이상 탐지에서 자동 복구까지

504-GPU LLM 학습의 운영 분석 보고

Lablup Technical Report

Lablup Inc.*

Abstract

대규모 AI 학습은 이제 본질적으로 분산 시스템 문제이며, 하드웨어 장애는 드문 예외가 아니라 일상적 운영 조건이 되었다 [1, 2]. 그러나 프로덕션 학습 클러스터의 운영 증거는 공개 문헌에서 여전히 제한적이다. 본 기술 보고서는 63개 노드 NVIDIA B200 프로덕션 클러스터(504 GPU)를 대상으로, 55일간의 Prometheus 시계열 데이터와 224회 다중 노드 학습 작업 세션을 포괄하는 73일간의 운영 로그를 바탕으로 수행한 실증 분석을 제시한다.

본 보고서는 5개 조직(SKT, Upstage, Lablup, NVIDIA Korea, VAST Data)이 통합 모니터링 파이프라인을 공유하는 조직 간 공동 운영 환경에 기반한다. 우리는 2–4노드 시험에서는 드러나지 않았지만 60노드 규모에서 나타난 스토리지 I/O 병목을 다섯 조직이 함께 진단한 과정을 기록하며, 이를 통해 단일 팀만으로는 격리할 수 없는 운영 규모 현상을 보여준다.

수개월에 걸친 사전학습 운영 기간에서 수집한 지표를 바탕으로 세 가지 정량 분석을 수행하였고, 그 결과 네 가지 발견을 도출하였다. 첫째, 장애 전조 탐지에서는 751개 Prometheus 지표와 XID로 특정된 10건의 GPU 장애를 대상으로 통계 분석을 수행하였다. 장애 유형 전반에 걸쳐 일관되게 지배적인 단일 지표는 없었으며, 이는 다중 신호 기반 탐지 전략의 필요성을 시사한다. 둘째, 체크포인트 I/O 프로파일링에서는 523개 체크포인트 이벤트를 분석하여 GPU VRAM에서 NFS 서버까지의 저장/로딩 경로를 추적했고, 200 Gbps RoCE 대역폭의 1.4–10.4%만 활용되는 "대역폭 패러독스"의 원인이 128-slot NFS, 네트워크 공유 파일시스템(Network File System) 원격 서버에 작업 요청(RPC, Remote Procedure Call) 계층 포화에 있음을 규명하였다. 셋째, 노드 제외 패턴 분석에서는 73일간 224회 다중 노드 학습 작업 세션을 분석하여 63개 노드 중 상위 3개 노드가 전체 제외의 50% 이상을 차지하는 집중된 분포를 확인하였다. 넷째, 자동 재시도 체인 분석에서는 12개 체인(총 73회 시도)의 체인 성공률 33.3%가 수동 복구의 12.5%보다 2.7배 높음을 정량화하였고, 재시도 간격의 중앙값은 11분(IQR 10–11분)이었다.

이 모든 분석은 작업 세션 수준에서의 워크로드 관리, GPU 중심 스케줄링, 통합 관측 체계를 제공하는 프로덕션 인프라 위에서 수행되었다.

*본 보고서는 "Lablup Inc. (2026)"로 인용해 주시기 바랍니다. 전체 저자 목록은 Section D에 있습니다. 관련 문의는 https://www.lablup.com/contact을 통해 보내 주세요.

Contents

1 서론

1.1 연구 배경

모델 규모가 1,000억 파라미터를 넘어서면 학습은 고립된 알고리즘 실행이 아니라 장기간 지속되는 분산 시스템 운영으로 기능하게 되었다. 단일 학습 run이 수백 대의 GPU를 수 주 동안 동기화된 상태로 유지해야 하므로, 안정적인 인프라를 전제하는 기존의 가정은 더 이상 성립하지 않는다. 예를 들어 Solar Open 기술 보고서 [3]는 60개 노드 NVIDIA B200 클러스터에서 1,020억 파라미터 MoE 모델을 학습하는 동안 다음과 같은 문제들이 발생했음을 문서화하였다.

- 초기 B200 배포 환경에서는 CUDA 13.0과 신규 GPU 아키텍처에 대한 소프트웨어 지원이 충분히 안정화되지 않아 호환성 문제가 반복됨. 특히 Triton 기반 커널은 백엔드 컴파일 오류로 그래프 컴파일에 실패할 수 있었고, ScaledDotProductAttention 경로에서도 커널 지원 또는 컴파일 문제가 발생하여 대체 구현 필요 ([3], Section 3.3.4)
- FSDP2에서의 다중 노드 확장 시 성능 저하: 16개에서 60개 노드로 확장 시 TPS가 5,500에서 4,267로 감소하여, HSDP를 통한 반복적 튜닝으로 처리량을 회복해야 했음 ([3], Section 3.3.2)
- 시그모이드(sigmoid) 연산 후 라우터 dtype 불일치(수정 시 13.7% 속도 향상), 불필요한 그룹 GEMM 패딩 오버헤드(빠른 경로 우회 시 14.5% 성능 향상), 과도한 토큰 배칭으로 인한 gradient norm 불안정 ([3], Sections 3.3.3–3.3.4)
- I/O lock 경합으로 인해 초기화에 8시간 이상이 소요되는 데이터 로딩 병목 ([3], Section 3.3.5)

이 사례들을 통해 대규모 학습을 중단, 재시작, 성능 변동이 반복되는 시스템 문제로 다루어야 함을 알 수 있다. 따라서 인프라와 오케스트레이션도 모델 설계만큼 중요한 분석 및 보고의 대상이다.

1.2 문제 정의

대규모 AI 인프라는 서로 강하게 얽힌 세 가지 과제에 직면해 있다.

낮은 자원 활용률. GPU는 고가이고 희소한 자원임에도 불구하고, 정적 할당 정책과 보수적인 운영 관행으로 인해 실제 활용률은 낮은 수준에 머무르는 경우가 많다. Microsoft의 프로덕션 GPU 클러스터 분석에 따르면 중앙값 GPU 활용률은 약 52%였다 [4].

확장성-안정성 충돌. 클러스터 규모가 커질수록 하드웨어 장애, 네트워크 지연 문제, 드라이버 오류가 더 빈번해지며, 학습 작업이 완전히 중단될 가능성이 높아진다. Meta는 16K-GPU 규모의 Llama 3 사전학습 중 수백 건의 예기치 않은 중단이 발생했으며, 대부분이 하드웨어 문제에 기인했다고 보고하였다 [1](Section 2에서 이 내용을 자세히 다룬다).

운영 복잡성. 프레임워크, 드라이버, 라이브러리 조합의 다양성은 환경 재현성을 저해하고 실험 품질의 변동성을 초래한다. 메가스케일(MegaScale)의 배포 경험은 이러한 소프트웨어 이질성을 대규모로 관리하는 것이 지속적인 운영 부담임을 보여준다 [5].

이 과제들에서 문제가 발생할 경우 연결된 문제일 가능성이 높기 때문에, 인프라 전반에 걸친 통합적 대응이 필요하다.

1.3 기여

본 보고서는 2025년 8월부터 12월까지 NVIDIA B200 프로덕션 클러스터(63개 노드, GPU 504장)에서 진행된 Solar Open 학습 과정에서 수집된 데이터를 분석한다. 이를 통해 크게 두 가지 면에서 기여할 수 있을 것이다. 먼저, 이 보고서에 나타난 모든 분석을 가능하게 한 운영 환경 자체를 정리한 연구 환경에 대한 기여이고, 다른 하나는 프로덕션 데이터로부터 도출한 네 가지 정량적 발견이다.

연구 환경 기여.

(S1) **조직 간 공동 운영 환경.** 5개 조직(SKT, Upstage, Lablup, NVIDIA Korea, VAST Data)이 협업하는 운영 환경과, 운영 규모에서만 모습을 드러낸 60개 노드 규모의 스토리지 I/O 병목 사례를 함께 문서화한다. 이로써 대규모 학습 현상이 2–4노드 사전 테스트로는 예측될 수 없으며, 단일 팀 모니터링만으로는 이 규모의 근본 원인을 식별하기 어렵다는 점을 보인다 (Section 3.5).

정량 발견.

(F1) **장애 전조 탐지.** XID로 특정된 10건의 GPU 장애에 751개 Prometheus 지표의 통계적 분석을 적용한다. 장애 유형별 지표 일관성이 두드러지지 않음을 확인하였으며, 사전 검출률 향상을 위한 시계열 ML 모델링이 진행 중임을 보고한다(Section 4.1).

(F2) **체크포인트 I/O의 프로파일링.** 55일간의 운영 데이터에서 523개 체크포인트 이벤트를 정량 분석하고, GPU VRAM에서 NFS 서버까지의 저장/로딩 경로를 프로메테우스(Prometheus) 지표로 프로파일링한다. 또한 그 결과로 200 Gbps RoCE 대역폭의 1.4–10.4%만 활용되는 대역폭 패러독스의 원인이 NFS RPC 프로토콜 계층의 128 slot 포화에 있음을 확인한다 (Section 4.2).

(F3) **노드 제외 패턴 분석.** 73일간 224회 다중 노드 학습 작업 세션의 노드 제외 빈도를 분석하여, 63개 노드 중 상위 3개 노드(gpu074, gpu119, gpu086)가 전체 제외의 50% 이상을 차지하는 집중하는 분포적 특징을 보이고 그 운영적 함의를 설명한다(Section 4.3.1).

(F4) **자동 장애 복구 심층 평가.** 12개 자동 재시도 체인(총 73회 시도)을 분석하여 체인 성공률(33.3%, 수동 대비 2.7배), 재시도 간격의 예측 가능성(중앙값 11분, IQR 10–11분), 다운타임 단축 효과(중앙값 1.9시간 vs 수동 3.3시간)를 정량화한다. 또한 구조적 장애에서의 한계를 함께 분석한다(Section 4.3.2).

2 대규모 학습에서의 장애 유형

본 절에서는 최근 프로덕션 보고서 [3, 1, 5]의 실증 데이터를 바탕으로 클러스터 규모의 장애 특성을 정리하고, 그에 대응하는 인프라 운영 요구사항(Section 3)을 도출한다.

2.1 대규모 학습의 장애 빈발 특성

대규모 학습 환경에서 장애는 예외가 아니라 예상 가능한 운영 특성이다 [2, 1]. 클러스터 규모가 커질수록 장애 빈도도 함께 증가한다. Meta는 최대 16,384개의 H100 GPU 클러스터에서 Llama 3 405B 모델을 54일간 사전학습하는 동안 419건의 예기치 않은 중단이 발생했다고 보고했다 [1]. Meta의 RSC-1, RSC-2 연구 클러스터 역시 총 약 24,000개의 A100 GPU 규모에서 클러스터 크기에 비례하는 장애 빈도를 보였다 [2]. Erben et al.은 현재 GPU 장애율을 기준으로 하면 100,000-GPU 규모에서는 약 30분마다 장애가 발생할 것으로 추산했다 [6].

2.2 대규모 클러스터의 장애 특성

대규모 분산 학습에서는 소수 노드의 문제가 클러스터 전체에 영향을 미친다. 다중 노드 학습은 워커 간 강한 동기화를 전제로 하므로, 단일 노드의 GPU 장애나 통신 이상만으로도 전체 작업이 중단될 수 있기 때문이다. 이에 대한 대표적 운영 대응이 노드 제외(node exclusion), 즉 특정 노드를 다중 노드 할당에서 배제하는 방식이다. 노드 제외는 하드웨어 장애 확인, 성능 저하 관찰, 운영자의 예방적 판단이 복합적으로 반영된 결과이며, 따라서 하드웨어 결함과 일대일로 대응하지 않는다 [2]. Section 4.3.1에서는 우리 클러스터의 노드 제외 패턴을 상세히 분석한다.

우리 클러스터의 장애를 더 넓은 맥락에 놓기 위해 대규모 레퍼런스와 비교하였다. Table 1는 약 1,500개 노드(10,000개 이상 GPU) 규모의 프로덕션 GPU 클러스터를 대상으로 한 ByteDance

Minder 시스템 [7]의 장애 분류 체계를 정리한 것이다. Minder는 CPU 및 GPU 활용률, PFC 카운터, 네트워크 처리량, 디스크 I/O, 메모리 등 시스템 전반 지표를 이용해 장애를 탐지하고 분류한다.

Table 1: ByteDance Minder 시스템의 장애 분류 체계

범주	장애 유형	설명	비율 (%)
호스트 내 HW	ECC 오류	GPU 메모리 데이터 손상 또는 유실	38.9
	PCIe 다운그레이딩	PCIe 링크 장애로 인한 전송률 저하	6.6
	NIC 탈락	OS에서 NIC 인식 불가	5.7
	GPU 카드 탈락	호스트에서 GPU 카드 분리	2.0
	NVLink 오류	GPU 간 NVLink 상호연결 장애	1.7
	AOC 오류	능동형 광케이블 오류	0.9
	소계		**55.8**
호스트 내 SW	CUDA 실행 오류	CUDA 프로그램 실행 실패	14.6
	GPU 실행 오류	페이지 폴트, OOM, 또는 GPU 행	7.7
	HDFS 오류	체크포인트 I/O 오류	5.7
	소계		**28.0**
호스트 간 NW	머신 접근 불가	SSH 또는 VM 서비스 장애	6.0
	소계		**6.0**
기타	—	—	10.3

Minder [7] Table 1 및 Appendix A 기반. ByteDance 프로덕션 GPU 클러스터에서 관측.

Minder는 시스템 전반의 지표를 종합 분석하여 장애를 분류하는 반면, 우리 클러스터는 NVIDIA GPU 커널 드라이버가 `dmesg`에 기록하는 *XID* 에러 코드(GPU 드라이버가 보고하는 수치형 장애 식별자)를 주된 장애 탐지 수단으로 사용한다. 각 XID 코드는 특정 장애 유형에 대응한다(예: XID 79 = GPU 카드 탈락, XID 94 = ECC 오류, XID 145/149 = NVLink 오류). 이후 절에서 제시하는 장애 사례 분석과 노드 제외 분석은 이 XID 기록에 기반한다.

