# 1장 - Optimizing Intra-Node Networking with Live-Migration-Based Colocation

*이 장 사용 시간: 00:25 · 누적 사용 시간: 00:25*

안녕하세요. 서울대학교 컴퓨터공학부 Bernhard Egger 교수님의 지도를 받는 민지홍입니다. 기존 애플리케이션과 연결을 유지하며, 라이브 마이그레이션으로 같은 호스트에 배치된 이점을 통신 성능으로 이어가는 연구를 발표하겠습니다.

# 2장 - Intra-Node TCP vs Shared-Memory IPC

*이 장 사용 시간: 00:55 · 누적 사용 시간: 01:20*

같은 노드의 두 프로세스는 메모리를 공유할 수 있지만, TCP를 거치면 직접적인 IPC보다 비용이 클 수 있습니다.

관련 연구인 SocksDirect의 8바이트 메시지 측정입니다. Linux TCP와 lockless 공유 메모리 큐의 왕복 지연은 각각 11과 0.25마이크로초, 단일 코어 메시지 처리율은 초당 약 90만 회와 2,700만 회입니다.

저는 이러한 기존 가속 연구를 바탕으로, 애플리케이션 호환성을 유지하며 남은 통신 비용을 더 줄여 공유 메모리 IPC에 최대한 가까운 경로를 만들고자 했습니다.

# 3장 - Colocated Apps Still Use the Network Stack

*이 장 사용 시간: 00:40 · 누적 사용 시간: 02:00*

처음부터, 또는 마이그레이션 후 같은 호스트에 배치된 두 VM을 보겠습니다. 소켓은 여전히 각 게스트의 TCP/IP 스택과 호스트 전달 경로를 거칩니다.

배치만 가까워져서는 네트워크 처리 비용이 사라지지 않습니다. 공유 메모리의 이점을 얻으려면 애플리케이션이 보는 동작을 유지하며 전송 경로까지 바꿔야 합니다.

# 4장 - Goal of this Thesis

*이 장 사용 시간: 01:00 · 누적 사용 시간: 03:00*

목표는 같은 호스트의 애플리케이션 통신을 투명하게 공유 메모리로 전환하고, 라이브 마이그레이션 중에도 기존 연결을 유지하는 것입니다. 배치가 바뀔 때 연결을 끊어야 한다면, 가속을 위해 애플리케이션까지 수정해야 하기 때문입니다.

왼쪽은 서로 다른 호스트에서 TCP로 통신하는 상태입니다. 함께 배치되면 기존 연결을 공유 메모리로 바꾸는 promotion을, 다시 떨어지기 전에는 연결 상태를 보존하며 네트워크로 돌아가는 demotion을 수행합니다.

같은 논리적 연결을 유지하는 이 원리를 VM, 컨테이너, 호스트 프로세스 쌍에 공통으로 적용하고자 합니다.

# 5장 - Android Automotive Example

*이 장 사용 시간: 01:35 · 누적 사용 시간: 04:35*

Android 공식 문서의 차량 화면 예시입니다. VM A는 지도·음악 재생 같은 일반 화면을, VM B는 안전 관련 표시와 차량 상태에 맞는 표시 여부 확인을 담당합니다.

두 화면은 겹쳐 하나의 디스플레이로 출력합니다. 위쪽 VM B의 화면은 평소 일반 화면 영역을 투명하게 두고, VM A가 응답하지 않으면 대체 표시로 덮습니다. 일반 화면에 문제가 생겨도 안전 표시는 유지됩니다.

두 VM은 정상 동작을 확인하고 화면 구성을 맞추려고 gRPC로 계속 통신합니다. 이 역할 분리를 유지하며 반복적인 노드 내부 통신 비용을 줄이고자 합니다. CPU와 전력이 제한된 차량 같은 임베디드 환경에서는 프로토콜 처리 비용을 줄여 다른 작업에 쓸 자원을 확보하는 것도 중요합니다.

왼쪽은 네트워크 스택 경로, 오른쪽은 연결별 공유 메모리로 대체하는 제안입니다. VM 경계는 유지하며 L4 TCP와 그 아래 IP 처리를 우회하고, 상위 HTTP/2와 gRPC는 유지합니다.

# 6장 - Traffic Within Cloud Infrastructure

*이 장 사용 시간: 01:05 · 누적 사용 시간: 05:40*

클라우드에서는 독립적으로 관리되는 서비스도 많이 통신하므로, 실행 환경을 합치지 않고 locality를 높여야 합니다.

