P2P 기반 분산형 클라우드를 이용한 소규모 조직의 유휴 컴퓨팅 자원 공유 시스템

A P2P-based Distributed Cloud System for Sharing Idle Computing Resources in Small-scale Organizations

구르미(9팀) : 금채원(2276029), 이서영(2276218), 송예린(2171023)

1. Team Info

1.1. 과제명

P2P 기반 분산형 클라우드를 이용한 소규모 조직의 유휴 컴퓨팅 자원 공유 시스템
A P2P-based Distributed Cloud System for Sharing Idle Computing Resources in Small-scale Organizations

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.

둘째, 내부 자원 공유를 도입하더라도 전통적 중앙집중형 아키텍처에 운영이 한 지점으로 수

렴하면 단일 장애점(SPOF,Single Point of Failure)에 취약해진다. 스케줄링·메타데이터·데이터 경로가 중앙에 몰리면 네트워크 대역폭과 디스크 I/O가 포화되고 큐 지연이 증가하여 성능이 제한되며, p95/p99 지연(tail latency)이 급증한다. 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, 내부 공유를 도입해도 중앙집중형 구조 탓에 안정성·가용성·성능의 근본 한계(SPOF·병목 현상)를 피하기 어렵다. 따라서 본 과제는 비용 구조(경제성)와 시스템 신뢰성/성능(안정성·확장성)을 함께 개선할 수 있는 분산형 아키텍처의 필요성과 타당성을 규명하는 것을 목표로 한다.

2.2. 기존 연구와의 비교

본 절은 해당 시스템이 (1) 중앙형 클라우드, (2) 프라이빗 클라우드, (3) DAOnetes와 어떻게 다른지 정리한다. 연구의 목표는 참여자 각자의 남는 자원을 안전하고 규칙적인 방식으로 서로 빌려 쓰게 하여 비용을 낮추고(TCO 절감), 장애 시 자동 회수·재배치를 가능하게 하는 것이다. 운영 방식은 상태를 알고(자원 상태 공유) → 맞는 곳에 맡기고(적합 노드 선택) → 끊기면 자동으로 바꾸며(자동 전환/재배치) → 쓴 만큼 근거를 남겨 비용을 정산하는 흐름이다.

2.2.1. 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 벤더의 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.

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.

2.2.2. 프라이빗 클라우드와의 비교

프라이빗 클라우드는 한 조직 내부에서 자산을 직접 보유하고 운영한다. vCenter나 Kubernetes 같은 중앙 컨트롤 플레인이 호스트·노드·워크로드를 일괄 관리하고, 고가용성(예: vSphere HA, K8s 셀프힐링)과 자동화를 통해 내부 품질과 일관성을 높인다. 다만 확장은 물리 자원과 예산(CAPEX)에 좌우되고, 외부 조직과 자원을 정책적으로 공유하기가 어렵다. 유휴 자원은 조직 울타리 안에 묶여 남기 쉬워 Overprovisioning 비용이 누적되며, 정산은 대개 내부 차지백/쇼백으로 처리된다.

제안하는 시스템은 여러 조직의 유휴 자원을 정책 기반으로 연합하는 분산형 모델이다. 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. 장애가 발생하면 자동 회수·재배치로 복구하고, 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.

2.2.3. 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.

	중앙형 클라우드	**프라이빗 클라우드**	DAOnetes	Mutual Cloud
자원 관리	대기업이 하드웨어를 소유 및 관리하고 사용자가 이를 임대하는 구조	한 조직이 서버를 소유하며 사내 권한에 따라 공유	참여자 각자가 소유하며 온체인 워크그룹이 승인·관리	참여자 각자가 소유하며 정책에 따라 안전하게 공유
운영 및 확장성	벤더 스케줄러로 높은 확장성 및 중앙 배정	중앙 관리 시스템에서 한 번에 관리. 확장 시 새 장비 마련으로 시간 및 비용이 큼	온체인 승인 절차를 따라 워크그룹 중심 확장	상태 공유와 네트워크 탐색으로 유휴 노드 자동 연결 분산 주소록 기반으로 요구 사항에 맞는 노드 자동 선택
장애 대응	고비용, 안정적	중앙 위주 자동 회수 및 재배치	토큰 재지정으로 재배치	자동 회수 및 재배치
비용/정산	사용량 임대료	장비 구매 및 공	라이선스/토큰으	사용 기록 기반