두 시스템은 모니터링 방식과 인프라 구성이 다르므로, 장애 분류 범위가 완전히 겹치지 않는다. 주요 차이는 다음과 같다.

- **HDFS 오류**: 우리 클러스터는 HDFS(Hadoop Distributed File System)가 아닌 NFS(Network File System)를 사용하므로 동일 분류는 적용되지 않는다. 다만 NFS 기반 체크포인트 I/O 문제는 Section 4.2.5에서 별도로 분석한다.
- **PCIe 다운그레이딩**: XID 에러를 발생시키지 않고 대역폭 저하로만 나타나므로, 본 보고서의 XID 기반 분석 범위에 포함되지 않는다 (별도의 대역폭 모니터링이 필요).
- **NIC 탈락 및 AOC 오류**: NIC(Network Interface Card)과 AOC(Active Optical Cable)는 GPU 드라이버 계층 외부에서 발생하여 XID로 보고되지 않으므로, 본 분석 범위에서 제외한다(전용 네트워크 모니터링이 필요).
- **CUDA 실행 오류**: 애플리케이션 또는 프레임워크 수준 오류로, 본 보고서의 하드웨어 장애 분석 범위에서 제외한다.

따라서 본 보고서의 분석은 XID로 탐지 가능한 장애(GPU 카드 탈락, ECC(Error-Correcting Code) 오류, NVLink 장애)에 집중하며, 별도의 모니터링 인프라를 필요로 하는 XID 범위 밖의 장애 유형은 다루지 않는다.

Table 2는 XID 코드를 Minder 장애 범주에 매핑하여, 55일 관측 기간 동안 우리 클러스터에서 기록된 17건의 장애 이벤트를 분류한 것이다.

Table 2: 우리 클러스터의 장애 분포, Minder 범주에 매핑

장애 유형 (Minder 범주)	XID 코드	건수	비율 (%)
NVLink 오류	145, 149	5	29.4
ECC 오류	94	2	11.8
GPU 카드 탈락	79	2	11.8
GPU 실행 오류	119	1	5.9
머신 접근 불가	—	2	11.8
기타 (성능 저하 등)	—	5	29.4
합계		**17**	**100.0**

63노드 클러스터 (504 B200 GPU), 55일 관측 기간. XID 코드를 Minder 장애 범주에 매핑하여 분류.

두 분포는 뚜렷한 차이를 보인다. Minder에서는 ECC 오류(38.9%)가 최다 범주인 반면, 우리 클러스터에서는 NVLink 오류(29.4%)가 가장 빈번하다. XID 코드는 NVLink 장애(XID 145/149)를 명시적으로 기록하며, ECC 이벤트는 짧은 관측 기간으로 인해 표본이 적다. 기타 범주(29.4%)에는 성능 저하 등 XID 코드에 직접 매핑되지 않는 운영 수준 이벤트가 포함된다. 이러한 차이는 모니터링 방식과 기간, 워크로드 크기 분포의 차이 때문에 발생한다. Section 4에서 개별 장애 사례를, Section 4.3.1에서 이러한 장애가 관측 기간 동안 노드 제외 패턴으로 어떻게 이어지는지 분석한다.

성능 저하형(Fail-slow) 장애와 지연 노드(straggler) 탐지. 위 분류는 GPU나 노드가 완전히 멈추는 정지형 장애(*fail-stop*) 장애에 초점을 두고 있다. 그러나 우리 클러스터의 노드 제외 데이터(Section 4.3.1)를 통해서 또 다른 유형의 장애의 유형이 있음을 알 수 있었다. gpu074, gpu119[1] 같은 노드는 완전히 정지하지 않았음에도 학습 속도를 떨어뜨려 반복적으로 제외되었다. 이는 구성 요소가 계속 동작하지만 성능이 저하된 상태인 성증 저하형 장애의 대표적 사례이다. 분산 학습은 반복할 때마다 모든 워커를 동기화하므로, 단 하나의 느린 노드가 전체 작업을 지연시킬 수 있다. 또한 명시적 오류 코드 없이도 나타날 수 있기 때문에 성능 저하형 장애는 정지형 장애보다 감지가 더 어렵다.

최근 프로덕션 연구들도 이 패턴이 널리 존재함을 보고한, Wu et al. [8]은 10,000+ GPU 클러스터에서 대규모 학습 작업(512–1,024 GPU)의 59%가 성능 저하형 장애형 장애의 지연 노드를 경험했고, 평균 작업 완료 시간이 34.59% 지연되었다고 보고하였다. Lin et al. [9]은 프로덕션 LLM 학습 클러스터에서 42.5%의 작업이 straggler의 영향을 받아 전체 GPU 시간의 10.4%가 손실된다고 밝혔으며, 주요 원인을 하드웨어 장애보다 파이프라인 단계 불균형이나 가비지 컬렉션 중단 같은 워크로드 수준의 불균형에서 찾았다. 우리 클러스터에서는 학습 반복별 처리량을 계측하지 않았기 때문에, 운영자가 세션 간 속도 차이를 인지한 뒤에야 성능 저하 노드를 식별할 수 있었다. Section 7.2에서는 이러한 사후 대응을 대체하기 위한 모니터링 확장 방향을 논의한다.

2.3 XID 오류 분류 및 복구 전략

NVIDIA GPU는 XID 오류 코드를 통해 하드웨어 및 소프트웨어 오류를 보고할 수 있다 [10]. 서로 다른 XID 코드는 근본적으로 다른 대응을 필요로 하므로, 이러한 코드를 올바르게 분류하는 것이 장애 귀인에 핵심적이다. Table 3는 우리 클러스터에서 관측된 XID 코드를 NVIDIA 정의 및 해결 유형별로 분류한 것이다.

[1]본 보고서에서 "gpuXXX"는 클러스터 내 노드 식별자(예: gpu074 = 74번 노드)를 나타내며, "GPU#N"은 해당 노드 내의 개별 GPU 인덱스(0–7)를 나타낸다. 각 노드에는 8개의 B200 GPU가 탑재되어 있다.

Table 3: 해결 조치별 XID 오류 코드 분류

XID	설명	해결	조치
하드웨어 조치 필요 (노드/*GPU* 리셋 필요)			
79	GPU가 버스에서 탈락	RESTART_BM	노드 재부팅
119	GSP RPC 타임아웃	RESET_GPU	GPU 리셋
145	NVLink RLW 오류	RESET_GPU	GPU 리셋
149	NVLink NETIR 오류	RESET_GPU	GPU 리셋
애플리케이션 수준 (프로세스 재시작으로 충분)			
31	GPU 메모리 페이지 폴트	RESTART_APP	작업 세션 재시작
43	GPU 처리 중단	RESTART_APP	작업 세션 재시작
94	격리된 ECC 오류	RESTART_APP	자동 정정

이 분류는 Backend.AI의 장애 처리 전략(Section 4.3)에 반영된다. 하드웨어 조치가 필요한 XID 오류(79, 119, 145, 149)는 노드 격리 및 예비 노드로의 작업 세션 마이그레이션을 트리거하며, 애플리케이션 수준 오류(31, 43, 94)는 해당 노드를 제외하지 않고 자동 재시도로 처리된다. 실제로 장애율은 시스템 계층(OS 커널, GPU 드라이버, 펌웨어 버전)과 워크로드 강도에도 좌우된다.

3 운영 인프라

Section 2의 장애 특성을 고려하면, 인프라는 두 가지 핵심 요구사항을 충족해야 한다.

1. **탐지, 격리, 복구를 통한 장애 내성:** 탐지는 하드웨어(GPU ECC 오류, 온도, 전력), 프로세스(컨테이너 상태, OOM(Out of Memory)), 애플리케이션(학습 진행, 손실 궤적), 네트워크(NVIDIA Collective Communications Library (NCCL) 타임아웃, 대역폭 저하) 등 여러 계층에서 작동한다. 손실 발산 등 애플리케이션 수준 판단은 학습 프레임워크에 위임한다. 장애를 탐지하면 문제 노드나 GPU를 스케줄링 대상에서 격리하여 장애 영향 범위를 제한한다. 복구 단계에서는 대체 자원을 할당하고 작업 세션을 재시작하되, 체크포인팅은 학습 프레임워크에 위임해 총 학습 처리량 손실을 최소화한다.

2. **작업 세션 수준 생명주기 관리.** 학습 작업의 생명주기는 작업 세션 수준에서 관리한다. 작업 세션은 여러 노드에 걸친 하나 이상의 컨테이너를 묶은 논리적 단위로, 체크포인트와 옵티마이저 상태 같은 학습 상태와 연결되어 있다. 작업 세션을 재시작하면 마지막 체크포인트에서 작업을 다시 시작한다. 시스템은 이러한 상태 전환을 빠짐없이 기록하고 보존하므로, 정확한 복구와 감사가 가능하다.

두 요구사항의 바탕에는 GPU를 1차 스케줄링 자원으로 두고, CPU와 메모리는 GPU가 배치된 위치에 맞춰 함께 할당되는 자원으로 다룬다는 전제가 있다. 전통적인 CPU 중심 오케스트레이션은 이러한 전제가 필수적이지 않았다.

다음으로는 이러한 요구사항을 구현하는 인프라 요소를 살펴보겠다. 전체 클러스터의 인프라 계층은 학습 프레임워크(가령, PyTorch, DeepSpeed, Megatron-LM 등) 아래에서 동작하며, 학습 환경 제공 외에도 자원 할당, 체크포인트 저장, 장애 알림 등을 담당한다. 장별 기술은 이후 분석의 맥락을 제공하기 위해 먼저 클러스터 하드웨어를 요약하고(Section 3.1), 이어 작업 세션 추상화와 복구 메커니즘(Section 3.2), 다중 계층 모니터링 파이프라인(Section 3.4)을 설명한다. 분석에서 직접 참조되는 구성 요소를 중심으로 서술하며, 세부 구현은 Appendix A에서 다룬다.

3.1 운영 클러스터

운영 클러스터는 63노드 NVIDIA DGX B200 시스템이다. 해당 클러스터의 하드웨어 구성을 요약하면 Table 4와 같다.

Table 4: 운영 클러스터 하드웨어 구성

구성요소	사양
노드 수	63대 (DGX B200): 학습용 60대, 스페어 3대
GPU	노드당 8× NVIDIA B200 (총 504개)
GPU 메모리	GPU당 192 GB HBM3e
시스템 RAM	노드당 2 TiB DDR5
GPU 간 통신 (노드 내)	NVLink 5세대, 양방향 1.8 TB/s
노드 간 통신	8포트 × 400G InfiniBand (3.2 Tbps/node)
스토리지 네트워크	200G RoCE(RDMA over Converged Ethernet)
스토리지	VAST Data E-Box, 2 PiB, NFS 마운트

학습에 필요한 가중치와 중간 계산 결과, 그리고 갱신된 정보가 각 GPU의 192,GB HBM3e 메모리에 올라가며, GPU는 이를 직접 읽고 쓰면서 연산을 수행한다. [11].

Forward/backward pass에서는 각 GPU의 SM이 Tensor Core와 CUDA Core를 사용해 연산을 수행하고, 노드 내 8개 GPU는 NVLink를 통해 텐서 병렬화 통신을 처리한다. 그래디언트 동기화(AllReduce)는 InfiniBand NDR(노드당 8포트 × 400G)을 통해 63개 노드에 걸쳐 수행되며, 이때 단 1개 노드라도 느려지면 전체 학습이 대기해야 하는 지연 노드(straggler) 문제가 발생한다 [12]. 체크포인트 저장과 데이터 로딩은 전용 200G RoCE 네트워크를 통해 VAST Data NFS 스토리지에 접근한다. compute(InfiniBand), storage(RoCE), management(Ethernet) 트래픽은 물리적으로 분리되어 상호 간섭을 방지한다.

이 통신 구조는 이후 분석의 물리적 기반이 된다. Section 4.2.5의 NFS RPC 분석은 스토리지 평면의 병목을 다루고, Section 4.1의 전조 분석에서는 GPU 원격 계측 데이터(DCGM) 외에 호스트 시스템 지표(인터럽트, 프로세스 상태, NFS I/O 등)가 장애 시점 이상 신호로 활용 가능하다는 것을 설명하겠다.

하드웨어 위에서 동작하는 소프트웨어 계층은 다음 Table 5에서 확인할 수 있다. 상단은 기본 컨테이너 이미지와 핵심 라이브러리(CUDA, cuDNN, NCCL, PyTorch)를, 하단은 이 클러스터에서 Solar Open 프로젝트 [3]가 사용한 학습 구성(병렬화 전략, 배치 크기, 시퀀스 길이 진행, 정밀도 형식) 예시이다.

Table 5: 운영 학습 소프트웨어 계층

구성요소	버전
기본 이미지	NGC PyTorch 25.08
CUDA	13.0.0
cuDNN	9.12.0
NCCL	2.27.7
PyTorch	2.9.0.dev20250830+cu130
TorchTitan	PyTorch nightly에 포함
Transformers	4.55.2
Flash Attention	2.7.4.post1
Python	3.12
학습 구성: *Solar Open* [3]	
병렬화	HSDP (10노드 샤딩 그룹 × 6 레플리카)
글로벌 배치 크기	13,440, GPU당 28
시퀀스 길이	4K → 32K → 100K (점진적 확장)
정밀도	FP8 + bfloat16 혼합

다음 절에서는 본 클러스터에서 사용된 오케스트레이션 계층의 작업 세션 관리 및 노드 격리 메커니즘을 기술하겠다.

3.2 작업 세션 추상화

딥러닝 학습 워크로드는 반복 전반에 걸쳐 옵티마이저 파라미터와 학습률 스케줄을 보존하는 상태 유지(stateful) 작업이다 [1, 5]. 장애 시 컨테이너는 완전히 소실되지만, 학습 작업 세션은 마지막 체크포인트 이후의 진행분만 소실된다. Backend.AI는 이 차이를 반영하여, 스토리지 볼륨과 생명주기 상태를 번들링하는 작업 세션을 컨테이너 대신 핵심 관리 단위로 사용한다(Table 6).