왼쪽은 데이터센터 트래픽의 약 71퍼센트가 같은 데이터센터에 머물 것이라는 Cisco 전망입니다. 오른쪽은 내부 고객 VM 트래픽의 10에서 40퍼센트가 서로 다른 tenant 사이에서 발생했다는 측정입니다. 기준은 다르지만, 내부 통신과 독립된 서비스 간 통신의 중요성을 보여줍니다.

한 고객이 다른 고객의 서비스를 자기 VM에 합칠 수는 없지만, 인프라는 각 실행 환경을 유지한 채 같은 호스트에 배치할 수 있습니다. USERNET은 이 locality를 실제 통신 성능으로 연결합니다.

# 7장 - Intra-Cloud Colocation Example

*이 장 사용 시간: 01:15 · 누적 사용 시간: 06:55*

값비싼 GPU 같은 물리 하드웨어를 VFIO로 정적 할당받은 서버 VM 하나를 유지하는 상황입니다. 이 VM이 NVIDIA Triton 추론 서비스를 제공하면, 같은 고객이나 여러 고객의 VM 서비스가 HTTP·gRPC 요청으로 GPU 자원을 함께 활용할 수 있습니다.

각 VM에 GPU를 따로 할당하지 않고 한 서버 VM의 자원을 네트워크 서비스로 공유합니다. GPU 서버는 그대로 두고 클라이언트 VM을 같은 호스트로 옮겨 locality를 높이고자 합니다.

오른쪽에서는 분리된 두 VM이 같은 호스트에서 공유 메모리로 데이터를 전달합니다. GPU 서비스는 제자리에 남습니다. 서비스 동작 중 배치를 바꾸므로, 기존 HTTP나 gRPC 연결도 유지해야 합니다.

# 8장 - Requirements for Transparent Communication

*이 장 사용 시간: 01:05 · 누적 사용 시간: 08:00*

요구사항은 애플리케이션 호환성, 도입 비용, 장기 운영 비용으로 구분합니다.

첫째, 투명성입니다. 기존 프로그램을 그대로 쓰도록 바이너리·소켓 동작·연결을 유지해야 합니다.

둘째, 처음 도입할 때의 변경 범위입니다.
게스트·호스트·하이퍼바이저의 변경을 줄여 기존 환경에 쉽게 적용하고자 합니다.

셋째, 도입 후 유지 비용입니다. 많은 개발자가 유지보수하는 운영체제·네트워크의 표준 기능을 최대한 활용하고, 최적화할 부분만 제한적으로 수정하고자 합니다.
직접 관리할 구현은 줄이면서 표준 기능의 개선과 수정은 계속 활용하는 것입니다.

즉, 둘째는 설치·적용, 셋째는 이후 업데이트·운영의 부담에 초점을 둡니다.

# 9장 - Related Work: Five Transport Families

*이 장 사용 시간: 01:45 · 누적 사용 시간: 09:45*

추가로 줄일 비용을 찾기 위해 공유 메모리 IPC를 기준으로 기존 기법의 남은 처리를 비교하고, 다섯 유형으로 묶었습니다. 차례로 설명드리겠습니다.

Kernel TCP는 전달 경로를 개선하되 커널 TCP 처리는 유지합니다.

Userspace TCP는 프로토콜 처리를 사용자 공간으로 옮기지만 패킷 처리는 남습니다.

Kernel IPC는 로컬 소켓이나 공유 버퍼로 TCP를 우회합니다.

Userspace SHM은 같은 커널의 서로 다른 프로세스가 사용자 공간에서 공유 메모리로 직접 통신합니다.

RDMA는 등록된 버퍼와 하드웨어 지원으로 전송을 가속합니다.

표의 마지막 열은 연결을 유지하며 경로를 바꾸는 Transport Hot Swapping입니다. 로컬 경로가 빨라도 배치 변경 시 연결이 끊기거나, 가속 경로로 전환할 수 없거나, 로드 밸런싱 등을 위해 다시 분리할 수 없다면 모든 워크로드에 적용하기 어렵습니다.

USERNET은 사용자 공간의 직접적인 바이트 큐를 더 최적화하고, 경로 전환을 inter-VM, inter-container, inter-process 모두에 결합한 것이 차별점입니다.

# 10장 - Contributions and Publications

*이 장 사용 시간: 01:05 · 누적 사용 시간: 10:50*

기여는 크게 두 부분입니다.

