으로 분산시켜, 기존 중앙형 클라우드 아키텍처의 구조적 취약점을 근본적으로 해소한다.

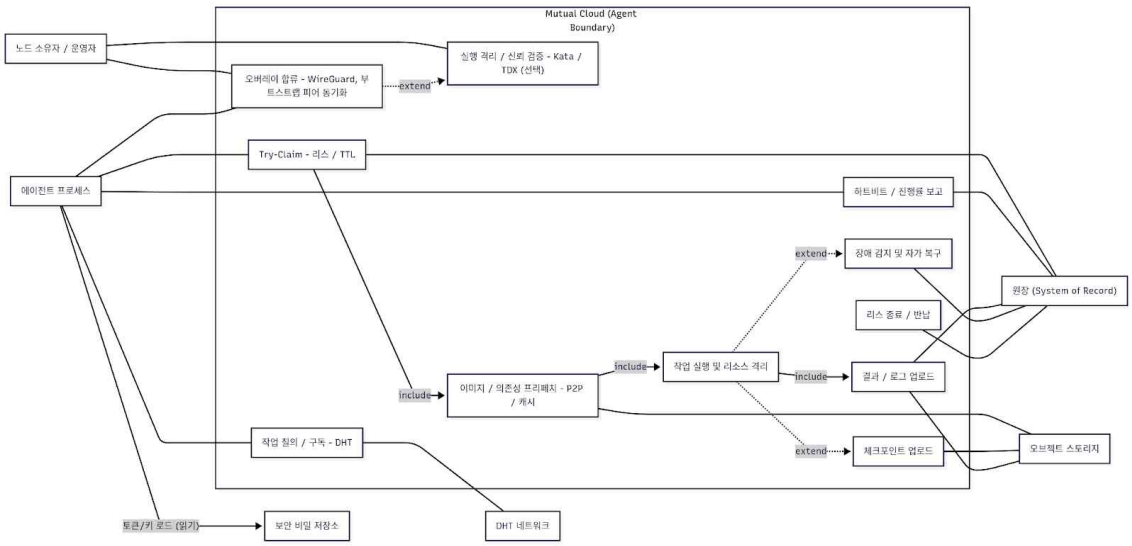
3. Project-Design 과제 설계

3.1. 요구사항 정의

Mutual Cloud는 중앙 제어기의 개입 없이도 동작 가능한 단일 장애점 없는 구조를 목표로 설계된, 분산 환경 기반의 자율 자원 공유 프레임워크이다. 이 시스템은 다수의 에이전트 노드가 참여하여 작업 탐색, 분산 매칭 및 임대 관리, P2P 데이터 전송, 자가 치유 기반의 내결함성 확보를 수행한다. 본 정의서는 상위 수준 유스케이스와 기능별 요구사항을 포함하며, 이를 통해 시스템이 달성해야 할 단일 장애점 없는 제어 구조와 이를 통한 자율 복원 목표를 구체화하고, 사용자·노드 소유자·운영자 간 분산 제어에 대한 공통 기준을 제공한다.