Table 6: 컨테이너와 작업 세션 특성 비교

특성	컨테이너	작업 세션
생명주기	프로세스 종료 = 완료	체크포인트까지 진행 = 완료
상태	상태 비저장 지향	상태 유지: 옵티마이저, 그래디언트
재시작 의미	처음부터 시작	체크포인트에서 재개
장애 영향	프로세스 소실	마지막 체크포인트 이후 진행분 소실

이 구분은 Section 4.3의 자동 복구 분석과 직결된다. "재시작"이 처음부터가 아닌 체크포인트 재개이므로, 복구 시간은 체크포인트 로딩 시간(중앙값 31분, Section 4.2.3)으로 결정된다.

3.3 Sokovan 스케줄러

작업 세션에 GPU를 할당하는 것은 소코반(Sokovan) 스케줄러가 담당한다. 스케줄링은 두 가지 수준에서 작동한다. 클러스터 수준에서는 대기 중인 작업 세션을 자원 그룹에 대해 평가하여 밀도와 우선순위를 제어하고, 노드 수준에서는 NUMA 인식 배치 정책을 적용하여 GPU, CPU 코어, 메모리를 동일 NUMA 노드에서 할당한다(Figure 1). 우리는 이전 운영 과정에서 이러한 동일 NUMA 노드 배치가 NUMA 노드 간 메모리 접근을 줄이고 처리량을 최대 1.30× 향상시킬 수 있음을 확인하였다 [13].

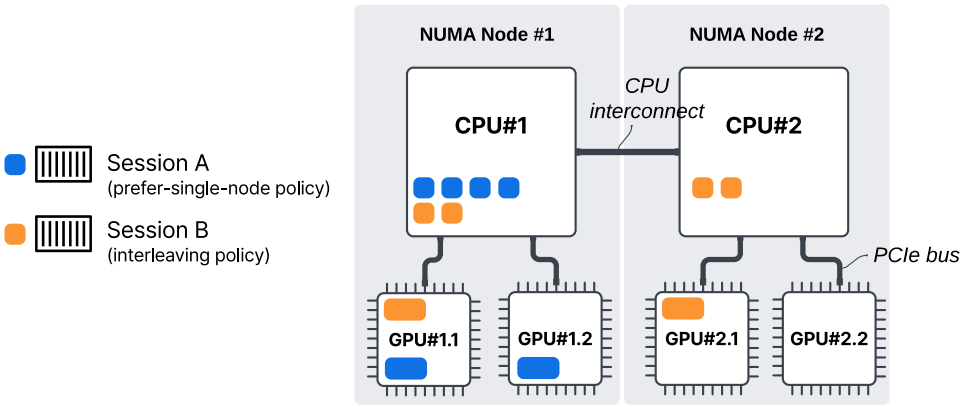

Figure 1: GPU 서버 노드 내 NUMA 인식 자원 할당 ([14]에서 발췌). Session A는 *prefer-single-node* 정책으로 NUMA Node #1에서 모든 자원을 할당하고, Session B는 *interleaving*(양쪽 NUMA 노드에 자원을 나누어 배치하는 방식) 정책으로 자원을 분산 배치한다.

분산 학습 작업은 일부 노드만으로 먼저 시작할 수 없고, 참여 노드가 모두 동시에 준비되어야 한다. 60노드 학습의 경우 60개 노드를 한꺼번에 할당해야 하는데, 부분 할당은 NCCL 초기화 중 나머지 참여자를 기다리는 교착 상태를 초래할 수 있기 때문이다. Sokovan은 이를 방지하기 위해 갱 스케줄링(gang scheduling)을 적용하며, N개 슬롯을 모두 할당하거나 전체 요청을 큐에 보관하는 전부 또는 전무(all-or-nothing) 방식으로 동작한다. 이를 통해 부분 할당된 작업이 나머지 슬롯을 기다리는 동안 GPU가 유휴 상태에 머무르는 자원 파편화를 방지한다. 이 제약은 Section 4.3.5에서 분석하는 자동 재시도 실패의 구조적 원인(가용 노드가 60개 미만일 때 재시도가 반복 실패하는 현상)과 직결된다.

3.4 다중 계층 모니터링

GPU 상태를 모니터링하고 관리하기 위한 도구인 NVIDIA DCGM(Data Center GPU Manager)만으로는 장애의 전조를 포착하기 어려운 경우가 많다. DCGM으로는 GPU의 온도, 전력, 사용률 같은 상태를 관찰할 수 있지만 실제 장애는 GPU보다 스토리지 I/O 지연, 네트워크 병목, 스케줄러 문제, 프로세스 비정상 종료 같은 다른 계층에서 먼저 시작되는 경우가 많기 때문이다. DCGM 단독 모니터링과 통합 다중 계층 모니터링의 차이를 표로 정리하면 Table 7과 같다.

Table 7: GPU 전용 모니터링과 다중 계층 모니터링의 비교

	DCGM 단독	**통합 모니터링 (스케줄러 + OS + DCGM)**
탐지 시점	장애 발생 이후 (사후적)	장애 발생 이전 (전조 탐지 가능)
관측 범위	GPU 칩 상태	OS, 네트워크, 스케줄러를 포함한 전체 계층

위의 Table 7 수 있는 것처럼 DCGM에서 제공하는 XID 에러 코드가 기록되는 시점은 GPU가 이미 정지한 이후이다. 반면, 시스템 지표(TCP 소켓 할당, 커널 메모리, 인터럽트)와 스케줄러 수준 지표(비동기 작업 수, RPC 지연)은 GPU 드라이버나 NCCL 통신 계층의 이상이 GPU 원격 계측 데이터보다 먼저 표면화되는 경로를 제공한다. 다음 Table 8은 우리 클러스터에서 Prometheus 호환 익스포터를 배포하여, 30초 간격으로 수집한 주요 지표를 나타낸다(Table 8).

Table 8: 노드당 배포된 Prometheus 익스포터

익스포터	**범위**	**주요 지표**
DCGM-exporter [15]	GPU	활용률, 온도, 전력, ECC 오류, NVLink, XID
node_exporter [16]	OS / 하드웨어	CPU, 메모리, 디스크 I/O, 네트워크, InfiniBand, NFS
`all-smi` [17]	크로스 플랫폼 GPU	GPU 전력, 시스템 메모리, 샤시 수준 원격 계측 데이터
Backend.AI	스케줄러	API 지연, 활성 작업 세션, RPC 지표

63개 노드에 걸쳐 총 약 751개의 고유 지표 이름이 수집된다. 이 중 장애 분석에 실제로 사용한 것은 약 305개의 활성 지표이며, ZFS 통계, Go 런타임 내부 등 분석 대상이 아닌 지표는 제외하였다. 55일간 연속 수집하여 약 126 GB의 비압축 원시 원격 계측 데이터가 생성되었으며, Prometheus 호환 시계열 데이터베이스인 VictoriaMetrics에 저장된다.

3.5 조직 간 공동 운영 환경

클러스터는 SK Telecom(클라우드 운영), Upstage(모델 개발), Lablup(Backend.AI 인프라), NVIDIA(하드웨어), VAST Data(스토리지) 등 5개 조직이 공동 운영한다. 각 조직에서 수집되는 운영 지표는 통합 모니터링 파이프라인(Section 3.4)을 통해 집계되며, 애플리케이션, 스케줄러, 네트워크, 스토리지 계층의 상태를 동일 시점 기준으로 함께 조회·분석할 수 있도록 구성되어 있다. 본 연구는 이러한 통합 관측 체계 외에도 운영 사례를 분석 한다. 수치로 나타나지 않은 지표 중에서 서로 다른 계층의 지표를 구성원들이 직접 비교하면서 원인을 분석할 수 있었기 때문이다.

대표 사례: 운영 규모에서 드러난 스토리지 I/O 병목. 운영 환경은 60노드 B200 학습 구성을 배포하는 과정에서 정교화되었다. 초기 검증 단계에서는 별다른 문제가 관찰되지 않았으나, 운영 규모로 확장한 뒤 예상하지 못한 I/O 병목이 나타났다. VAST 스토리지 기반 학습 초기화는 수 분 내에 완료될 것으로 예상되었지만, 실제 초기화에는 8시간 이상이 소요되었고 지속 처리량 역시 이론적 한계를 크게 밑돌았다. 이 현상은 Solar Open 테크니컬 리포트에서도 보고된 것과 같이 공동 검증 결과 I/O 잠금 경합이 주요 원인인 것으로 밝혀졌다 [3].

초기에는 문제의 원인을 특정 계층만으로 설명하기 어려웠다. 애플리케이션 로그에서는 학습 초기화 지연만 확인되었고, 스토리지 지표만으로는 어떤 접근 패턴이 병목을 유발하는지 파악하기 어려웠다. 이후 애플리케이션, 스케줄러, 네트워크, 스토리지 지표를 동일 시점 기준으로

함께 상관 분석한 결과, 애플리케이션이 의도한 대규모 순차 I/O와 실제 스토리지 계층에 전달된 단편화된 소규모 랜덤 I/O 사이의 차이가 병목의 원인으로 확인되었다.

공동 디버깅 결과, 예상했던 대규모 순차 I/O 패턴과 실제 스토리지 계층에 도달한 단편화된 소규모 랜덤 I/O 패턴 사이의 불일치가 분산 메타데이터 서비스를 포화시키는 것으로 드러났다. 각 노드의 스토리지 NIC 수신 속도(약 4–10 GiB/s)는 단일 노드 관점에서는 특별하지 않았지만, 60노드가 동시에 만들어낸 집계 접근 패턴은 메타데이터 경로를 압도하고 있었다. 애플리케이션 측의 파일 샤딩(랭크별 Arrow 파일 분할)과 스토리지 측의 비동기 삭제 및 readahead 튜닝을 결합한 결과, 초기화 시간을 8시간 이상에서 8분 미만으로 단축할 수 있었다.

이후 분석에 대한 시사점. 이 사례를 통해 Section 4에서 어떤 기준으로 분석해야 하는지에 대한 두 가지로 시사점을 알 수 있다. 첫째, 2–4노드 규모에서 관찰되는 성능 특성만으로는 60노드 규모의 거동을 예측하기 어렵다. 이러한 스토리지·메타데이터 병목은 운영 규모에 이르러서야 드러나므로, 소규모 사전 테스트만으로는 구조적으로 발견하기 어렵다. 둘째, 단일 팀의 고립된 모니터링만으로는 이 규모의 근본 원인을 식별하기 어렵다. 조직 간 공유 지표 파이프라인이 있어야 Section 4와 같은 체계적 운영 분석이 가능하다. 이후의 정량 분석은 이 두 조건을 바탕으로 하고 있다.

4 운영 데이터 분석

본 절에서는 프로덕션 환경에서 수집한 운영 데이터를 기반으로, 공동 디버깅 사례, 장애 전조 분석, 체크포인트/NFS 분석, 자동 복구 분석의 네 가지 사례를 분석한다. 전체 학습 운영 기간 중 프로메테우스(Prometheus) 시계열 데이터로 수집·보존된 55일 구간이 분석 대상이다. Section 4.1에서는 장애 발생 시점의 지표 분석을, Section 4.2에서는 NFS 스토리지 I/O 및 전체 계층에 걸친 체크포인트 데이터 경로를 분석한다. 이어서 Section 4.3에서는 노드 제외 패턴과 자동 복구 메커니즘의 효과를 정량적으로 평가한다.

분석의 목적은 탐지(detection), 원인 파악(diagnosis), 복구(recovery)로 이어지는 운영 흐름을 구성하며, 모니터링 신호가 실제 복구 파이프라인과 어떻게 연결되는지를 보여준다. 이러한 현상이 왜 운영 규모에서만 관찰되는지는 Section 3.5에서 제시한 공동 디버깅 사례를 통해 이미 논의하였다.

4.1 장애 탐지 및 전조 분석

앞 절이 장애 발생 이후의 원인 추적을 다루었다면, 본 절은 장애가 표면화되기 전에 이상 징후를 감지할 수 있는지를 묻는다.

Table 2는 장애를 이벤트 단위로 집계한다. 전조 분석을 위해서는 이를 노드×시점 단위로 더 세밀하게 재구성해야 한다. 그 결과 14건의 클러스터 다운타임에서 총 21건의 장애를 식별하였다[2]. 이들은 GPU 하드웨어 장애 13건(XID 감지 10건, XID 미감지 3건), 성능 저하형(fail-slow) 장애 4건, 원인 미상 4건으로 분류할 수 있다. 이 중 XID 기록을 통해 장애 노드와 시점을 곧바로 특정할 수 있었던 10건만을 지표 교차 분석의 대상으로 삼았다(Section 4.1.1). 나머지 11건은 XID가 없어 자동 위치 특정이 어려웠다.

4.1.1 분석 대상 및 장애 분류

전조 분석을 위해서는 장애 노드와 시점이 특정되어야 한다. 앞서 식별한 21건 중 XID 에러로 노드와 시점이 특정된 10건을 분석 대상으로 선정하였다(Table 9). 장애 유형은 XID 에러 코드 기반으로 분류하였으며, 하나의 사례에 여러 XID가 동시 발생한 경우 모든 유형에 중복 포함하였다(예: gpu071은 NVLink와 Bus Fault 모두에 해당). 10건 전부 GPU 하드웨어 관련 장애였으며, NVLink 오류(XID 145/149)가 6건으로 가장 빈번하였다.

[2] 하나의 다운타임에서 여러 노드가 동시에 장애를 겪을 수 있다. 예를 들어 10/23 다운타임에서는 gpu085와 gpu122가 각각 독립적으로 장애를 일으켰다.

Table 9: 21건의 GPU 장애 중 XID 에러로 노드·시점이 특정된 10건의 개요. 장애 유형은 XID 에러 코드 기반 분류. 세션 실행 날짜는 학습 세션 시작부터 장애 발생까지의 경과 시간.