첫째는 GECON 2026에 발표된 OverlaySys입니다. 프로그램 수정 없이 통신 처리를 바꾸도록 시스템콜과 시그널을 가로채는 기반을 제공합니다. 기존에도 이런 연구가 있었지만, USERNET 개발에 필요한 userspace interception framework의 한계를 더 극복하도록 개선했습니다. 뒤에서 설명드리겠습니다.

둘째는 심사 중인 USERNET입니다. 직접적인 공유 메모리 소켓에 사용자 공간의 대기와 readiness 처리를 결합했습니다.

Transport Hot Swapping으로 배치가 바뀌어도 연결을 유지합니다. 약 30개 시스템과 구성을 비교해 어떤 기존 방식보다 무엇이 나아지는지 확인한 평가도 기여입니다.

# 11장 - Architecture Overview

*이 장 사용 시간: 01:35 · 누적 사용 시간: 12:25*

공통 구조와 송수신 과정을 USERNET 아키텍처부터 설명드리겠습니다.

배치가 바뀌어도 애플리케이션의 연결을 유지하도록, 논리적 소켓이 연결 상태와 전송 경로를 분리합니다. 위쪽 애플리케이션은 기존 소켓 인터페이스를 사용하고, OverlaySys가 지원 대상 연산을 USERNET에 전달합니다. 논리적 소켓은 연결 상태와 데이터 순서를 관리하며 경로를 선택합니다.

같은 노드에서 가속 조건을 만족하면 오른쪽 공유 큐를, 그 외에는 시스템 TCP를 사용합니다. VM 쌍은 IVSHMEM, 같은 커널의 컨테이너 쌍이나 호스트 프로세스 쌍은 memfd를 사용하며, 공유 방식이 달라도 소켓 처리 원리는 같습니다.

데몬은 데이터를 전달하지 않고 상대 확인, 자원 준비, 경로 전환을 조율합니다. 아래쪽 대기·준비 상태 처리도 논리적 소켓에 속합니다. 뒤에서 설명할 hybrid polling은 긴 대기에서 UDP와 유닉스 도메인 소켓을 blocking 인터페이스로 사용해, 계속 폴링하며 CPU를 점유하는 대신 알림까지 잠듭니다.

# 12장 - Send/Recv Walk-Through

*이 장 사용 시간: 01:20 · 누적 사용 시간: 13:45*

같은 노드에서는 TCP/IP 패킷 처리 대신 연결별 바이트 큐로 송수신할 수 있습니다. 공유 메모리로 전환된 연결의 한 방향을 보겠습니다.

왼쪽 애플리케이션이 send를 호출하면 USERNET은 상대의 읽기 위치를 검증하고 빈 공간을 확인한 뒤, 데이터를 공유 큐에 복사하고 쓰기 위치를 공개합니다. 오른쪽 수신자는 갱신된 상태를 확인해 애플리케이션 버퍼로 복사하고 읽기 위치를 갱신합니다. 그러면 송신자가 그 공간을 재사용할 수 있습니다.

이 경로의 TCP/IP 처리는 없애지만, 큐에 넣고 꺼내는 복사는 남습니다. 큐의 범위와 위치 갱신은 사용 전에 검증합니다. 아래 점선은 대기 중인 상대를 깨우는 알림으로, 이미 큐를 확인하는 상대에게는 필요 없습니다. 반대 방향도 별도 큐로 동일하게 동작합니다.

# 13장 - USERNET Interception with OverlaySys

*이 장 사용 시간: 01:15 · 누적 사용 시간: 15:00*

입출력을 유저스페이스로 옮겨도 시그널 핸들러의 소켓 재호출은 동작해야 합니다. 그림은 이때의 교착 문제입니다.

왼쪽 read가 소켓 잠금을 잡은 동안 비동기 시그널이 도착하고, 핸들러도 같은 소켓에서 read를 호출하면 그 잠금을 기다립니다. 원래 read는 핸들러가 반환해야 잠금을 풀 수 있어 서로 기다립니다.

USERNET은 이를 막으려고 보호 연산 중 지연 가능한 핸들러를 미루고, OverlaySys로 시그널 정보와 전달 마스크를 저장합니다. 원래 입출력이 완료되거나 중단 처리로 잠금과 자원 참조를 해제한 안전한 지점에서 시그널을 다시 전달합니다. 핸들러의 read도 이제 잠금을 잡을 수 있습니다. 시그널 정보를 유지하면서 안전한 실행 시점을 보장합니다.