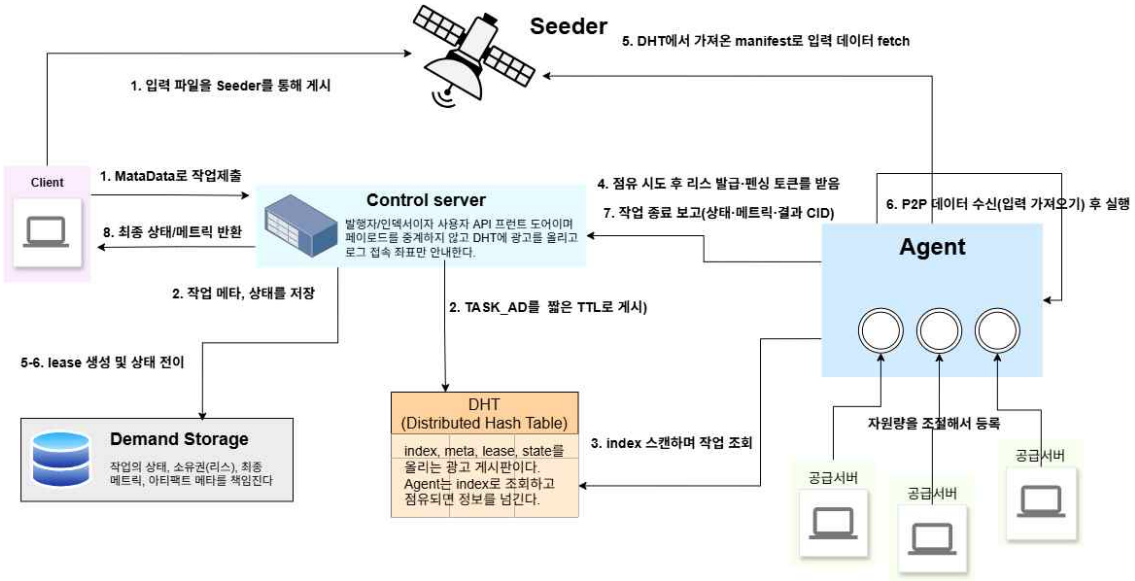


[그림 1] 상위 수준 유스케이스



[그림 2] 에이전트 관점 단일 장애점 제거 구조 상세

3.2. 전체 시스템 구성



[그림 3] 전체 시스템 구성

본 시스템의 구성 요소는 다음과 같다.

• Demand Storage(DB): 작업의 상태, 소유권(리스), 최종 메트릭, 아티팩트 메타를 책임진다. Control 노드가 일시적으로 중단되더라도 DB는 작업 상태의 영속적 기준(Persistent Ground Truth)으로 남아 재시작 시 전체 작업 일관성을 복원하는 역할을 수행한다.
• Control: 발행자/인덱서이자 사용자 API 프런트 도어이며 페이로드를 중계하지 않고 DHT에 광고를 올리고 로그 접속 좌표만 안내한다.
• DHT: 작업 발견/광고(announce/discover) 와 완료 신호 미러에만 쓰이고 락/순서/원자성은 책임지지 않는다. Control은 DHT에 작업 광고(announce)를 등록하고 로그 접속 좌표를 안내하지만, 실제 데이터 페이로드나 스케줄링 로직은 중계하지 않는다.
• Agent : DHT로 작업을 발견하고 Demand에서 try-claim(리스+펜싱토큰)을 받아 실행하며 데이터는 Object Storage와 직접 교환한다. Control 노드의 개입 없이도 다른 노드와 직접 상태 동기화 및 복구를 수행하며, 작업 데이터는 Object Storage와 직접 교환한다. Agent는 자체 하트비트(Heartbeat)와 체크포인트(Checkpoint)를 통해 장애 시 자가 복원(Self-recovery)을 수행한다.
• 실시간 로그: 사용자 <—> 에이전트 직결(WS/gRpc)이고 Control은 에이전트 주소• 토큰을 반환하는 역할만 한다. Control은 단지 에이전트의 주소 및 인증 토큰을 반환하는 역할만 수행하며, 데이터 경로에는 관여하지 않는다. 이를 통해 제어 경로와 데이터 경로가 완전히 분리되어, 로그 수집 과정에서의 단일 장애점을 제거한다.

3.3 전체 시스템 구성 및 동작 흐름

본 절에서는 Mutual Cloud의 전체 동작 흐름을 단계별로 기술한다. 시스템은 사용자(Client),

Control Node, Demand Storage(DB), DHT(Distributed Hash Table), Agent Node, 그리고 실시간 로그 채널(Log Stream) 로 구성된다. 모든 제어·탐색·데이터 전송은 중앙 서버를 경유하지 않으며, P2P 기반의 단일 장애점이 없는 구조로 설계되었다.

(1) 사용자 → Control : 작업 등록 (Task Submission)
사용자는 Control Node의 REST API(`/api/tasks`)를 통해 작업 메타데이터(요구 자원, 우선순위, 만료 시간 등)를 등록한다. Control은 내부적으로 Demand Storage(DB)에 `/jobs` 레코드를 생성하고, job_id, 에이전트 공개키, 시더 허용 정책 등 P2P 지시 정보를 사용자에게 반환한다. Control은 작업 페이로드를 중계하지 않으며, 메타데이터 등록 및 인덱싱(Indexing)만 수행한다.