Node	날짜	장애 유형	XID	세션 실행 시간
gpu071	10/20	NVLink + Bus Fault	79, 145	157.6h
gpu085	10/23	NVLink Error	145, 149	0.8h
gpu122	10/23	ECC Error	94	4.7h
gpu085†	10/23	NVLink Error	145, 149	0.7h
gpu116	10/25	Bus Fault + ECC	79, 94	37.8h
gpu071	11/9	NVLink + Bus Fault	79, 145	44.8h
gpu096	11/17	ECC Error	94	165.3h
gpu123	11/20	NVLink Error	145, 149	1.9h
gpu068	11/24	NVLink Error	145, 149	15.3h
gpu071	11/29	GSP RPC Timeout	119	62.1h

† gpu085는 10/23 같은 날 두 차례 별도 세션에서 장애가 발생하여 독립 사례로 포함.

4.1.2 장애 유형별 전조 패턴

60개 노드가 동일한 학습 작업을 수행하므로, 이상 탐지는 장애 노드가 나머지 59개 정상 노드의 분포에서 얼마나 벗어나는지를 평가하는 문제로 볼 수 있다. 다음 절에서는 XID 코드별로 장애 사례를 분류하고, 각 유형에서 관찰된 대표 지표 패턴을 사례 중심으로 제시한다. 이를 통해 장애 노드가 peer 분포에서 명확히 벗어나는 지표를 식별하고, 자동화된 이상 노드 탐지의 가능성을 보인다.

NVLink 관련 장애(XID 145/149, 6건). Figure 2를 통해 gpu071에서 NVLink 장애와 Bus Fault가 함께 발생한 사례(10/20)를 확인할 수 있다. 상단은 호스트 시스템에서 처리한 인터럽트 수 (`node_intr_total`, 30초당 카운터 증가량), 하단은 현재 실행 가능한 프로세스 수 (`node_procs_running`, 순간값)이다. gpu071은 XID 발생 전까지 두 지표 모두에서 peer 분포와 유사한 값을 보인다. 그러나 XID 발생 직후 인터럽트 수는 일시적으로 급증한 뒤 peer 수준(약 300K)보다 크게 낮은 70K–100K 수준으로 떨어진다. 이는 NVLink 및 Bus Fault 이후 호스트의 인터럽트 처리 패턴이 급격히 변한 결과로 해석된다. 실행 가능 프로세스 수 역시 학습 중에는 peer와 비슷한 수준을 유지하다가, XID 발생 이후 거의 0에 가까운 수준으로 감소한다. 이는 GPU 장애 이후 학습 워커가 종료되었거나 GPU/통신 대기 상태로 전환되면서, CPU에서 즉시 실행 가능한 프로세스가 크게 줄어든 결과로 해석된다.

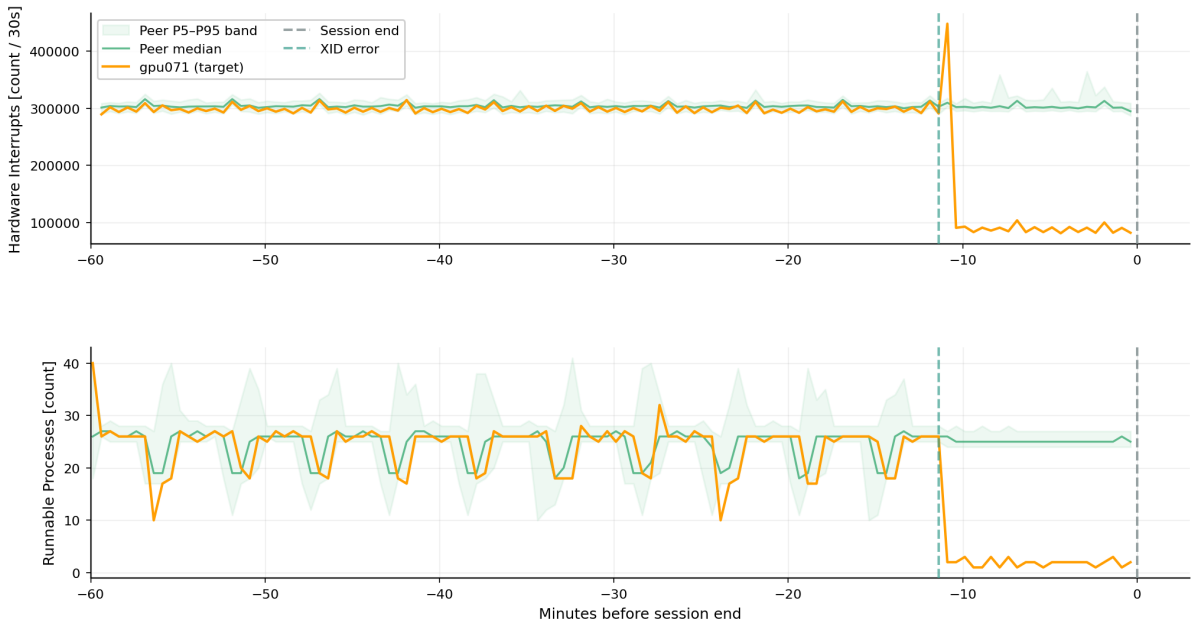

Figure 2: NVLink + Bus Fault (gpu071, 2025-10-20). 상단: 하드웨어 인터럽트(`node_intr_total`, 30초당 카운터 증가량). XID 시점에 peer(약 300K) 대비 약 70K–100K로 급감. 하단: 실행 가능 프로세스 수(`node_procs_running`, 순간값). XID 시점에 워커 프로세스 종료로 0으로 급감.

Contained Memory Error(XID 94, 3건). ECC(Error-Correcting Code)로도 정정하지 못한 다중-비트 메모리 결함 가운데, 그 영향이 오류를 유발한 단일 애플리케이션에만 머무는 경우가 XID 94[3]로 분류된다. 같은 GPU의 다른 작업은 영향받지 않으며, 해당 프로세스만 재시작하면 복구된다. Figure 3은 gpu096에서 발생한 XID 94 사례의 인접 1시간을 보여준다. 상단은 NFS GETATTR 응답 시간(`node_mountstats_nfs_operations_response_time_seconds_total`), 하단은 page-out 횟수(`node_vmstat_pgpgout`)이며, 둘 다 30초 구간 증분이다. 두 지표 모두 XID 시점에 peer 대비 뚜렷한 spike를 보이는데, 특히 GETATTR 응답 시간은 spike 직후 사실상 0으로 떨어진다. XID 94의 정의상 오류와 동시에 워커 프로세스가 종료되므로, 그 워커가 만들어내던 NFS 호출 자체가 사라진 결과로 읽힌다. 동시 spike는 워커 종료 직후 커널이 file handle을 회수하면서 발생한 NFS revalidation, 그리고 남아 있던 dirty page를 디스크로 내보내는 writeback flush의 흔적으로 추정되지만, 본 데이터만으로 인과를 단정하기는 어렵다.

[3]NVIDIA, "Analyzing XID Errors," https://docs.nvidia.com/deploy/xid-errors/analyzing-xid-catalog.html.

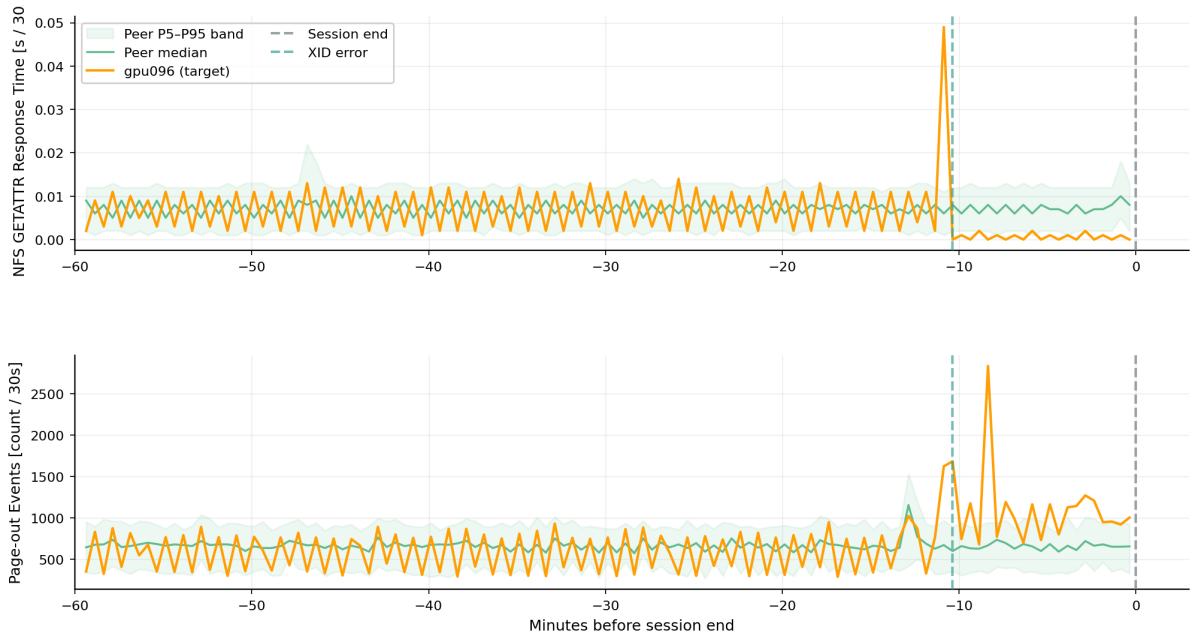

Figure 3: ECC Error (노드 gpu096, 2025-11-17). 상단: NFS GETATTR 응답 시간 (`node_mountstats_nfs_operations_response_time_seconds_total`, 30초 구간 증분). XID 시점에 peer 대비 spike 후 사실상 0으로 급락. 하단: page-out 이벤트(`node_vmstat_pgpgout`, 30초 구간 증분). XID 시점에 spike.

DCGM이 제공하는 메모리 행 재매핑(remapped rows) 카운터는 GPU 메모리에 영구적 결함이 얼마나 쌓이고 있는지를 보여주는 지표다. ECC 오류가 발생하면 결함이 있는 메모리 행(row)을 예비 행으로 교체하는데, 앞에서 이야기한 것과 같이 일시적 오류로 인한 교체와 영구 결함(uncorrectable)으로 인한 교체가 따로 집계된다. 후자가 늘어난다는 것은 그 GPU의 메모리가 회복 불가능하게 손상되고 있다는 신호다. Figure 4 상단은 노드 gpu122(GPU#1)에서 정정 불가 재매핑 카운터가 단계적으로 증가하는 모습을 보여준다. 카운터가 늘어나는 시점마다 XID 94(Contained ECC Error)가 함께 보고되었으며, 결국 해당 GPU의 작업이 중단되었다. 두 신호가 동시에 움직인다는 점은 ECC 오류와 메모리 손상의 진행이 같은 사건의 두 단면임을 시사한다. NVIDIA는 메모리 뱅크당 정정 불가 재매핑이 8회에 이르면 `ROW_REMAP_FAILURE` 플래그를 발동한다. [18]. 이 단계에 도달하면 더 이상의 재매핑이 불가능하므로 GPU 자체를 교체해야 한다.

반면 Figure 4 하단의 노드 gpu124(GPU#2)는 다른 양상을 보인다. 정정 가능(correctable) 재매핑이 약 55일에 걸쳐 누적되었는데, 처음에는 천천히 늘다가 10월 28일경부터 급증해 일주일 만에 200을 넘었고, 이후 254에 이르렀다. 그동안 정정 불가 재매핑이나 XID 에러는 한 번도 발생하지 않았으나, 결국 GPU가 호스트에서 인식되지 않는 상태가 되어 교체되었다. NVIDIA 문서 [18]는 정정 가능 에러를 하드웨어가 자동 복구하는 항목으로 분류하는데, 본 사례처럼 누적 속도가 급변하는 추세는 즉각적 장애 신호는 아니더라도 메모리 상태가 악화되고 있다는 신호일 수 있으므로, 단일 카운트 값보다 증가 추세를 추적할 필요가 있다.

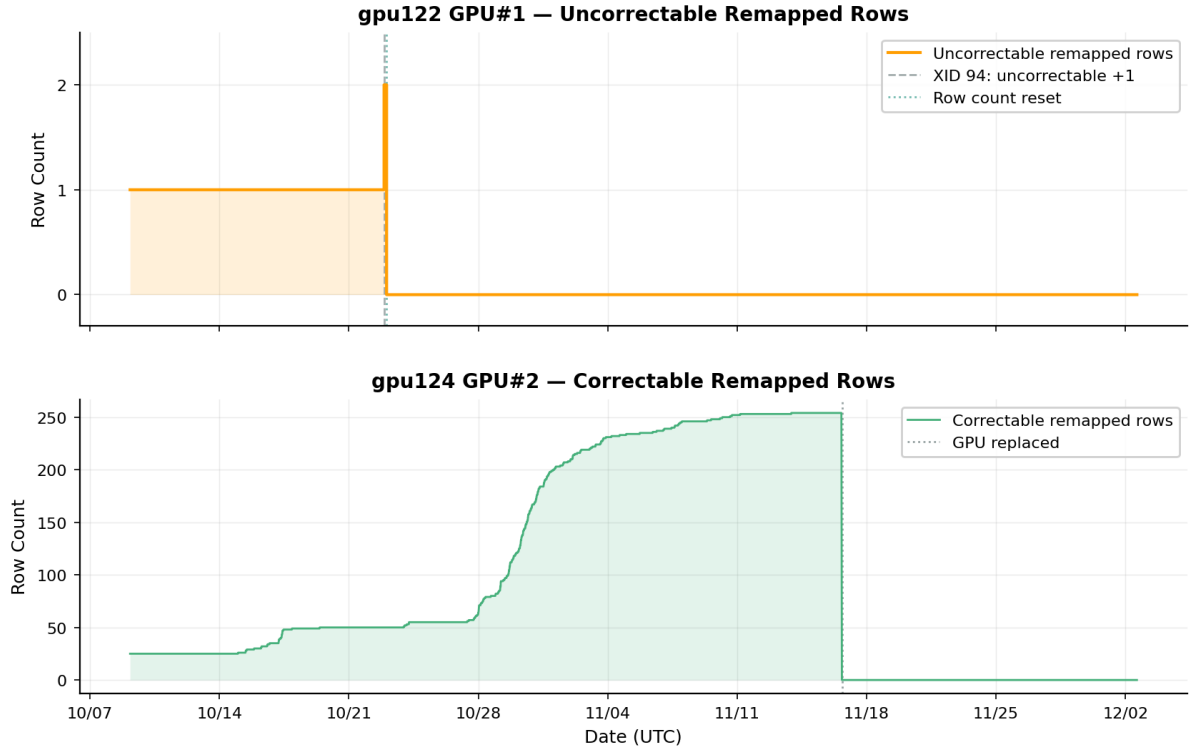

Figure 4: ECC 메모리 행 재매핑 타임라인. 상단: gpu122 GPU#1 정정 불가(영구 결함) 재매핑. 재매핑이 늘어난 시점에 XID 94 ECC 오류가 함께 발생하여 GPU 정지. 하단: gpu124 GPU#2 정정 가능(자동 정정) 재매핑. 55일간 254행까지 점점 빠르게 누적되었으나 XID 오류는 발생하지 않음. 11/17일 GPU가 시스템에 보이지 않아 교체됨.