# 14장 - Hybrid Polling

*이 장 사용 시간: 01:30 · 누적 사용 시간: 16:30*

Linux block I/O의 hybrid polling에서 영감을 얻어 대기 순서를 반대로 적용했습니다.

Linux는 과거 I/O 처리 시간에 따라 먼저 잠들고, 타이머 만료 후 폴링으로 완료를 확인합니다. 장치 작업 중 CPU 낭비를 줄이면서 예상 완료 시점에 직접 확인해 빠르게 반응하려는 전략입니다.

USERNET은 짧은 공유 메모리 통신에서 잠들고 깨는 비용을 피하려고 먼저 폴링합니다. 데이터나 공간이 준비됐으면 즉시 진행하고, 아니면 기본값으로 최대 100마이크로초 확인합니다. 대기가 길면 UDP나 유닉스 도메인 소켓에서 blocking해 CPU를 다른 작업에 돌려줍니다.

알림을 놓치지 않도록 대기 등록 후 재확인하고 잠들며, spurious (nested) wakeup에 대비해 깨어나서도 큐를 확인합니다. Inter-VM에서는 UDP 알림 유실에 대비해 한 번의 대기를 기본 최대 1초로 제한하고 재확인하되, 애플리케이션의 원래 시간 제한은 유지합니다.

# 15장 - I/O Multiplexing

*이 장 사용 시간: 01:25 · 누적 사용 시간: 17:55*

이벤트 루프를 수정하지 않으려면 USERNET 연결과 일반 소켓·파이프 등 파일 디스크립터를 함께 기다리면서 어느 쪽도 놓치지 않아야 합니다.

USERNET은 select, poll, epoll과 시그널 마스크 변형을 지원하며, 공유 큐의 준비 상태와 커널 이벤트를 합쳐 애플리케이션에 전달합니다. 앞 장처럼 상태 확인, 대기 등록, 재확인을 거쳐 필요하면 커널에서 기다립니다. 알림은 내부에서 처리하고 애플리케이션에는 실제 소켓 이벤트만 반환합니다.

USERNET 큐가 계속 준비돼 있어도 다른 파일 디스크립터를 확인해야 합니다. 기본값은 유저스페이스 확인 예산 100마이크로초, 커널 재확인 간격 1밀리초입니다. 매 호출마다 다시 세지 않고 호출 사이에 유지되는 글로벌 데드라인을 사용하며, epoll은 epfd별로 관리합니다. 따라서 즉시 반환이 반복돼도 시한이 밀리지 않고, 도달하면 커널의 다른 이벤트도 확인합니다.

# 16장 - Transport Hot Swapping - Promotion

*이 장 사용 시간: 01:25 · 누적 사용 시간: 19:20*

Promotion은 통신 중인 TCP 연결을 공유 메모리로 바꾸는 과정입니다. 새 경로의 데이터가 TCP에 남은 데이터를 앞지르지 않도록 바이트 순서를 지켜야 합니다.

왼쪽 준비 단계에서는 새 쓰기를 잠시 제한하고 진행 중인 쓰기를 정리한 뒤 큐를 준비합니다. 기존 TCP 쓰기도 처리하고 양쪽 경로를 검증합니다. 이어서 양 끝이 paired commit으로 전환을 함께 확정합니다. 합의 후 공유 메모리를 사용하되, 수신 측은 남은 TCP 데이터를 먼저 읽어 이전·새 경로의 순서를 유지합니다.

준비 중 복구 가능한 실패라면 기존 TCP를 유지하고, 일관되게 전환할 수 없으면 연결을 실패 처리해 대기 연산을 깨웁니다. 동기·비동기 promotion 모두 지원하며, 애플리케이션의 논리적 연결은 유지합니다.

# 17장 - Transport Hot Swapping - Demotion

*이 장 사용 시간: 01:40 · 누적 사용 시간: 21:00*

공유 메모리를 쓰는 두 대상이 다른 노드로 분리될 때, 아직 읽지 않은 데이터를 남겨두고 떠나면 안 됩니다. Demotion은 이 데이터를 보존하며 호스트 공유 메모리 의존성을 제거하는 과정으로, 이동 전 준비와 이동 후 복구로 나뉩니다.