1) https://github.com/workbenchapp/daonetes

	(장기 상승·종속 우려)	간 등을 직접 구매 및 내부 정산	로 접근 제어, 설계에 따라 과금방식 상이	후불제로 추가 비용 최소화

2.3. 제안 내용

2.3.1. Mutual Cloud

본 연구에서는 P2P 기반의 프라이빗/커뮤니티 클라우드(Private/Community Cloud) 형태의 자원 공유 시스템인 'Mutual Cloud'를 제안한다. Mutual Cloud는 각 참여 노드가 자신의 유휴 컴퓨팅 자원을 공용 자원 풀에 제공하는 분산형 시스템이다. 중앙 스케줄러에 의한 단일 장애점(SPOF) 및 병목 현상을 배제하기 위해, DHT(Distributed Hash Table)를 활용한 분산 스케줄링 메커니즘을 핵심으로 채택하였다.

시스템 아키텍처는 제어면(Control Plane)과 데이터면(Data Plane)으로 분리된다. 제어면은 노드 상태, 작업 정보 등 경량 메타데이터를 DHT에 기록하고 조회함으로써 분산화된 제어를 수행한다. 데이터 면은 실제 작업 데이터 및 결과물을 노드 간 P2P 방식으로 직접 전송하여 전송 효율성을 극대화한다. 본 시스템의 핵심 목표는 다음과 같다. 첫째, 온프레미스(On-premise) 환경의 유휴 자원의 활용 극대화한다. 둘째, 중앙 제어부 없이 노드의 장애를 시스템 스스로 감지하고 작업을 재할당하는 자가 치유 구조를 확보한다. 셋째, 신뢰할 수 없는 환경에서도 보안과 격리를 보장하고, 작업 제출부터 결과 반환까지 일관성있는 사용자 경험을 제공한다. 시스템의 주요 공유 자원은 CPU 및 GPU와 같은 컴퓨팅 자원이며, RAM과 로컬 스토리지는 작업 수행에 필요한 임시 메모리 및 스크래치 공간으로 활용된다.

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 자가치유(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 '뮤츄얼 클라우드' 시스템은 경제적 효율성 증대와 운영 안정성 확보라는 두 가지 핵심 목표를 달성함으로써, 기존 클라우드 컴퓨팅 인프라의 고질적인 구조적 문제를 해결한다.

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 $\Delta C \approx 40\%$ 감소시키는 것을 목표로 한다. 더 나아가, 기존에 큰 비용으로 구매하였으나 대부분 방치되던 고가 장비의 CPU/GPU 평균 활용률을 업무/비업무 시간대 분리 집계에서 $U0 \rightarrow U1(\Delta U \approx 60\%p)$ 향상시키는 것을 검증한다.

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.

P2P + DHT(Distributed Hash Table) 기반 분산 제어 아키텍처는 중앙 집중형의 구조적 리스크를 근본적으로 해결하며 시스템 안정성 및 가용성을 향상시킨다. 중앙 서버가 존재하지 않기 때문에 특정 노드에 장애가 발생해도 전체 시스템은 중단 없이 정상적으로 동작한다. 본 연구는 노드 장애율 p 환경의 장애 주입 실험에서 서비스 성공률 $\geq 99.99\%$, 가용성 $\geq 99.999\%$ (Five Nines), 평균 복구시간(MTTR) ≤ 2분을 달성하는지를 평가하여 분산 아키텍처의 내결함성을 입증한다.

또한, 노드 수나 작업 요청이 증가하더라도 부하가 중앙 서버에 집중되지 않고 네트워크 전체에 분산되어 시스템 규모가 커져도 성능 저하 없이 안정적인 운영이 가능하다. 규모 확장 측면에서는 노드 수 $N \rightarrow 10N$ 증가 시 DHT 조회 지연의 $O(\log N)$ 특성을 유지하면서 스케줄링 대기시간이 ≤ 5 ms로 억제되고, 처리량이 α배로 증대되는 선형성에 가깝게 동작하는지를 검증한다. 이 모든 데이터 전송 및 작업 처리가 P2P(Peer-to-Peer) 방식으로 이루어져 중앙 서버의 네트워크 및 스토리지 병목 현상을 원천적으로 해결한다.