10건의 사례를 종합하면, 장애 유형 전반에 걸쳐 일관되게 지배적인 precursor 지표는 존재하지 않는다. 같은 XID 코드(예: NVLink 145/149) 내부에서도, 같은 노드의 재발 사례(gpu071의 두 차례 NVLink 장애)에서도 가장 강한 신호의 조합이 달라진다. 따라서 단일 지표에 의존하기보다 다수 지표를 동시에 평가하는 다중 신호 접근이 필요하다. 사전 검출률을 끌어올리기 위해 시계열 다변량 패턴과 교차 지표 상관관계 변화를 학습하는 ML 모델링이 진행 중이다.

4.2 체크포인트 저장과 복구: 스토리지 병목 분석

체크포인트 동작은 장애 시 손실되는 진행량과 복구에 필요한 시간을 함께 결정한다. 장애가 발생하면 마지막 체크포인트 이후의 학습이 손실되고, 재개 속도는 NFS가 체크포인트와 데이터셋을 얼마나 빠르게 제공하는지에 좌우된다.

본 절에서는 먼저 학습 I/O 프로파일과 체크포인트 주기를 정량화하고, 이어서 W&B run을 기반으로 장애 손실과 재시작 병목을 분석한다. 마지막으로 GPU VRAM에서 NFS 서버까지의 체크포인트 경로를 Prometheus 지표로 추적하여 비동기 체크포인트 파이프라인과 NFS RPC 병목을 규명한다.

4.2.1 학습 I/O 프로파일과 체크포인트 주기

GPU 사용률과 NFS I/O 패턴을 이용하면 시계열을 세 가지 학습 단계로 구분할 수 있다(Figure 5). 우선순위가 높은 순서대로 Save(체크포인트 저장; 클러스터 합산 NFS 쓰기 > 20 GB/s), Load(체크포인트 및 데이터 로드; GPU 사용률 $< 50\%$ 이면서 클러스터 합산 NFS 읽기 > 2 GB/s), 그리고 나머지 구간을 Training으로 분류하였다.

안정 학습 구간에서 GPU 사용률은 99% 이상을 유지하며, 약 2시간 13분 간격으로 짧은 하강이 관찰된다. 이 하강은 NFS 쓰기 스파이크와 동기화되어, 체크포인트 저장 단계에서 GPU 연산이 일시 정지됨을 보여준다(블로킹 단계와 비동기 단계의 구분은 Section 4.2.4에서 상세 분석).

체크포인트당 노드별 쓰기량은 약 20 GB으로 전 기간에 걸쳐 일정하며, 클러스터 합산 피크 쓰기 속도는 30초 샘플링 평균 기준으로 분석 초반 약 43 GB/s에서 11월 중순 이후 약 31 GB/s로 감소하였다. 체크포인트 저장은 수십 초 이내에 완료되므로, 실제 순간 피크는 이보다 높을 수 있다. 노드당 쓰기량이 일정한 상태에서 클러스터 합산 속도가 감소한 것은, 동일한 데이터량의 쓰기가 더 긴 시간에 걸쳐 분산되었을 가능성을 시사한다. 정확한 원인(NFS 서버측 부하, 스토리지 용량 증가에 따른 성능 변화 등)은 추가 조사가 필요하다.

세션 시작 시에는 체크포인트와 학습 데이터 로딩으로 인해 NFS 읽기가 클러스터 합산 약 230 GB/s까지 급증하며, 약 25분에 걸쳐 데이터가 페이지 캐시와 user buffer를 거쳐 GPU VRAM에 적재된다(데이터 경로별 정량 분석은 Section 4.2.4). 이후 학습 중에는 NFS 네트워크 트래픽이 사실상 0에 수렴하고, 모든 데이터 접근이 페이지 캐시에서 서비스된다.

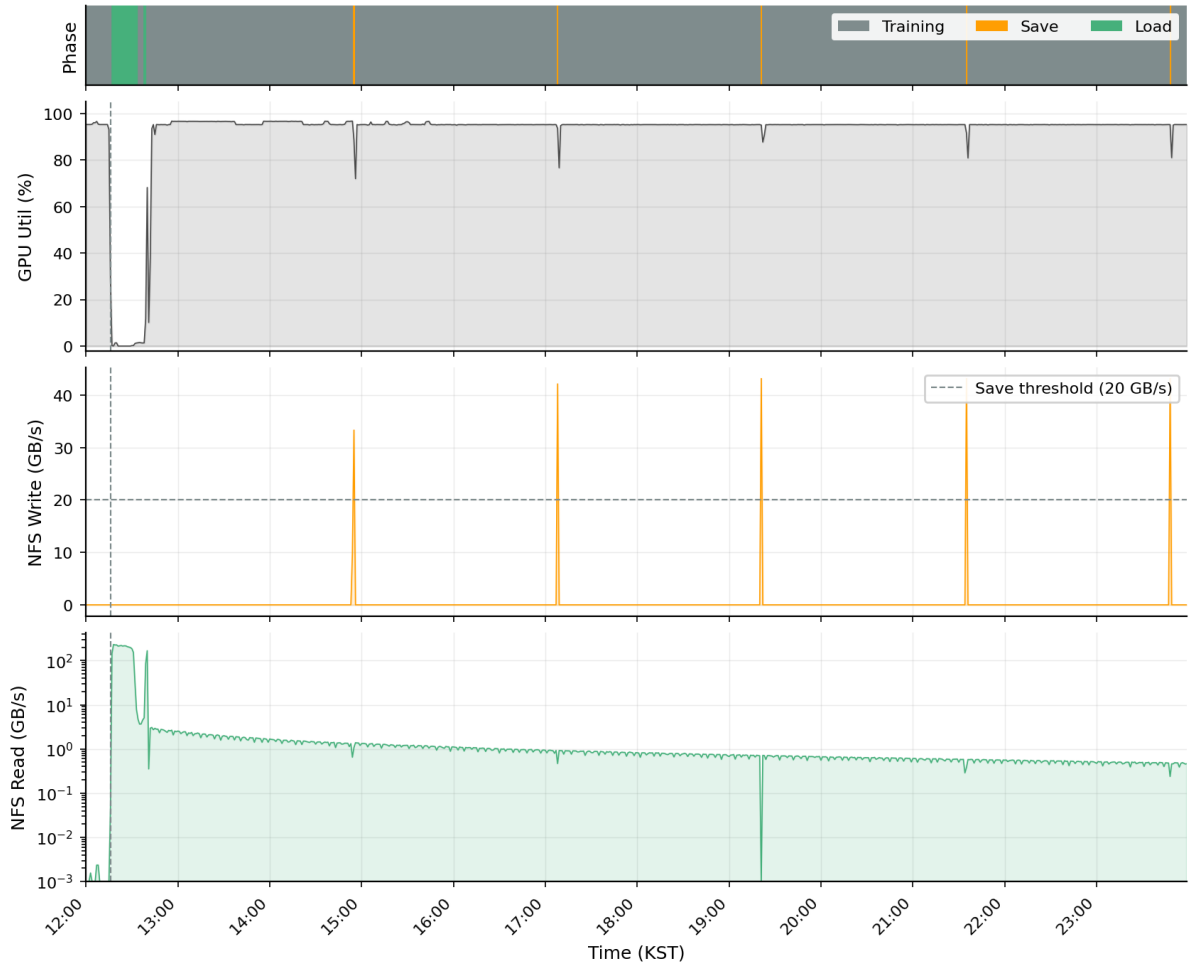

Figure 5: 학습 I/O 프로파일(2025-10-25 12:00~10-26 00:00 KST). 상단부터: 학습 단계 분류 바(회색=Training, 주황=Save, 초록=Load), 클러스터 평균 GPU 사용률(%), 클러스터 합산 NFS 쓰기 속도(GB/s, 회색 점선=20 GB/s Save 감지 임계값), 클러스터 합산 NFS 읽기 속도(GB/s, 로그 스케일). 회색 대시선은 세션 시작 시점(12:16 KST)이다.

55일간 NFS 쓰기 스파이크를 기반으로 총 523개의 체크포인트 저장 이벤트를 자동 감지하였다. 체크포인트는 VAST Data NFS 스토리지에 저장되며, 간격은 학습 구성에 따라 다르게 나타났다. Solar Open 기술 보고서 [3](Section 3.4)에 따르면, 학습은 세 단계로 진행된다. 사전학습(시퀀스 길이 4K, GPU당 배치 28, 글로벌 배치 13,440), 컨텍스트 확장 1단계(32K, GPU당 배치 3, 글로벌 배치 1,440), 컨텍스트 확장 2단계(100K, GPU당 배치 1, 글로벌 배치 480). 각 단계에서 측정된 체크포인트 간격은 다음과 같다.

- **4K 시퀀스 단계**(사전학습): 중앙값 2.23시간(133.5분), 표준편차 5.4분으로 안정적. 분석

기간의 대부분(466건)을 차지.

- **32K 시퀀스 단계**(컨텍스트 확장 1단계): 3.32시간(199분)으로 증가. 시퀀스 길이 확장에 따른 스텝 시간 증가가 원인으로 추정.
- **100K 시퀀스 단계**(컨텍스트 확장 2단계): 1.36시간(81.5분). 32K 대비 단축되었으며, 이론적 최적값에 근접하는 설정으로 최적화 효과는 후술.

NFS 스토리지 사용량은 분석 기간 동안 약 450 TB에서 963 TB로 약 510 TB 증가하였다(Figure 6). 총 용량 약 2,252 TB 대비 사용률은 약 20%에서 43%로 변화하였다.

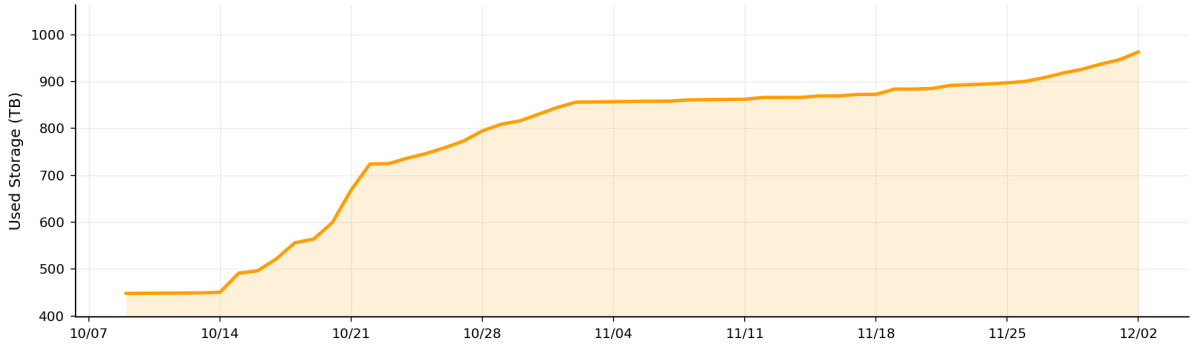

Figure 6: 55일간 NFS 스토리지 사용량 추이 (총 용량 약 2,252 TB). 체크포인트 파일 누적이 주요 증가 요인이다.

4.2.2 장애 비용과 체크포인트 주기

체크포인트 주기는 저장 오버헤드와 장애 시 손실 사이의 직접적인 trade-off를 결정한다. 저장 주기를 짧게 하면 손실은 줄지만 저장 오버헤드가 늘고, 주기를 길게 하면 오버헤드는 줄지만 장애 시 버려지는 학습량이 커진다. Young/Daly 모델 [19, 20]은 이 trade-off를 정량화하는 대표적 기준점이다.

다만 이 모델은 장애가 시간에 대해 균등하게(memoryless) 발생한다고 가정한다. 실측 데이터에서는 운영자가 체크포인트 직후에 종료하는 패턴(손실 0.05~0.1시간)과 예기치 않은 종료가 중후반에 발생하는 패턴(손실 2~3시간)이 혼재하여 이 가정에 부합하지 않으므로 [21], 아래의 이론값은 운영 목표를 잡기 위한 참조점으로 활용한다. 그럼에도 본 분석을 통해 운영상의 교훈을 얻을 수 있다: 체크포인트 저장 오버헤드($\delta = 18$–31.7초)가 작으므로 주기를 단축하는 비용이 낮고, 실제로 100K 시퀀스 단계에서 주기를 81.5분으로 줄인 결과 총 비용(1.82%)이 이론 최적(1.72%)에 근접하였다.