(2) 사용자(Seeder) 준비 및 시딩 (Client-side Seeding)
사용자는 입력 데이터를 AES-GCM 방식으로 암호화 및 조각화한 뒤, Merkle DAG 구조로 연결하여 manifest_root_cid를 생성하고, 이를 기반으로 libp2p/QUIC 프로토콜로 시딩(Seeding)을 시작한다. 이후 `POST /jobs/{id}/manifest {cid, seeders[], enc_meta}` 요청을 통해 Demand Storage(DB)에 입력 데이터의 루트 CID 및 암호화 메타정보를 등록한다. 이 단계에서 데이터는 중앙 저장소를 거치지 않고 시더(Client)가 직접 유지한다.

(3) Control → DHT 광고 (Task Advertisement), Demand 기록
Control은 TASK_AD {job_id, demand_url, topic, exp, sig} 와 p2p/{job_id}/manifest {cid, seeders[], enc_meta} 를 DHT에 짧은 TTL로 게시한다. 이 광고(announce)는 DHT 네트워크와 Demand에 분산 복제되어 저장되며, Control 노드가 중단되더라도 TTL 기간 동안 탐색이 유지된다. DHT는 단순한 인덱스 레이어로, 락(lock)·순서(order)·원자성(atomicity)을 관리하지 않는다.

(4) 에이전트 발견 및 클레임 (Agent Discovery & Claim)
Agent Node는 DHT를 주기적으로 조회(Discover)하여 새로운 작업 광고를 탐색한다. 작업이 발견되면 Demand Storage의 `/try-claim` API를 호출해 리스(Lease) 및 펜싱 토큰(Fencing Token) 을 획득하고, 상태를 `assigned → running`으로 전환한다. 이 시점 이후부터는 Control의 개입 없이 Agent가 자율적으로 작업을 수행한다.

(5) 에이전트 P2P 수신 및 실행 (Task Fetch & Execution)
Agent는 DHT/Demand에서 수신한 manifest_root_cid 와 시더 목록(seeders[]) 을 참조하여, P2P 채널(libp2p/QUIC) 을 통해 Seeder로부터 데이터를 직접 받아온다. 수신된 데이터는 복호화 후 컨테이너 환경에서 실행된다. 컨테이너 실행 중 수집된 자원 사용량(CPU, Memory, I/O) 은 Agent 내부에서 집계되며, Control은 실행 경로나 데이터 스트림에 관여하지 않는다.

(6) 실시간 로그 직결 (Real-time Log Streaming)
사용자가 `GET /api/tasks/{id}/logs` 를 호출하면, Control은 해당 작업을 수행 중인 Agent의 WebSocket 엔드포인트와 인증 토큰을 반환한다. 이후 사용자는 이 정보를 이용해 Agent

와 직접 WebSocket(gRPC) 세션을 수립하며, 실시간 로그 및 상태 정보를 수신한다. Control은 초기 연결(Handshake) 과정에만 관여하고, 이후 데이터 스트림에는 개입하지 않는다. 따라서 Control 장애 시에도 기존 WebSocket 세션은 유지되며, 제어 경로(Control Path)와 데이터 경로(Data Path)가 완전히 분리된 단일 장애점이 없는 구조를 실현한다.

(7) 종료 보고 및 상태 갱신 (Result Reporting)
작업이 완료되면, Control은 Demand Storage의 `POST /jobs/{id}/finish` API를 호출하여 작업 상태(succeeded/failed), 최종 메트릭(cpu_avg, mem_peak 등), 결과물의 CID 목록(result_root_cid, artifacts[]) 을 보고한다. 이 보고는 멱등(Idempotent) 하게 설계되어 중복 전송에도 일관된 결과를 보장한다. Demand Storage(DB) 가 상태의 기준으로 동작하며, Control의 장애 여부와 무관하게 결과가 기록된다.