본 논문은 위에서 제시된 주요 지표(TCO, 활용률, 처리량/지연, 가용성/복구, 확장성 등)를 표준화된 워크로드(배치/추론)와 규모(N,2N,5N,10N)에서 체계적으로 측정하여, 퍼블릭 클라우

드와 비효율적 온프레미스 사이의 양자택일을 넘어 지속 가능한 자체 컴퓨팅 인프라의 실증 가능성을 보여주는 것을 의의로 한다.

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 뮤츄얼 클라우드는 경제적 효율성과 운영 안정성이라는 두 가지 목표를 달성하기 위해 상호 유기적으로 동작하는 네 가지 핵심 기능 모듈로 구성된다. 각 기능은 P2P와 DHT 기반의 탈중앙화 철학을 기반으로 설계되어 중앙 집중형 시스템의 구조적 한계를 근본적으로 해결한다.

2.5.1. 탈중앙화 노드 탐색 및 자원 관리 (Decentralized Node Discovery and Resource Management)

중앙의 관리 서버 없이 네트워크에 참여하는 모든 노드의 상태와 자원 정보를 관리하고, 새로운 노드가 참여하거나 이탈하는 것을 자율적으로 처리한다. 이 기능은 중앙 관리 서버 없이 DHT(Distributed Hash Table)를 통해 네트워크에 참여하는 모든 노드의 상태(유휴/작업 중), 자원 사양(CPU, GPU, 메모리)을 관리하고, 노드의 참여와 이탈을 주기적으로 DHT에 '발행(Publish)'한다. 다른 노드는 작업을 요청하기 위해 DHT에 특정 조건(예: 'GPU를 가진 유휴 노드')을 '질의(Query)'하여 후보 노드 목록을 확보한다. 이를 통해 중앙 관리자의 역할을 네트워크 전체에 분산시킴으로써 SPOF 문제를 원천적으로 해결하고, 노드 수가 증가해도 안정적으로 동작하는 확장성의 기반을 마련한다. 이러한 분산 탐색 방식의 확장성을 증명하기 위해, 노드 수를 100개부터 10,000개까지 점진적으로 늘리는 시뮬레이션 환경을 구축한다. 각 규모에서 임의의 노드가 특정 자원 조건(예:"GPU=True, Status=Idle")을 가진 다른 노드를 탐색하도록 1,000회 요청하고, 평균 조회 지연 시간과 조회 성공률을 측정하여 노드 수(N) 증가에도 O(logN) 특성을 유지하며 안정적으로 동작하는지 검증한다.

2.5.2. 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($\Delta U \approx 60\%p$ 향상)과 총 작업 처리량을 비교 분석한다.

2.5.3. P2P 기반 작업 실행 및 데이터 전송 (P2P-based Task Execution and Data Transfer)

실제 작업과 데이터를 처리하기 위해 작업이 할당되면 요청 노드(Client)와 수행 노드(Agent)는 P2P 방식으로 직접 통신 채널을 설정하여 대용량 데이터(예: 소스 코드, 데이터셋, 컨테이너 이미지)를 전송하고 작업을 실행한다. 이는 중앙 서버의 네트워크 및 스토리지 병목 현상을 근본적으로 해결하여 TCO 절감($\Delta C \approx 40\%$ 감소)에 기여한다.

이러한 P2P 전송 방식의 병목 해소 효과를 정량적으로 입증하기 위해, 중앙 서버 경유 방식과 P2P 직접 전송 방식의 성능을 비교한다. 10GB 크기의 컨테이너 이미지를 각 방식으로 100회 전송하며, 평균 전송 완료 시간과 중앙 관측 지점의 네트워크 트래픽을 측정하여 P2P 방식의 우수성을 보인다.