첫 단계에서는 입출력을 잠시 멈추고 큐를 고정해 각 송신자가 상대가 아직 읽지 않은 데이터를 보존합니다. 양쪽이 복구 상태를 확정하고 매핑과 필요한 제어 연결을 정리하면, 데몬이 마이그레이션 준비 완료를 알립니다. 데이터를 보존했으므로 애플리케이션의 소비나 이동 후 네트워크 복구를 기다리지 않고 VM을 이동할 수 있습니다.

실행 상태가 복원되면 두 번째 단계에서 상대를 인증하고 프레임 기반 TCP를 만듭니다. 바이트 위치를 확인하며 보존 데이터를 먼저, 새 데이터를 이어 전달합니다. 이후 조건이 맞으면 일반 TCP로 전환할 수 있습니다. 이동 전 안전한 분리와 이동 후 동일한 스트림 복구를 통해 애플리케이션의 소켓 재연결을 피합니다.

# 18장 - Migration Timeline Example

*이 장 사용 시간: 01:05 · 누적 사용 시간: 22:05*

서버는 NODE1에 남고 클라이언트만 이동하며, 도착 후 promotion, 출발 전 demotion을 수행합니다.

처음 NODE2에서는 TCP로 통신합니다. NODE1에 도착하면 기존 연결을 공유 메모리로 전환해 머무는 동안 공유 큐를 사용합니다. 다시 떠나기 전에는 아직 읽지 않은 데이터를 보존하고 매핑을 해제합니다. 준비 후 이동하고, NODE2에서 실행 상태가 복원되면 프레임 기반 TCP로 통신을 잇습니다.

전환 중 잠시 멈출 수 있지만, 애플리케이션의 파일 디스크립터와 논리적 연결은 유지됩니다. 연결이나 스트림을 재구성하지 않고도 배치 변화에 대응할 수 있습니다.

# 19장 - Evaluation Environment

*이 장 사용 시간: 01:00 · 누적 사용 시간: 23:05*

동일 자원에서 통신 성능과 위치 변경 시의 동작을 나눠 평가했습니다.

두 노드는 100기가비트 이더넷으로 연결하고 Ubuntu 26.04와 Linux 7.0을 사용했습니다. 실험 제어와 VM 이미지 저장은 별도 관리·스토리지망의 NAS1이 담당합니다.

각 엔드포인트에 물리 코어 여섯 개, VM·컨테이너에 메모리 8 GiB를 할당하고 비교군에도 같은 한도를 적용했습니다.

정적 실험은 양쪽을 NODE1에 놓았습니다. 마이그레이션은 서버를 NODE1에 둔 채 클라이언트만 NODE2에서 NODE1으로 옮겼다가 돌려보냈습니다.

# 20장 - Selecting Comparable Systems

*이 장 사용 시간: 01:40 · 누적 사용 시간: 24:45*

많은 후보에서 성능과 마이그레이션 지원을 기준으로 비교군을 선정했습니다.

약 30개 시스템을 공통 Ubuntu 환경에서 Netperf 스트리밍·요청응답, 스레드 하나, 연결 하나로 평가했습니다. 지원하는 VM·컨테이너·프로세스 환경은 따로 비교했습니다.

각 환경·지표별로 전체 결과, 검증된 pre-copy, 검증된 post-copy 후보군의 최고 결과를 뽑았습니다. 마이그레이션 후보는 연결을 유지하며 colocation과 분리 시 가속 경로를 전환해야 합니다.

중복 선정은 한 번만 포함하고, 공유 메모리나 하이브리드 대기 등 USERNET과 구조가 비슷한 시스템을 추가했습니다.

유지보수를 연구 동기로 꼽았듯, 기존 구현 상당수는 최신 환경에서 수정 없이 동작하지 않았습니다. 공정한 비교를 위해 같은 Ubuntu 26.04에서 플랫폼 이식과 정확성 수정만 허용하고 전송·동기화 구조는 유지했습니다. 벤치마크도 Netperf와 NetPIPE의 소켓 처리만 수정하고 워크로드와 측정 방식은 유지했습니다.

이후 회색 막대는 해당 환경·작업의 최고 비교군이므로 그래프마다 시스템이 다를 수 있습니다.

# 21장 - VM/Container: ~2× STREAM and 4–6× RR

*이 장 사용 시간: 01:50 · 누적 사용 시간: 26:35*

단일 연결에서 최고 비교군 대비 개선폭을 확인했습니다. 세로축은 환경별 최고 비교군을 1로 둔 상대 성능으로, 높을수록 좋습니다. 왼쪽은 스트리밍 처리량, 오른쪽은 요청응답률 비율이며, 회색이 비교군, 초록색이 USERNET입니다.