(8) 최종 조회 및 결과 수신 (Result Retrieval)
사용자는 `GET /api/tasks/{id}` 로 최종 상태와 메트릭을 조회한다. 결과물은 CID로 반환되며, 사용자는 P2P 경로를 통해 Seeder 또는 Agent로부터 직접 다운로드 한다.

(9) 재큐잉 및 타임아웃 관리 (Requeue & Timeout)
Control의 requeueLoop 프로세스는 Demand DB의 상태를 주기적으로 점검하여 `assigned`, `running`, `timeout` 상태의 작업을 감시한다. 만료되거나 실패한 작업은 정책에 따라 queued 상태로 되돌리거나 다른 노드에 재할당된다. 리스(Lease) 만료 판단은 DB 기준으로 수행되며, Control이 중단되더라도 Agent가 독립적으로 재시도(try-claim) 를 수행할 수 있다.

~~10) 보존·정리~~
~~서터 유지 시간(예: 완료 후 N서간/일) 정책으로 클라이언트 서터는 중단된다.DHT 레코드는 TTL 만료로 자연 소거되고 Demand DB는 보존기간 후 아카어브/삭제된다.~~

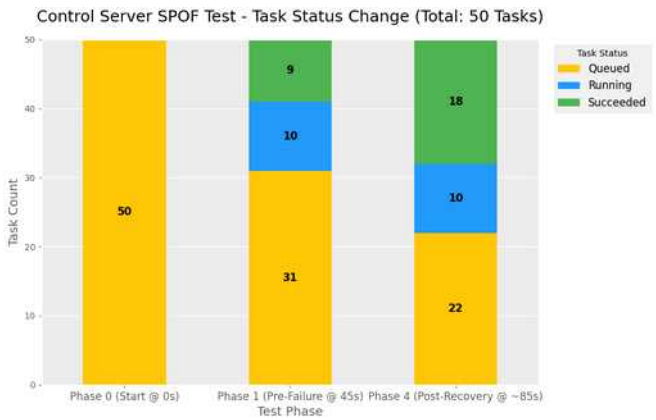
위와 같이 Mutual Cloud의 전체 워크플로우는 Control Node 혹은 Agent Node의 장애나 네트워크 단절 상황에서도 Agent-DHT-DB 축을 중심으로 자율 순환(Self-managed Loop) 되도록 설계되었다. 작업 생성, 탐색, 실행, 보고, 정리의 모든 단계가 P2P와 DHT에 의해 분산되어 제어 평면과 데이터 경로 모두에서 단일 장애점을 제거한다.

3.4. 시스템 검증 실험
3.4.1. SPOF 제거 검증 실험 (SPOF Elimination Test)
SPOF란 Single Point Of Failure의 약자로 시스템의 특정 구성 요소가 고장났을 때 전체 시스템의 가동이 멈추는 것을 의미한다. 중앙 집중형 아키텍처 체제인 기존 클라우드는 SPOF 문제가 발생하면 클라우드에 의존하는 모든 인프라가 동시에 멈추는 일이 일어날 수 있다. 본 시스템은 P2P 분산형 아키텍처이기 때문에 기존 중앙 집중형 클라우드의 문제를 해결 가능하다.

Mutual Cloud 시스템에서 SPOF 문제가 발생할 수 있는 지점은 Control 서버와 작업의 상

태를 저장하는 demand DB이다. Control 서버에 장애가 발생하면 demand DB에서 상태를 롤백해서 연쇄적 붕괴를 막을 수 있다. demand DB에 장애가 발생하여 데이터가 다 날아간다면 미리 이중화시킨 데이터베이스로 데이터 복구가 가능하다. 시스템의 DB로 구축한 PostgresSQL은 데이터베이스 복제화(Replication) 가능을 내장하고 있다. 모든 작업 생성, lease 갱신, 완료 기록은 단일 primary에서 일관성있게 처리하고 replication은 조회만 가능하게 구성하여 장애 발생 시 읽기 전용 replication을 통해 서비스 연속성을 유지할 수 있다.