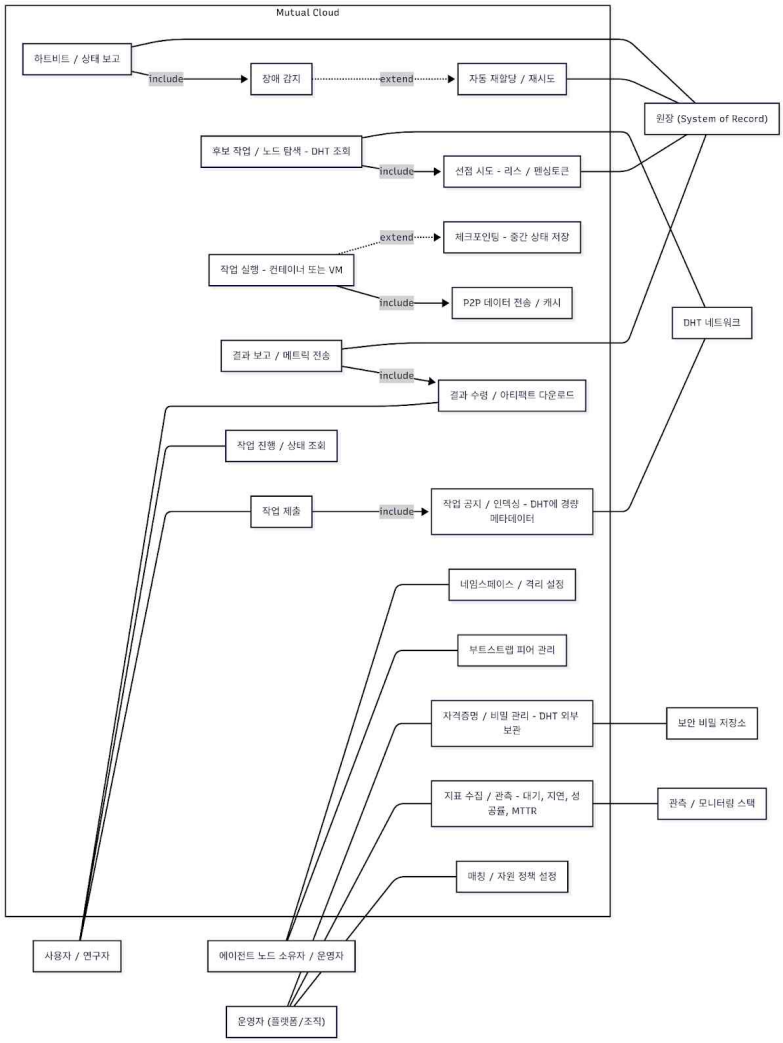
2.5.4. 자동화된 내결함성 및 복구 (Automated Fault Tolerance and Recovery)

본 기능은 작업 수행 중인 노드에 장애가 발생하거나 네트워크 연결이 끊겼을 때, 이를 자동으로 감지하고 작업을 다른 유휴 노드에 재할당하여 서비스 연속성을 보장하는 '안전망' 역할을 한다. 노드 간 주기적인 헬스 체크(Health Check)나 하트비트(Heartbeat) 메시지를 통해 장애를 감하고, 작업 수행 노드의 장애 발생 시 해당 작업을 다른 유휴 노드에 재할당하여 연속성을 보장한다. 작업 수행 노드의 장애가 감지되면, 스케줄러는 해당 작업을 '실패' 상태로 변경하고 즉시 DHT에 다른 유휴 노드를 질의하여 작업을 재시도(Retry)한다. 작업의 중간 상태를 저장하는 체크포인팅(Checkpointing) 기법을 적용하여, 처음부터가 아닌 중단된 지점부터 작업을 재개할 수 있다. 이러한 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.