실측 손실 시간. W&B에서 비정상 종료된 23개 run[4]의 손실 시간을 측정하였다(Figure 7). 각 run에 대해 NFS 쓰기 스파이크 시각으로 직전 체크포인트를 식별하고, 종료 시점과의 차이를 산출하였다. 23건의 평균 손실 시간은 0.98시간, 총 손실 시간은 약 22.6시간이다.

[4] 이 23건은 W&B에 기록된 비정상 종료 run으로, Table 2의 17건(Prometheus/XID 기반 장애 이벤트)과는 집계 단위가 다르다: W&B run은 학습 프레임워크 관점의 세션 종료이고, Table 2의 장애 이벤트는 하드웨어 계층의 XID 에러 발생 건수이다.

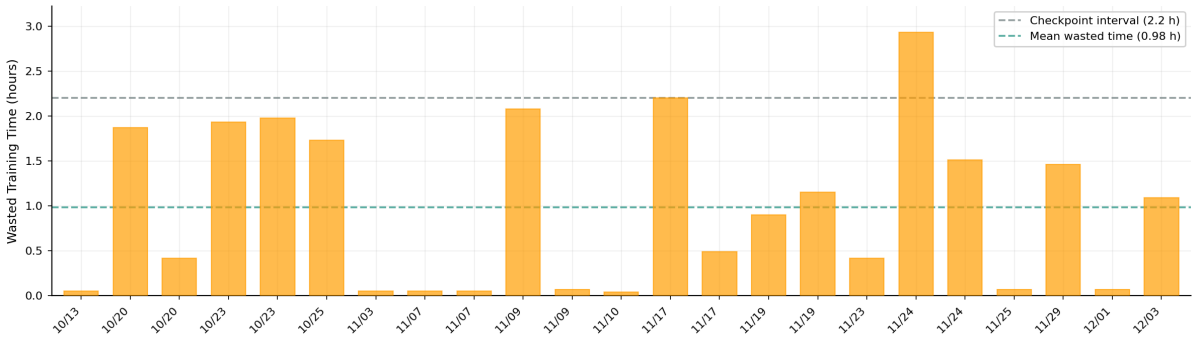

Figure 7: 비정상 종료된 W&B run별 손실된 학습 시간(23건). 회색 점선은 4K 단계의 체크포인트 간격(2.2시간), 청록색 점선은 평균 손실 시간(0.98시간).

주기 최적화. 체크포인트 주기에는 두 가지 비용이 상충한다. 주기를 줄이면 저장 오버헤드(δ/T)가 늘어나고, 주기를 늘리면 장애 시 평균 손실($T/2M$)이 커진다. Young/Daly 모델은 이 합을 최소화하는 주기 $T_{\text{opt}} = \sqrt{2\delta M}$을 제시한다.

저장 시간 δ는 Prometheus 30초 샘플링에서 직접 측정할 수 없으므로, NFS 쓰기 스파이크가 몇 개의 연속 샘플에 걸치는지를 관측하여 역산하였다(Table 10). MTBF(M)는 24개 run의 총 학습 시간 1,294시간을 비정상 종료 23건으로 나누어 56.2시간으로 추정하였다.

Table 10: 체크포인트 저장 소요 시간(δ)의 통계적 추정. $\bar{N}$은 NFS 쓰기 스파이크가 걸치는 평균 연속 30초 샘플 수.

학습 단계	이벤트 수	$\bar{N}$ (샘플 수)	δ (초)
4K 시퀀스	466	1.60	18.0
32K 시퀀스	36	2.06	31.7
100K 시퀀스	21	2.00	30.0

Table 11에 학습 단계별 비용을 비교하였다. 4K와 32K 단계에서는 실제 주기가 이론 최적의 약 3배로, 장애 손실(1.98~2.95%)이 비용을 지배하였다. 100K 단계에서는 주기를 81.5분으로 단축하여 이론 최적(58.1분)의 1.4배까지 근접하였고, 총 비용(1.82%)이 이론 최솟값(1.72%)과 0.10%p 차이였다. 저장 오버헤드는 모든 단계에서 0.6% 이하로, 주기 단축의 비용이 낮음을 보여준다.

Table 11: 학습 단계별 체크포인트 주기 비용 비교 ($M = 56.2$시간).

학습 단계	δ	실제 주기	T_{opt}	저장 오버헤드	총 비용
4K 시퀀스	18 s	133.5분	44.9분	0.22%	2.20%
32K 시퀀스	31.7 s	199.0분	59.7분	0.27%	3.22%
100K 시퀀스	**30 s**	**81.5분**	**58.1분**	**0.61%**	**1.82%**

4.2.3 재시작 로딩 시간과 대역폭 활용률

장애 후 복구 과정은, 학습을 다시 시작하기 위해 저장해 둔 체크포인트(checkpoint)를 읽어오는 것에서 시작된다. 따라서 재시작 로딩 시간은 복구 지연을 결정하는 1차 요인이라고 할 수 있다(Figure 8). 이 절에서 분석하는 재시작 로딩 시간은 세션 시작, Startup/Loading 단계를 거쳐 끝나는 시점의 구간 기준을 GPU 사용률이 낮은 상태(GPU utilization < 50%)에서 높은 NFS 읽기 트래픽(cluster-wide NFS read > 2 GB/s)이 관측되는 Startup/Loading 단계가 종료될 때까지의 시간으로 정의하였다. 실제 분석은 학습이 1시간 이상 지속된 세션 중에서 재시작 로딩 구간이 명확했던 20건을 대상으로 했다.

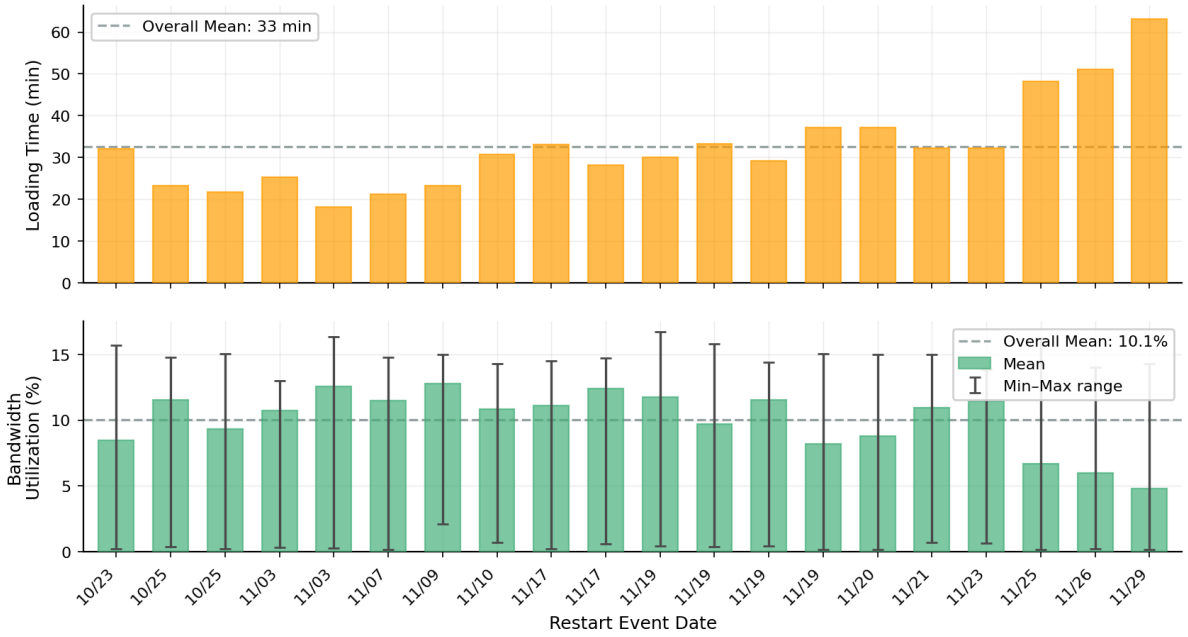

Figure 8: 세션 로딩 시간 및 NFS 대역폭 활용률(20건). 상단: 로딩 시간(분), 회색 점선=평균(33분). 하단: 이론적 최대 대역폭(1,500 GB/s) 대비 NFS 읽기 활용률. 막대=평균, 오차 막대=최솟값에서 최댓값까지(30초 간격 Prometheus 샘플 기준, 전체 평균 10.1%), 회색 점선=평균(10%).

평균은 33분, 중앙값은 31분이었고, 재시작 동안 관측된 평균 NFS 읽기 속도는 이론적 최대 대역폭의 약 10% 수준에 머물렀다. 만약 네트워크 링크 자체가 병목이었다면 활용률은 최대치에 근접해야 한다. 그러나 실제로는 상당한 유휴 대역폭이 남아 있었으므로, 재시작 지연의 원인을 네트워크 대역폭 부족으로 설명하기는 어렵다. 결과적으로 네트워크 업그레이드(200 Gbps → 400 Gbps RoCE)만으로는 재시작 시간을 실질적으로 단축하기 어렵다고 판단되었다. 네트워크 링크가 아니라면 또 다른 원인이 될 수 있는 것은 상위 소프트웨어 I/O 경로 때문일 수 있기 때문에 네트워크 위에서 데이터를 처리하는 소프트웨어 단계들을 좀더 살펴볼 필요가 있다. Table 12는 본 절에서 관측한 체크포인트 I/O의 주요 운영 특성을 요약한 것이다. 저장 측면에서는 체크포인트 빈도, 저장 지연, 노드별 쓰기량 및 클러스터 합산 쓰기 대역폭을 정리하고, 복구 측면에서는 장애로 인한 평균 손실 시간과 재시작 로딩 시간, 그리고 실제 NFS 대역폭 활용률을 함께 제시한다.

Table 12: 체크포인트 I/O 정량적 분석 요약(55일, 60노드 학습 / 63노드 B200 클러스터)

항목	메트릭	측정값
스토리지 사용량	시작 / 종료 사용률 변화	450 TB / 963 TB 약 20% → 43%
체크포인트	감지 이벤트 수 간격 (4K / 32K / 100K) 저장 소요 시간 δ (4K / 32K / 100K) 노드당 쓰기량 클러스터 합산 피크 쓰기 (30초 평균)	523개 2.23 / 3.32 / 1.36 h 18.0 / 31.7 / 30.0 s 약 20 GB 31~43 GB/s
학습 손실 시간 (23 runs)	평균 손실 시간 총 손실 시간	0.98 h 22.6 h
세션 로딩 (20건)	로딩 시간 평균 / 중앙값 평균 대역폭 활용률	33 min / 31 min 약 10%

주된 장애 유형(단일 노드 교체 대 전체 클러스터 재부팅)과 페이지 캐시 잔존 여부에서 편차가

존재하는지 확인하기 위해 체크포인트 저장 및 로딩 과정의 데이터 경로를 따라 `mmap()`, 페이지 캐시(page cache), NFS RPC 처리 계층을 분석했다.

4.2.4 체크포인트 데이터 경로: GPU에서 NFS까지

체크포인트 저장과 로딩은 모두 두 단계(Phase 1, Phase 2)로 동작하며, 각 단계가 사용하는 시스템 호출 경로가 다르다. Figure 9를 통해 체크포인트 데이터가 시스템 내부를 어떻게 이동하는지를 알 수 있다. 저장(save) 과정에서는 데이터가 GPU VRAM에서 CPU 메모리(/dev/shm)를 거쳐 NFS 스토리지로 전달되며, 로딩(load) 과정에서는 반대로 NFS 데이터가 먼저 커널 페이지 캐시에 적재된 뒤 GPU VRAM으로 복원된다.

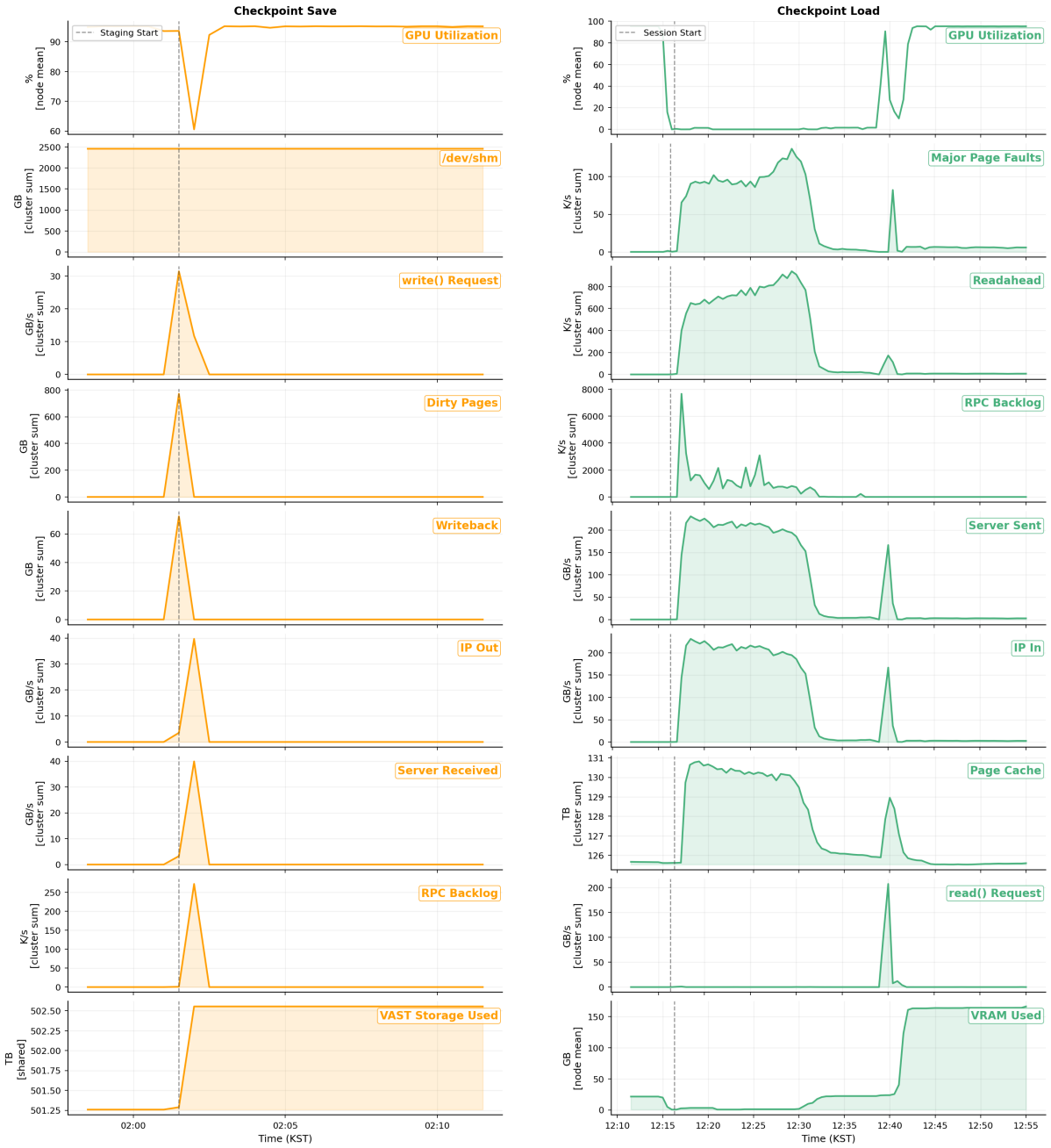

Figure 9: 체크포인트 I/O 데이터 경로 프로파일. 좌측은 저장(save) 경로, 우측은 로딩(load) 경로의 9개 계층 메트릭을 데이터 흐름 순서로 배열하였다. 점선은 staging 시작(좌)과 세션 시작(우) 시점이다. 로딩 과정에서는 네트워크 대역폭 자체보다 page cache와 RPC 처리 계층(RPC backlog)의 활동이 재시작 직후 크게 증가하는 것을 확인할 수 있다. 반면 실제 `read()` 요청은 훨씬 나중에 증가하여, 데이터가 먼저 페이지 캐시에 적재된 뒤 이후 GPU 로딩 단계에서 사용되었음을 보여준다.