스트리밍은 VM·컨테이너 약 두 배, 로컬 프로세스 약 64퍼센트 향상됐습니다. 요청응답은 VM 약 5.6배, 컨테이너 약 4.2배이며, VM 간 절대 성능은 초당 127.5기가비트와 168만 트랜잭션입니다.

캐시 공유 효과를 최대한 줄이려고 서버·클라이언트를 다른 L3 그룹에 배치했지만, Junction의 로컬 구성에는 이를 그대로 적용할 수 없었습니다. Junction은 하나의 Linux 프로세스에서 애플리케이션을 uProc이라는 사용자 공간 실행 단위로 관리합니다. 측정 구성은 두 uProc을 같은 런타임 스레드·코어에서 실행해 L3를 공유하며, 원래 비교에서 USERNET은 그 요청응답률의 0.76배였습니다.

캐시 공유 조건을 더 공정하게 맞추려고 USERNET을 같은 L3 그룹의 서로 다른 코어에 배치하자 1.46배였습니다. 연한 막대가 이 별도 실험입니다.

# 22장 - Sub-Microsecond Latency and Over 130 Gbit/s

*이 장 사용 시간: 01:05 · 누적 사용 시간: 27:40*

NetPIPE로 작은 메시지의 지연과 큰 메시지의 처리량 개선을 확인했습니다.

가로축은 세 환경입니다. 왼쪽 세로축은 1바이트 왕복 시간의 절반으로, 마이크로초 단위이며 낮을수록 좋습니다. USERNET은 약 0.3마이크로초로 VM·컨테이너 비교군보다 낮지만, 로컬에서는 Junction이 더 낮습니다.

오른쪽 세로축은 8 MiB 처리량으로, 초당 기가비트 단위이며 높을수록 좋습니다. USERNET은 약 136에서 151기가비트로 세 환경의 최고 비교군을 앞서, 짧은 교환과 대량 전송의 효과를 확인했습니다.

# 23장 - Bulk Throughput Rises, Then Reaches a Plateau

*이 장 사용 시간: 00:50 · 누적 사용 시간: 28:30*

동시 실행 수에 따른 대량 전송 확장성을 확인했습니다. 가로축은 클라이언트 worker 수, 세로축은 총처리량으로 초당 기가비트 단위이며 높을수록 좋습니다. worker당 연결 네 개와 128 KiB 블록을 사용했습니다.

USERNET은 하나에서 두 worker로 늘릴 때 크게 증가한 뒤 대체로 비슷하게 유지됩니다. 네 worker에서 VM 약 170, 프로세스 약 200기가비트로, worker 수에 비례해 계속 증가하지는 않습니다.

# 24장 - Request Rate Reaches 4 Million Transactions/s

*이 장 사용 시간: 00:50 · 누적 사용 시간: 29:20*

작은 요청응답의 동시 처리 성능도 확인했습니다. 가로축은 클라이언트 worker 수, 세로축은 초당 백만 트랜잭션 단위의 요청응답률로 높을수록 좋습니다. 요청·응답은 각각 1바이트, 연결은 worker당 네 개입니다.

네 worker에서 USERNET은 세 환경 모두 초당 약 380만에서 400만 건으로 비교군을 앞섰습니다. 프로세스의 한 worker에서는 Junction이 앞서지만, USERNET은 동시 실행 수와 함께 처리량이 크게 증가합니다.

# 25장 - Memcached Reaches 3.5–3.7 Million Operations/s

*이 장 사용 시간: 00:50 · 누적 사용 시간: 30:10*

실제 키·값 서비스의 개선도 확인했습니다. Memcached가 1 KiB 값을 제공하고 memaslap이 요청합니다. 가로축은 클라이언트 worker 수, 세로축은 초당 백만 연산 단위의 처리량으로 높을수록 좋으며, worker당 연결은 네 개입니다.

네 worker에서 USERNET은 초당 약 350만에서 370만 연산으로, VM·컨테이너의 VMA와 프로세스의 TAS를 앞섰습니다. 기존 소켓 기반 서비스에도 개선이 이어졌습니다.