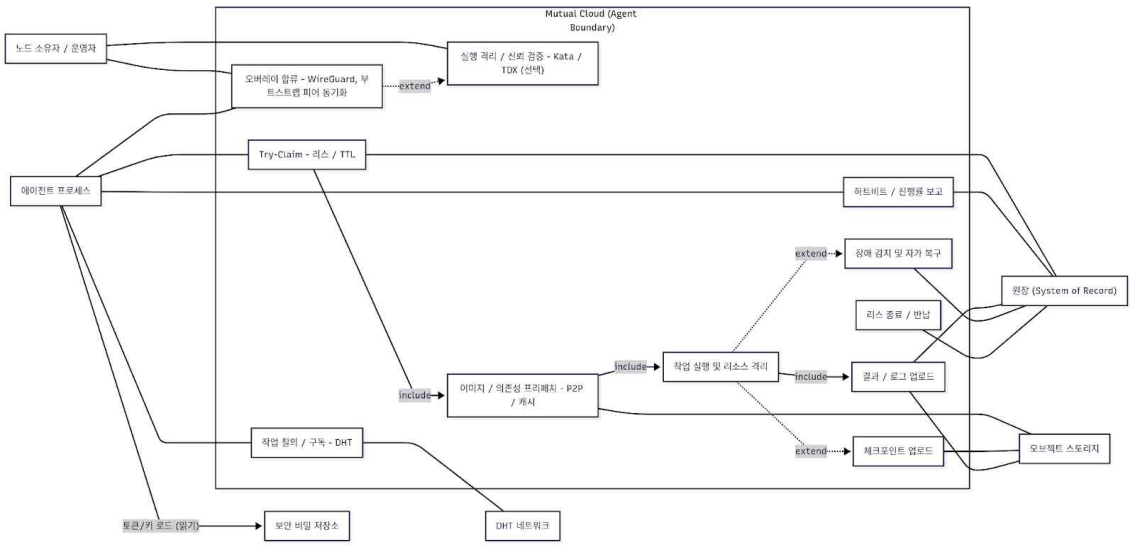
3. Project-Design 과제 설계

3.1. 요구사항 정의

Mutual Cloud는 분산 환경에서 다수의 에이전트 노드가 참여하여 작업 탐색, 매칭 및 선점, P2P 데이터 전송, 내결함성 확보를 수행하는 프레임워크이다. 본 정의서는 상위 수준 유스케이스와 기능별 요구사항을 포함한다. 이를 통해 시스템이 달성해야 할 확장성, 가용성, 효율성, 비용 절감 목표를 구체화하고, 사용자·노드 소유자·운영자 등 이해관계자 간 공통 기준을 제공한다.