좀더 구체적으로 살펴보면 다음과 같은 세부 과정을 거친다고 할 수 있다.

저장 (Figure 9 좌측). Phase 1 (블로킹)에서는 GPU VRAM의 모델/옵티마이저 상태가 CPU 측 staging buffer(본 클러스터에서는 `/dev/shm` tmpfs에 사전 할당된 영역)로 복사되며 학습이 일시 정지된다. Phase 2 (비동기)에서는 staging 데이터가 `write()` 시스템 호출을 통해 커널 페이지 캐시로 전달된 뒤, writeback 스레드가 NFS WRITE RPC로 서버에 전송하며, 이때 학습이 재개된다. 30초 간격의 모니터링에서는 두 Phase가 거의 동시에 발생하는 것처럼 보이지만, 실제로는 GPU 사용률 감소 이후 staging buffer 사용량 증가, `write()` 호출, 페이지 캐시 및

writeback 증가, NFS 쓰기 트래픽과 RPC backlog 증가를 거쳐 최종적으로 스토리지 사용량이 증가하는 순서가 일관되게 관측되었다.

`/dev/shm` RAM 임시공간 사용량(GB)은 학습 시작 시 중간 버퍼(staging buffer)가 사전 할당된 후 매 저장마다 같은 영역이 재사용되기 때문에 일정하게 유지된다. 60개 학습 노드는 두 그룹(48개 노드 약 48 GB, 12개 노드 약 9 GB)으로 나뉘며, 매 체크포인트마다 노드당 NFS 쓰기 양도 각각 약 26 GB와 약 2 GB로 같은 패턴을 보였다. 이는 큰 `/dev/shm` 영역이, 자신의 샤드를 NFS로 직접 전송하는 노드들의 중간 버퍼로 쓰이고 있음을 뜻한다. `write()` 시스템 호출 누적량과 NFS 서버 수신 누적량은 노드당 20.55 GB, 클러스터 합산 1,295 GB로 일치하여, 저장 데이터가 모두 `write()` 경로를 통과했음을 확인할 수 있다.

로딩 (Figure 9 우측). 체크포인트 로딩은 두 단계로 진행된다. 먼저 Phase 1(0~22분)에서는 체크포인트 데이터가 NFS 스토리지에서 페이지 캐시로 로딩된다. 이후 Phase 2(22~28분)에서는 페이지 캐시에 적재된 데이터가 GPU VRAM으로 복사되며, 이 시점부터 GPU 사용률이 증가하면서 학습이 재개된다.

Phase 1에서는 재시작 직후 1~3분 사이에 NFS 읽기 속도가 최대 약 230 GB/s까지 증가하며 RPC backlog도 함께 급증하였다(128 slot 포화). 이 과정에서 약 5 TB 규모의 데이터가 커널 페이지 캐시(page cache)에 적재되었다. 반면 같은 구간에서 `read()` 시스템 호출은 거의 발생하지 않았는데, 이는 데이터가 먼저 운영체제의 페이지 캐시에 로딩되고 있었음을 의미한다.

이후 Phase 2에서는 약 24분 시점부터 `read()` 요청이 급격히 증가하며 페이지 캐시에 저장된 데이터가 user buffer를 거쳐 GPU VRAM으로 복사되었다. 약 26분에는 VRAM 사용량이 25 GB에서 163 GB로 빠르게 증가하였고, GPU 사용률 역시 0%에서 94%로 회복되면서 학습이 재개되었다.

또한 일부 구간에서는 `read()` 요청량이 실제 NFS 읽기 증가량보다 더 크게 관측되었는데, 이는 이전에 페이지 캐시에 남아 있던 데이터가 cache hit로 재사용되었기 때문으로 해석된다. 본 클러스터는 RAM 여유(MemAvailable 약 95%)가 충분하므로, 이전 세션의 페이지 캐시가 재시작 이후에도 유지될 수 있다. 이러한 cache hit 여부는 Section 4.2.3에서 관측된 로딩 시간 편차의 한 원인으로 보인다.

4.2.5 NFS RPC 병목: 대역폭 패러독스의 해소

로딩이 200 Gbps RoCE 대역폭의 약 10%만 활용하는 좀더 명확하게 원인을 규명하기 위해, NFS 요청의 소요 시간을 두 구간으로 분해하였다. (1) RPC slot 대기: 클라이언트가 동시에 보낼 수 있는 RPC 수가 한도(본 클러스터 mountstats 관측값 128개)에 도달하면 slot이 풀릴 때까지 기다리는 시간. (2) 네트워크+서버 처리: slot 확보 후 요청을 전송하고 서버가 처리하여 응답을 반환할 때까지의 시간(네트워크 왕복 시간 포함). 두 구간은 Prometheus의 NFS operations 메트릭(`queue_time_seconds_total` / `response_time_seconds_total` / `requests_total`)으로부터 산출하였다(Figure 10).

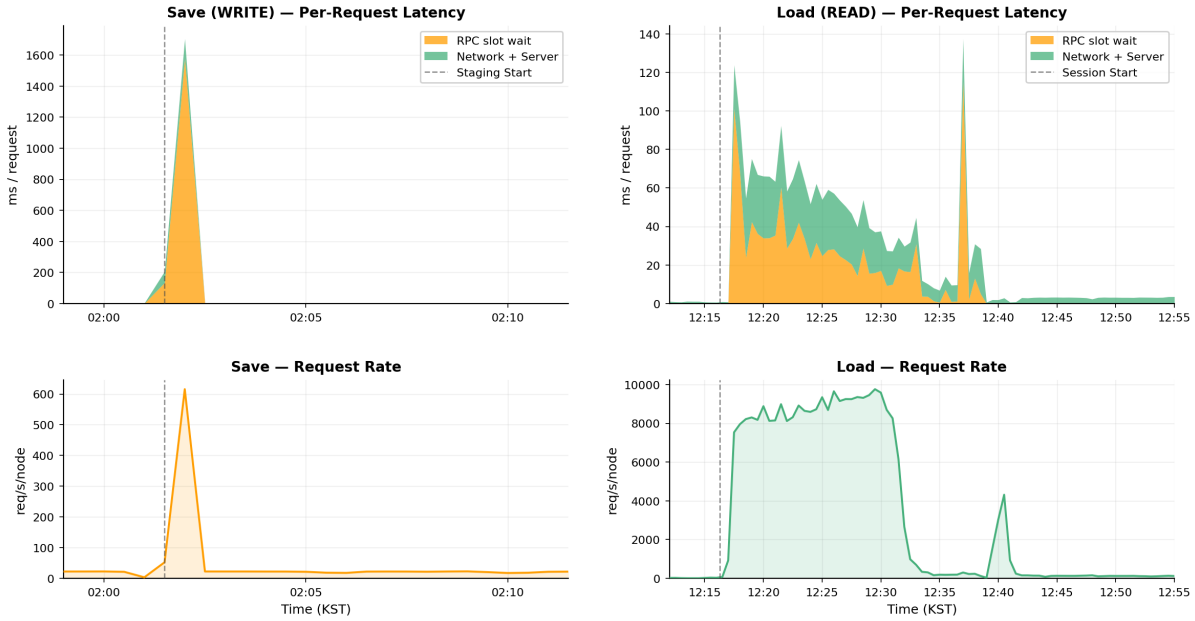

Figure 10: NFS RPC 요청당 지연 시간 분해. 상단: 요청당 지연을 RPC slot 대기(주황)와 네트워크 와 서버 처리(초록)로 분해. 하단: 노드당 요청 발생 속도. 저장(WRITE)은 30초 샘플 한 개에 집중된 쓰기 스파이크로, 해당 시점의 요청당 지연이 약 1,621 ms이다. 로딩(READ)은 요청당 지연이 낮지만(평균 59 ms; 시작 후 약 1.5분과 24분 부근에서 약 160 ms/req의 짧은 스파이크 두 차례 관측) 높은 요청 속도(8,000~9,000 req/s/node)가 약 23분간 지속된다.

Table 13: 저장과 로딩의 RPC 요청 패턴 및 요청당 지연 비교.

		저장 (WRITE)	**로딩 (READ)**
발생 패턴		30초 샘플 1개에 집중	약 23분간 지속
노드당 요청 속도		약 600 req/s (집중 시점)	평균 약 5,400, 피크 약 9,000 req/s
요청당 지연	**총 소요**	**1,621 ms**	**59 ms**
	RPC slot 대기	1,495 ms (92.2%)	31.5 ms (53.3%)
	네트워크+서버 처리	126 ms (7.8%)	27.3 ms (46.2%)

측정 결과(Table 13)에 따르면, 저장(WRITE) 요청 하나가 완료되기까지 평균 1.621초가 소요되었다. 그러나 이 중 실제 네트워크 전송 및 서버 처리 시간은 126 ms에 불과했고, 나머지 1.5초(92%)는 RPC slot이 비워질 때까지 대기하는 시간이었다. 즉 요청 대부분은 네트워크를 통해 데이터를 전송하는 데 사용된 것이 아니라, 클라이언트 측 RPC queue에서 순서를 기다리는 데 소비되었다. 이는 병목이 네트워크 대역폭이나 NFS 서버 처리 능력보다는 클라이언트 측 RPC slot 부족에 있음을 의미한다. 또한 WRITE 요청은 READ 요청과 달리 데이터 전송 이후 안정 저장(commit) 과정까지 포함하므로, 요청 하나당 서버 처리 시간이 더 길게 측정되었다(126 ms 대 27 ms). 같은 수의 RPC slot(128개)을 사용하더라도 WRITE 요청은 slot을 더 오랫동안 점유하게 되므로, 단위 시간당 처리 가능한 요청 수가 감소한다. 그 결과 저장 시점의 대량 쓰기 요청이 RPC queue를 빠르게 포화시키는 현상이 나타난 것으로 보인다.

대역폭 패러독스. 이론적 네트워크 대역폭은 노드당 25 GB/s(200 Gbps RoCE), 클러스터 합산 1,500 GB/s이다. 30초 샘플링 기준 대역폭 활용률은 저장 시 평균 1.4%(피크 2.7%), 로딩 시 평균 10.4%(피크 14.7%)이다. 200 Gbps를 400 Gbps로 증설해도 로딩 시간은 단축되지 않는다. 병목이 네트워크 대역폭이 아니라 그 위의 RPC 프로토콜 계층(slot 한도 128)에 있기 때문이다. 본 관측만으로 최적 RPC slot 수를 직접 도출할 수는 없지만, 반복적인 128 slot 포화 현상은 현재 설정이 대규모 60노드 학습 workload에 충분하지 않을 가능성을 보여준다. NFS 클라이언트의 RPC slot

한도 증대, 클라이언트 측 I/O 병합(readahead 최적화), 또는 서버 측 응답 시간 단축을 검토할 수 있다.

4.3 장애 패턴과 자동 복구

본 절은 다중 노드 학습의 장애 대응을 두 가지 보완적 관점에서 분석한다. Section 4.3.1은 어디에서 문제가 반복되는지를, Section 4.3.2은 장애 이후 얼마나 효과적으로 복구가 이루어지는지를 다룬다. 두 분석은 서로 연결되어 있다. 자동 재시도의 효과를 제한하는 예비 노드 부족 문제(Section 4.3.5)가 바로 노드 제외 분포의 직접적 결과이기 때문이다.

4.3.1 노드 제외 패턴

노드 제외는 클러스터 전반에 균등하게 나타나지 않고 일부 노드에 집중된다. 73일간 224회의 다중 노드 학습 작업 세션에서 같은 노드들이 반복적으로 60노드 학습에서 제외되었다. 클러스터는 63대의 GPU 노드로 구성되며, 60노드 작업 세션이 시작되면 Sokovan은 현재 자원이 비어 있는 노드 집합에서 필요한 노드를 선택한다. 운영자는 특정 노드에 단일 노드 작업 세션을 사전 할당함으로써 그 노드를 사실상 다중 노드 스케줄링에서 제외할 수 있다. 스페어 노드가 3대뿐이고 이들 역시 자주 점유되기 때문에, 대규모 학습의 실제 노드 구성은 거의 고정된다.