본 실험은 P2P 환경에서 제어 서버(Control Node)의 장애 발생 시 Agent가 독립적으로 작업을 완료하고, 시스템이 정상적으로 복구될 수 있는지를 검증하기 위해 수행되었다. 즉, 제어 평면이 단일 장애점(SPOF)이 아님을 실증적으로 확인하는 것이 목적이다. 실험 환경은 Control 서버 1개, Seeder 1개, Agent 10개로 구성하였으며, 총 50개의 작업(각 30초 실행)을 제출한 뒤 75초 시점에서 Control 서버를 SIGKILL로 강제 종료하였다. 이후 30초간 Control 서버가 다운된 상태에서도 Agent들은 작업을 독립적으로 수행하고, Control 서버 다운을 감지한 후 완료결과를 주기적으로 전송하였다. 장애 구간(75~105초)이 끝난 후, succeeded 상태의 작업이 지속적으로 증가하여, Agent들이 제어 서버 개입 없이 자율적으로 작업을 완료했음을 확인할 수 있었다. 또한 Control 서버 재시작 직후(105초 이후) queued 및 running 상태의 작업이 정상적으로 변환되며, Control 서버 재시작 시(DB 동기화 후) 모든 상태는 즉시 복원되었다.



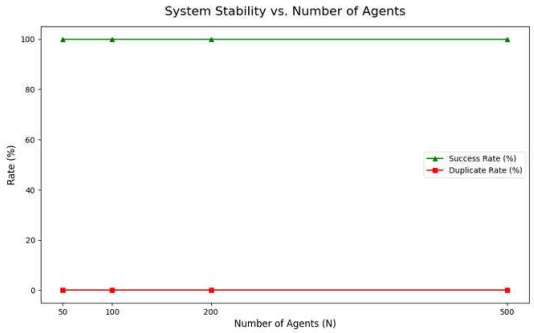
[그림 4] 제어 서버 장애 및 복구 과정에서의 작업 상태 변화

그림 4은 제어 서버 장애 및 복구 과정에서의 작업 상태 변화를 보여준다. 장애 구간(75~105초) 동안에도 succeeded 상태의 작업이 지속적으로 증가하여, Agent들이 제어 서버 개입 없이 자율적으로 작업을 완료했음을 확인할 수 있었다. 또한 Control 서버 재시작 직후(105초 이후) queued 및 running 상태의 작업이 정상적으로 변환되며, 이는 제어 서버가 즉시 DB로부터 상태를 복구하여 중단 없는 운영을 재개함을 보여준다. 결과적으로 Mutual Cloud의 제어 평면은 Control 서버 장애 시에도 전체 시스템의 연속성과 작업 일관성을 유지하는 Non-SPOF 구조임을 실험적으로 검증하였다.

3.4.2. 확장성 평가 (Scalability Evaluation)

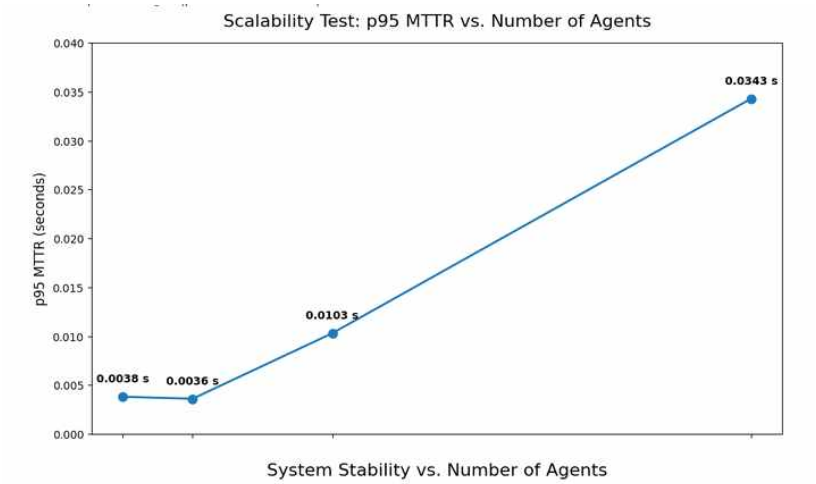
클라우드 환경은 수많은 노드, 네트워크, 스토리지, 컨테이너, API가 상호 작용하는 대규모 분