# 26장 - nginx Serves More Requests with USERNET

*이 장 사용 시간: 01:20 · 누적 사용 시간: 31:30*

nginx와 wrk로 기존 HTTP 인터페이스에서의 개선을 확인했습니다. 가로축은 wrk 클라이언트 스레드 수, 세로축은 초당 천 건 단위의 요청 처리량으로 높을수록 좋습니다. 스레드당 네 연결을 keep-alive로 재사용하며, 표시한 조건에서 USERNET이 비교군보다 높습니다.

컨테이너·프로세스의 두 스레드 지점 하락은 원본 로그로 확인했습니다. nginx worker는 항상 네 개이며 모두 실행 중이지만, 두 스레드 조건의 중앙값 실행에서는 세 개만 연결을 받아 데이터를 처리했습니다. Keep-alive가 이 배정을 유지해 서버의 실제 병렬성이 낮아졌습니다.

공정한 비교를 위해 서버 worker 수와 CPU 할당은 동일하게 유지하고 연결 분배에 별도 처리를 하지 않았습니다. 클라이언트 두 스레드 자체가 느리다는 뜻은 아닙니다.

# 27장 - Connections Survive VM Post-Copy Migration

*이 장 사용 시간: 01:45 · 누적 사용 시간: 33:15*

위치가 바뀌어도 기존 연결을 유지하며 공유 메모리 가속을 적용할 수 있는지 확인했습니다.

가로축은 시작 후 경과 시간으로 초 단위, 세로축은 iperf3 여섯 연결의 총 수신 처리량으로 초당 기가비트 단위입니다. 연결을 유지한 채 클라이언트를 서버 호스트로 옮겼다가 분리합니다.

Colocated에서 초록색 USERNET은 약 160기가비트, 회색 비교군은 약 90기가비트이며, 분리 후 기존 연결로 원격 경로를 복구합니다.

분리 중에는 통신이 정지합니다. 공유 메모리 분리 준비, VM 실행 환경 복원, 전송 경로 복구가 순차적으로 포함되므로 정지 구간 전체가 VM 메모리 전송 시간은 아닙니다.

두 곡선의 원격 TCP는 단일 연결 intra-node Netperf 처리량으로 선정한 OVS-DPDK switchdev VF입니다. 하지만 inter-node 여섯 연결 VM 실험에서는 잘 확장되지 않아 원격 처리량이 낮습니다.

전환 중 정지는 있어도 애플리케이션 재연결 없이, colocated 상태에서 더 빠른 경로를 사용할 수 있습니다.

# 28장 - Connections Survive Container/Process Migration

*이 장 사용 시간: 01:15 · 누적 사용 시간: 34:30*

같은 연결 유지·경로 전환을 컨테이너와 프로세스에서도 CRIU lazy migration으로 확인했습니다. 가로축은 경과 시간으로 초 단위, 세로축은 iperf3 여섯 연결의 총 수신 처리량으로 초당 기가비트 단위입니다.

초록색 실선은 USERNET 컨테이너, 점선은 USERNET 프로세스, 회색은 컨테이너 비교군 Linux ipvlan입니다. USERNET은 원격에서 공유 메모리로 전환하고 분리 후 네트워크를 복구하면서 원래 연결을 유지합니다.

두 원격 경로에는 앞의 VM 실험 같은 낮은 처리량 문제가 없고, 그 빠른 경로보다도 colocation 후 처리량이 높습니다. 기반 기능은 달라도 연결 유지·경로 전환 원리는 같습니다.

# 29장 - Resource Usage Comparison

*이 장 사용 시간: 01:40 · 누적 사용 시간: 36:10*

성능과 함께 엔드포인트의 CPU·전력 비용을 비교했습니다. 네 그래프의 가로축은 VM·컨테이너·프로세스입니다. 회색 Remote는 NODE1과 NODE2를 합산하고, 초록색 Colocated는 양쪽 엔드포인트가 실행되는 NODE1만 집계합니다.

왼쪽 위 세로축은 같은 자원 측정 실행의 처리량으로, 초당 기가비트 단위이며 높을수록 좋습니다. 오른쪽 위는 기가비트당 누적 CPU 시간으로, CPU-초/Gbit 단위이며 낮을수록 좋습니다. 전송량으로 나눠 호스트 수가 달라도 비교할 수 있습니다. 아래 세로축은 각각 CPU 패키지 전력과 시스템 소비 전력으로, 와트 단위이며 낮을수록 좋습니다.