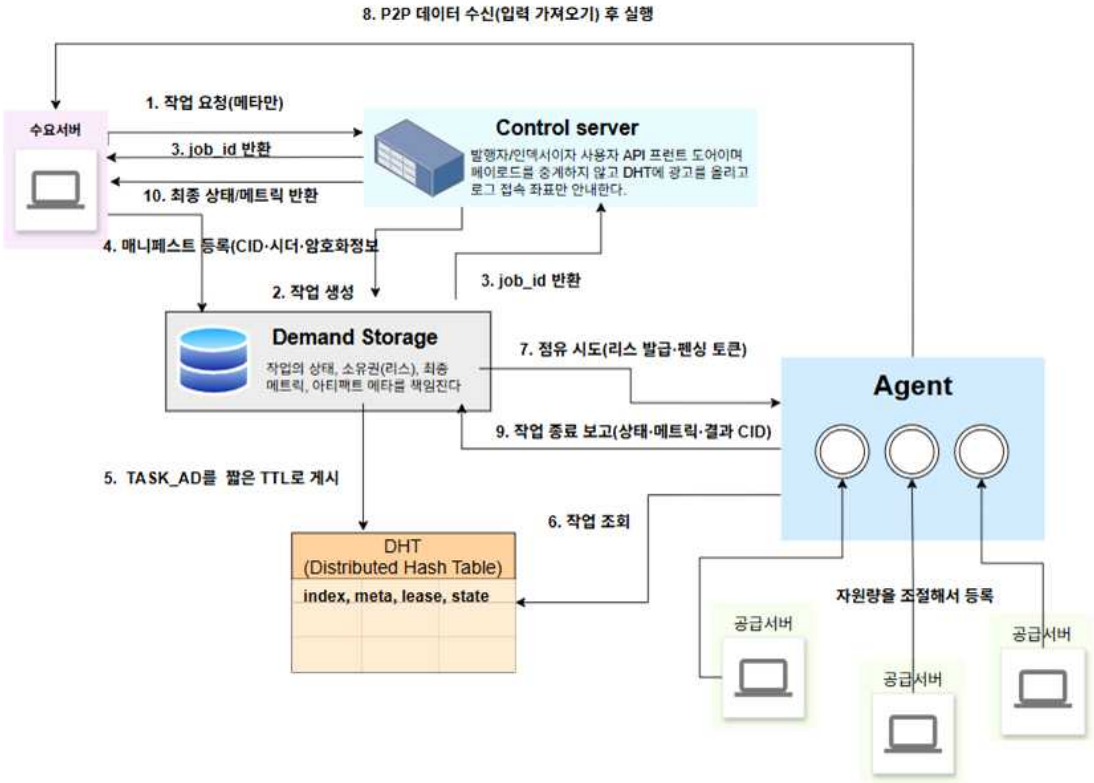


[그림 1] 상위 수준 유스케이스



[그림 2] 에이전트 관점 상세

3.2. 전체 시스템 구성



[그림 3] 전체 시스템 구상도

3.2.1. 시스템 구성요소

• Demand Storage(DB): 작업의 상태, 소유권(리스), 최종 메트릭, 아티팩트 메타를 책임진다.

• 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.

• Agent : DHT로 작업을 발견하고 Demand에서 try-claim(리스+펜싱토큰)을 받아 실행하며 데이터는 Object Storage와 직접 교환한다.

• 실시간 로그: 사용자 <—> 에이전트 직결(WS/gRpc)이고 Control은 에이전트 주소• 토큰을 반환하는 역할만 한다.

3.2.2. 시스템 구성의 흐름

1) 사용자 → Control POST /api/tasks(메타만)

Control은 내부에서 Demand /jobs 생성 후 job_id와 에이전트 공개키·시더 허용정책 등 P2P 지시사항을 사용자에게 그대로 돌려준다.

2) 사용자(Seeder) 준비

입력을 암호화(AES-GCM) 하고 조각화 → Merkle DAG로 manifest_root_cid 생성 후 시딩

시작(libp2p/QUIC·WebRTC)하고 POST /jobs/{id}/manifest {cid, seeders[], enc_meta} 를 Demand에 등록한다.

3) Control/Demand → DHT 광고

TASK_AD{job_id, demand_url, topic, exp, sig} 와 p2p/<job_id>/manifest{cid, seeders, enc} 를 짧은 TTL로 게시한다.

4) 에이전트 발견·클레임

에이전트는 DHT에서 광고를 읽고 Demand의 try-claim 으로 리스+펜싱 토큰을 획득하면 assigned→running으로 전환한다.

5) 에이전트 P2P 수신·실행

DHT/Demand에서 받은 manifest_root_cid·시더 목록으로 클라이언트에서 P2P로 직접 fetch → 복호화 → 컨테이너 실행.

실행 중 컨테이너 stats는 내부 집계, 실시간 로그는 에이전트 로컬 WS로 제공한다.

6) 로그 직결

사용자가 GET /api/tasks/{id}/logs 를 호출하면 Control은 에이전트 WS 엔드포인트+토큰을 조회해 반환. 사용자는 에이전트에 직접 연결(진행률·요약 자원사용량만 노출).

7) 종료 보고(멱등)

에이전트는 POST /jobs/{id}/finish 로 상태(succeeded/failed), 최종 메트릭(cpu_avg, mem_peak…), result_root_cid/artifacts(CID 목록) 를 Demand에 보고한다.

8) 최종 조회

사용자는 GET /api/tasks/{id} 로 최종 상태/메트릭을 조회하고, 결과가 CID로 제공되면 P2P로 다운로드한다.

9) 재큐잉/타임아웃 관리

Control의 requeueLoop 가 assigned/running/timeout 을 감시해 만료·실패 시 정책에 따라 queued 로 되돌리거나 재제안한다.

10) 보존·정리

시더 유지 시간(예: 완료 후 N시간/일) 정책으로 클라이언트 시더는 중단된다.

DHT 레코드는 TTL 만료로 자연 소거되고 Demand DB는 보존기간 후 아카이브/삭제된다.