산 구조로, 부분 장애나 지연이 빈번하게 발생한다. 따라서 단순히 정상 동작하는 시스템보다, 예측 불가능한 장애에도 전체 서비스가 중단되지 않는 견고성(Robustness) 이 핵심이다. 본 절에서는 Mutual Cloud의 견고성과 확장성을 실험적으로 검증하였다.
확장성 실험은 제어 평면의 성능이 노드 수 증가에 따라 어떻게 변화하는지를 평가하기 위해 수행되었다. 전체 에이전트 수(AGENTS)를 50, 100, 200, 500으로 확장하고, 작업 수(TASKS)도 이에 비례하여 증가시켰다. 모든 실험은 동일 환경에서 4회 반복 수행되었으며, 복구 시간의 95번째 백분위수(MTTR p95) 를 주요 지표로 사용하였다.



[그림 5] 에이전트 수(N) 증가에 따른 시스템의 작업 성공률과 중복 실행률의 변화

그림5는 에이전트 수(N) 증가에 따른 시스템의 작업 성공률(Success Rate, 녹색 선)과 중복 실행률(Duplicate Rate, 빨간 선)의 변화를 보여준다. 에이전트 수가 50에서 500으로 10배 증가하고 장애 노드 수도 5개에서 50개로 늘어났음에도 작업 성공률은 100%를 유지되었으며, 중복 실행률은 0%로 안정적이었다. 이는 시스템이 확장되는 환경에서도 작업 처리의 정확성과 신뢰성을 보장함을 입증한다.



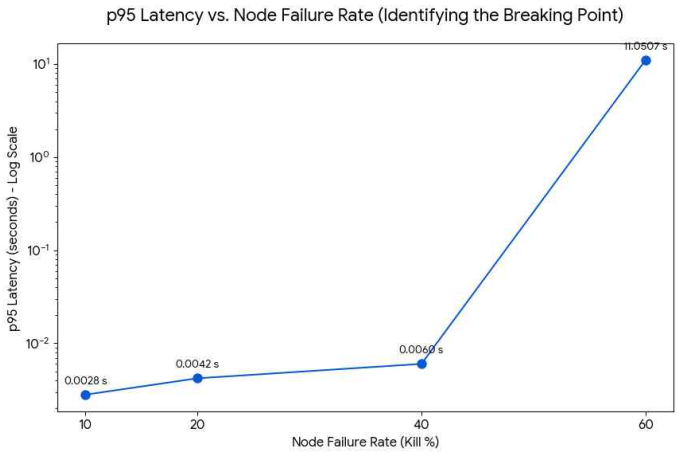
[그림 6] 에이전트 수 증가에 따른 MTTR(p95)의 변화

그림 6는 에이전트 수 증가에 따른 MTTR(p95)의 변화를 나타낸다. X축은 에이전트 수(AGENTS), Y축은 복구 시간(MTTR, 초 단위)이다. 에이전트 수가 50 → 500하고 장애 노드

수(5 -> 50)와 총 작업 수(500 -> 5000)가 10배 증가하는 동안 작업 성공률은 100%로 완벽하게 유지되었으며 장애 복구 시간(p95 —MTTR)은 최대 0.034초로 측정됐다. 이는 제어 트래픽이 중앙 컨트롤러를 경유하지 않고 P2P로 분산되기 때문에, 실질적으로 O(1)에 근접한 (near-constant) 성능을 달성한 결과로 해석된다. 결론적으로, Mutual Cloud의 분산 제어 평면은 중앙집중형 구조의 확장성 병목을 제거하고, 높은 견고성과 자율 복원력(Self-Healing Resilience) 을 실험적으로 입증하였다.