전송량당 CPU 비용은 VM에서 크게 줄고 컨테이너·프로세스에서는 대체로 비슷합니다. VM 시스템 전력은 약 310에서 173와트로 줄었습니다. 비워진 NODE2를 집계에서 제외한 결과로, 실험 중 실제로 전원을 끈 것은 아닙니다.

# 30장 - Connection Lifetime Must Amortize Setup

*이 장 사용 시간: 01:35 · 누적 사용 시간: 37:45*

빠른 통신의 이득과 연결 초기화 비용을 함께 봐야 합니다. VM 간 connect는 공유 메모리의 동기 초기 설정·promotion을 포함해 약 21.5밀리초이고, OVS-DPDK switchdev VF는 약 43마이크로초입니다. 연결당 일회성 비용이므로 HTTP keep-alive로 재사용하면 여러 통신에 분산할 수 있습니다.

오른쪽은 추가 비용 회수에 필요한 통신량의 모델입니다. 같은 OVS-DPDK 대비 추가 연결 비용 약 21.42밀리초를, 1바이트 Netperf 요청응답률에서 계산한 왕복당 절감 시간 약 29마이크로초로 나누면 약 740회입니다.

별도 connect 측정과 정상 상태 성능을 결합한 계산으로, 실제 애플리케이션의 손익분기점을 측정한 것은 아니며 마이그레이션·close 비용도 제외했습니다.

따라서 연결 수명이 중요합니다. 짧은 연결은 초기 설정·마이그레이션 비용만큼의 통신 이득을 얻지 못할 수 있습니다.

# 31장 - Limitations

*이 장 사용 시간: 00:50 · 누적 사용 시간: 38:35*

통신 경로 제공과 사용 시점 결정은 별개입니다. USERNET은 전송·제어 메커니즘을 제공하지만, 워크로드 배치·연결의 남은 수명·마이그레이션 이득을 자동으로 예측하지는 않습니다.

io_uring 연동, 서로 다른 종류의 실행 환경 간 공유 메모리 통신, 중첩 가상화는 추가로 확장할 부분입니다.

데몬의 연결 승인과 자원 할당은 신뢰합니다. 공유 큐 메타데이터는 검증하지만 별도의 payload 암호화나 부채널 방어는 제공하지 않습니다. 이 범위에서 효과를 평가했습니다.

# 32장 - Future Work

*이 장 사용 시간: 01:00 · 누적 사용 시간: 39:35*

향후 목표는 지원 프로그램을 늘리고 상황별 비용을 줄이는 것입니다. 화살표는 각 연구가 바꿀 구성 요소를 가리킵니다.

Interception은 dlopen처럼 실행 중 불러오는 라이브러리까지 확장하고자 합니다.

대기 시간과 CPU 여유가 작업마다 다르므로, polling 예산을 부하에 맞게 조정하고 I/O multiplexing의 하이브리드 처리 전략도 확장할 수 있습니다.

원격 통신을 가속하도록 inter-node 기법과 결합하고자 합니다. 연결 설정 시 반복 작업을 줄이기 위한 공유 메모리 자원 재사용과 pooling도 평가할 계획입니다.

# 33장 - Summary

*이 장 사용 시간: 01:15 · 누적 사용 시간: 40:50*

정리하겠습니다. 같은 노드에 배치된 이점을 통신 성능으로 이어가려면, 애플리케이션 연결을 유지하며 전송 경로를 바꿔야 합니다.

USERNET은 직접적인 공유 메모리 통신과 Transport Hot Swapping을 VM·컨테이너· 프로세스 환경에 결합합니다. VM·컨테이너 Netperf 평가에서는 지표별 최고 비교군 대비 약 두 배의 처리량과 네 배에서 여섯 배의 요청 처리율을 보였습니다.

지원 애플리케이션은 수정 없이 동작하며, 게스트 모듈은 VM 간 매핑에만 필요하도록 변경 범위를 제한했습니다. 표준 인터페이스와 기존 TCP 스택으로 별도 유지할 네트워킹 구현도 줄였습니다.

이렇게 워크로드가 모이거나 떨어져도 기존 논리적 연결을 유지하며 로컬 통신을 가속하는 방법을 제시했습니다.

# 34장 - Thank You

*이 장 사용 시간: 00:15 · 누적 사용 시간: 41:05*

이상으로 발표를 마치겠습니다. 경청해 주셔서 감사합니다. 질문이나 의견을 주시면 답변드리겠습니다.