3.4.3. 장애 내성 평가 (Fault Tolerance Evaluation)

본 실험은 시스템이 견딜 수 있는 최대 장애 수준(Breaking Point)을 확인하기 위해 수행되었다. 전체 에이전트 수(AGENTS)는 100개로 고정하고, 무작위로 선택된 일정 비율의 노드를 강제 종료시켜(KILL_PERCENT = 10, 20, 40, 60%) 구간별 복구 성능을 측정하였다. 각 구간별 실험은 동일한 조건에서 4회 반복 수행되었으며, 주요 지표는 MTTR p95(Mean Time To Recovery, 95th percentile)와 작업 성공률(Success Rate)이다.
모든 구간에서 작업 성공률 100% 를 달성하였으며, 100개의 노드 중 60개가 유실된 상황에서도 단 하나의 작업 손실도 발생하지 않았다. 이는 Mutual Cloud의 장애 감지 및 작업 재시도 메커니즘이 완전하게 작동함을 의미한다.



[그림 7] 장애 비율에 따른 MTTR 변화

Kill % (장애 비율)	N (Agents)	TASKS	Success Rate	MTTR p95 (꼬리 복구 시간)
10%	100	1000	100.0%	0.028 s
20%	100	1000	100.0%	0.042s
40%	100	1000	100.0%	0.0060 s
60%	100	1000	100.0%	11.0507 s

[표 2] 실험 데이터 세트

10~40% 구간에서는 MTTR이 0.028초에서 0.060초로 점진적으로 증가하였으며, 이는 DHT 기반 자율 재할당 메커니즘이 병렬 복구 효율을 안정적으로 유지함을 보여준다. 반면, 60% 구간에서는 MTTR이 11.05초로 급격히 상승하였는데, 이는 복구 실패가 아니라 남은 40개의 노드가 1,000개의 작업을 처리하기에 한계에 도달한 리소스 포화(Resource Saturation) 현상 때문이다. 특히 p95 MTTR(11.05초)은 시스템의 하트비트 만료 시간(TTL = 20초) 보다 짧게 측정되었으며, 이는 중앙 제어기의 TTL 감지에 의존하지 않고 에이전트 주도형 복구(Agent-driven Recovery) 가 즉시 수행되었음을 의미한다.
결과적으로, Mutual Cloud는 전체 노드의 40%가 동시에 장애를 겪는 극한 상황에서도 100%의 작업 성공률과 0.060초 이내의 평균 복구 시간을 유지하였다. 이는 중앙 제어기 없이도 자가 치유(Self-healing) 기반의 분산 복구 메커니즘이 안정적으로 동작함을 실험적으로 입증한 결과이다.

4. 참고문헌

[1] L. Schiff et al., “In-Band Synchronization for Distributed SDN Control Planes,” ACM SIGCOMM, 2016.
[2] M. Westerlund and N. Kratzke, "Towards Distributed Clouds: A Review About the Evolution of Centralized Cloud Computing, Distributed Ledger Technologies, and a Foresight on Unifying Opportunities and Security Implications," Proc. IEEE Int. Conf. on High Performance Computing and Simulation (HPCS), pp. -, Orleans, France, July 2018.
[3] Y. Hassanzadeh-Nazarabadi, S. Taheri-Boshrooyeh, S. Otoum, S. Ucar, and Ö. Özkasap, "DHT-Based Communications Survey: Architectures and Use Cases," arXiv preprint, arXiv:2109.10787, 2021.
[4] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," Proc. 1st Int. Workshop on Peer-to-Peer Systems (IPTPS), Lecture Notes in Computer Science, vol. 2429, pp. 53-65, Oct. 2002.
[5] R. K. Vankayalapati and C. Pandugula, "AI-Powered Self-Healing Cloud Infrastructures: A Paradigm for Autonomous Fault Recovery," Migration Letters, vol. 19, no. 6, pp. 1173-1187, Dec. 2022.
[6] D. E. Sarmiento, A. Lebre, L. Nussbaum, and A. Chari, "Decentralized SDN Control Plane for a Distributed Cloud-Edge Infrastructure: A Survey," IEEE Communications Surveys & Tutorials, vol. 23, no. 1, pp. 256-281, Jan. 2021.
[7] C. M. Colson, M. H. Nehrir, and R. W. Gunderson, "Distributed Multi-Agent Microgrids: A Decentralized Approach to Resilient Power System Self-Healing," Proc. 4th Int. Symp. on Resilient Control Systems (ISRCS), Boise, ID, USA, Aug 2011.