Figure 11는 224회의 다중 노드 학습 세션에서의 노드 제외 분포를 보여준다. 분포는 집중되어 있는데, 가장 많이 제외된 상위 3개 노드(gpu074, gpu119, gpu086)가 전체 제외의 50% 이상을 차지하며, 대부분의 노드는 5% 미만의 제외율을 보인다.

분석 기간 중 60노드 학습 세션이 가동된 시간 비율은 약 96.6%로 측정되었다. 최장 세션은 222.9시간(9.3일)에 달하였으며, 상위 5개 세션이 각각 3.6일 이상 연속 운영되었다.

노드 제외의 원인을 분석하기 위해, 60노드 학습 제외 시간 중 같은 노드에 단일 노드 세션이 할당된 기간이 차지하는 비율을 산출하였다(Figure 12, 13). 이 비율은 노드 제외가 운영자의 의도적 격리(단일 노드 세션 점유)인지, 63개 중 60개를 선택하는 스케줄러 특성상 자연스러운 미선택인지를 구분하는 지표로 활용할 수 있다.

상위 제외 노드 중 다수는 의도적 격리에 해당한다. gpu074(100%), gpu086(97%), gpu116(99.6%), gpu113(92%)는 제외 시간의 거의 전체가 단일 노드 점유와 겹쳐, 운영자가 학습 성능 저하(통신 지연, 학습 속도 감소 등)를 우려해 단일 노드 세션을 할당해 명시적으로 격리한 결과이다. gpu119(69%, 절대 겹침 793시간), gpu122(72%, 447시간)는 비율은 다소 낮지만 절대 겹침 시간이 매우 길어, 의도적 격리가 빈번한 노드로 분류된다.

반면 gpu085(제외 393시간 중 4%), gpu098(제외 20시간 중 2%)는 단일 노드 점유와 거의 겹치지 않아, 의도적 격리보다는 63개 중 60개를 무작위 선택하는 스케줄러 특성상 자연스럽게 미선택된 결과로 보인다.

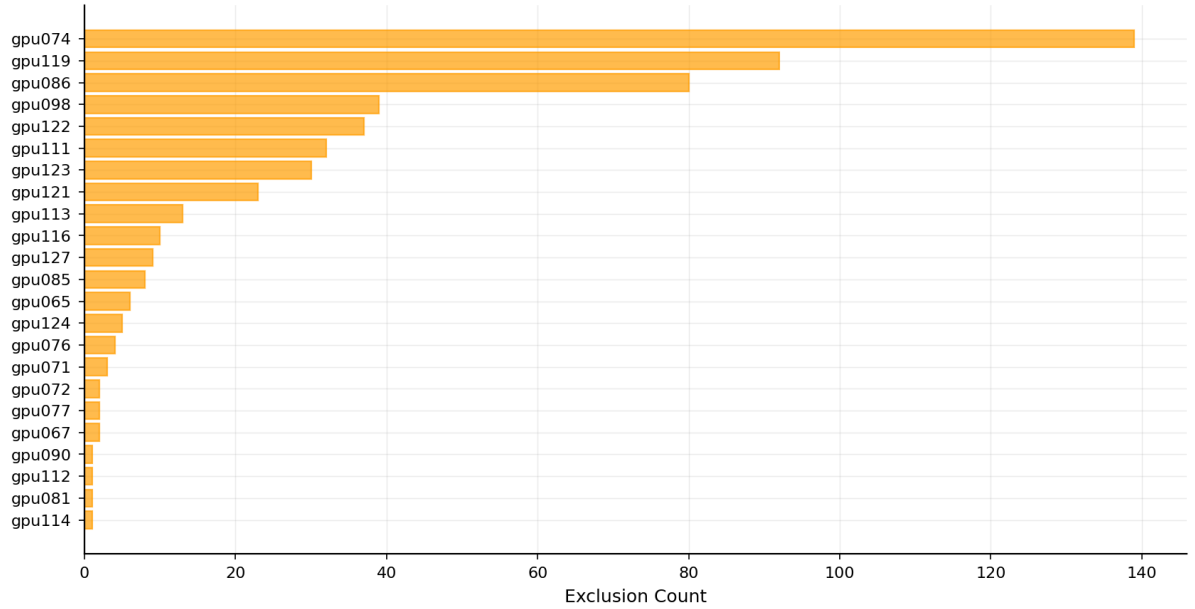

Figure 11: 73일간 224회 다중 노드 학습 세션에서의 노드 제외 빈도. 상위 3개 노드(gpu074, gpu119, gpu086)가 전체 제외의 50% 이상을 차지하는 집중된 분포를 보인다.

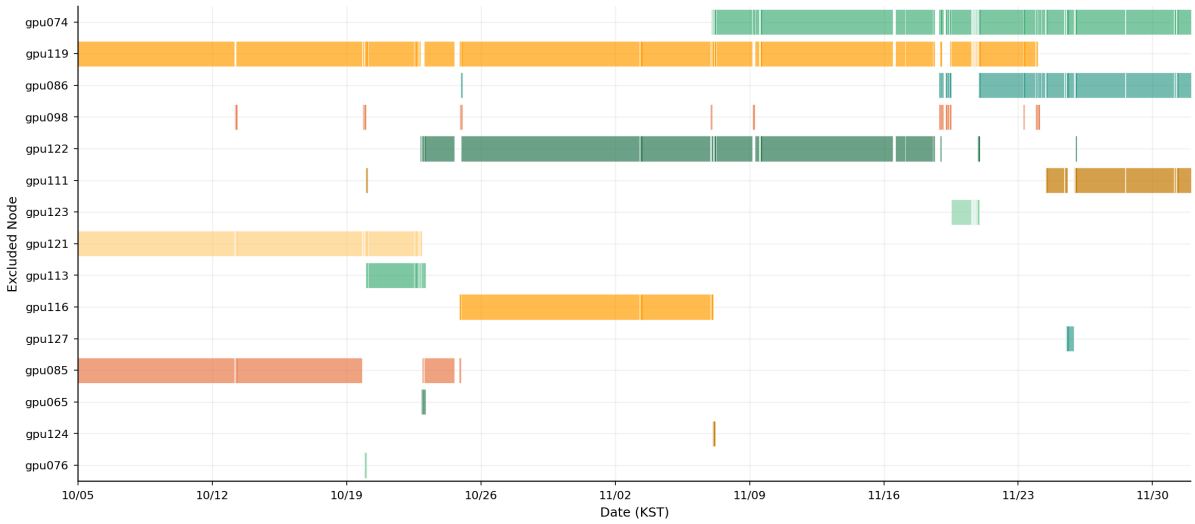

Figure 12: 상위 15개 노드의 60노드 학습 세션 제외 타임라인. 각 바는 해당 노드가 제외된 학습 세션의 기간을 나타낸다. Figure 13과 비교하면 대부분의 제외가 의도적 단일 노드 점유와 시간상 겹침을 확인할 수 있다.

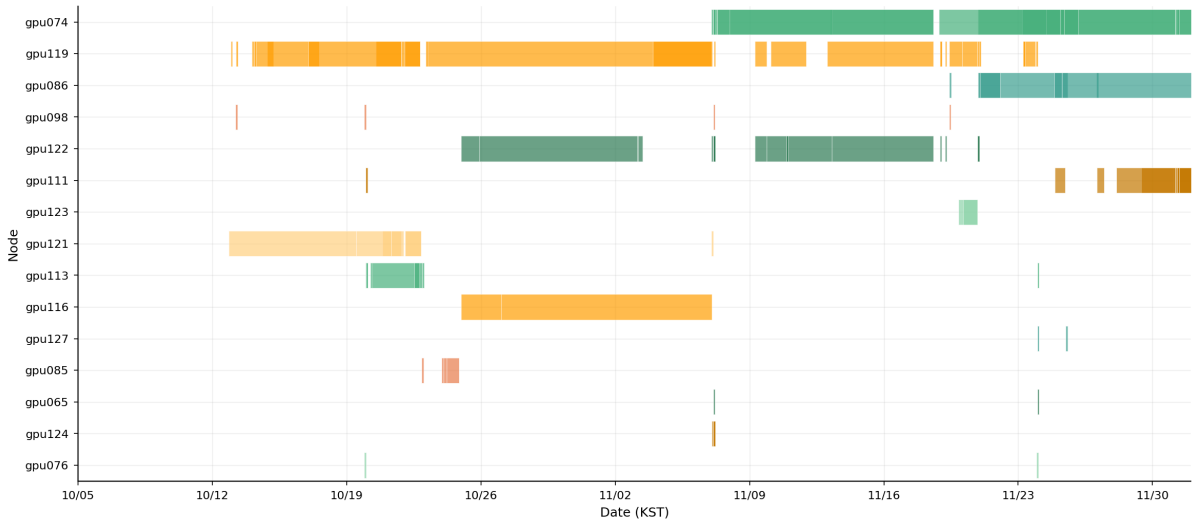

Figure 13: 동일 15개 노드에 대한 단일 노드 세션 점유 타임라인. 운영자가 문제 노드에 단일 노드 세션을 할당하여 60노드 학습 스케줄링에서 의도적으로 제외하는 운영 패턴이 관찰된다. gpu074(100%), gpu086(97%), gpu116(99.6%) 등은 60노드 학습 제외 시간의 거의 전체가 단일 노드 점유와 겹친다.

요약하면, 상위 3개 노드(gpu074, gpu119, gpu086)가 전체 제외의 50% 이상을 차지하는 집중된 분포를 보이며, 이들 중 다수가 운영자의 의도적 제외(단일 노드 세션 점유로 명시적 격리)에 해당한다. 일부 노드(gpu085 등)는 단일 노드 점유와 겹치지 않아 스케줄러의 무작위 미선택으로 자연스럽게 제외된 것으로 보인다.

4.3.2 자동 재시도 구성 및 체인 개요

노드 제외 분석이 어떤 노드에서 장애가 반복되는지를 보여주었다면, 자동 재시도 분석은 장애 이후 얼마나 빠르고 효과적으로 복구되는지를 다룬다. Backend.AI FastTrack은 작업 수준에서 구성 가능한 자동 재시도 전략을 제공한다(Figure 14). 운영자는 세 가지 핵심 파라미터를 설정한다: (1) 장애 시 자동 재시도 여부, (2) 최대 재시도 횟수, (3) 재시도 지연 시간. 본 관측 기간에는 재시도가 활성화되었고, 재시도 간격은 약 10분으로 설정되었다. 다음 분석은 이 설정의 운영 효과를 정량적으로 평가한다.

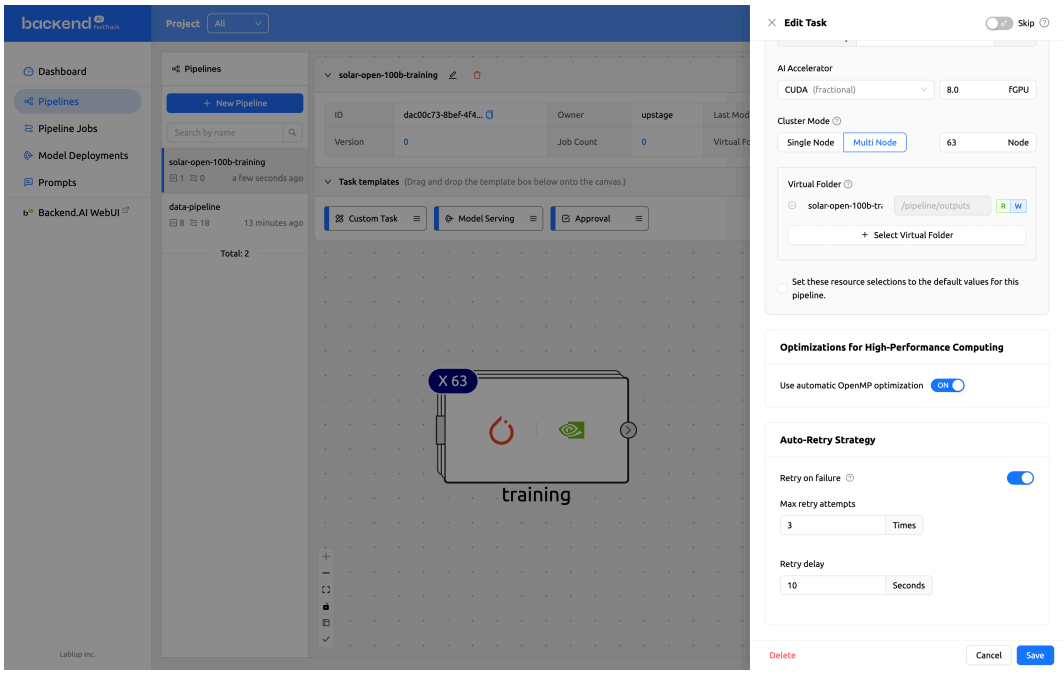

Figure 14: Backend.AI FastTrack의 자동 재시도 구성 화면. 운영자는 작업별로 장애 시 재시도 여부, 최대 재시도 횟수, 재시도 지연 시간을 설정할 수 있다. 이외에도 GPU 수, 노드 수, 스토리지 마운트 등 자원 구성이 가능하다.

73일간의 운영 로그에서 동일 작업명으로 연속 실행된 작업 세션들을 자동 재시도 "체인"으로 식별하였다. 12개의 자동 재시도 체인(총 73회 시도, 재시도 61회)이 식별되었으며, 그 결과는 Table 14와 같다.

Table 14: 12개 자동 재시도 체인의 결과 분류.

결과	설명	건수
SUCCESS	재시도 후 학습 도달	4
FAIL (학습 후)	첫 시도 학습 진행, 이후 재시도 전부 실패	3
FAIL (시작 실패)	처음부터 시작 실패, 재시도 전부 실패	5

Figure 15는 전체 작업 세션의 시간순 타임라인이다.

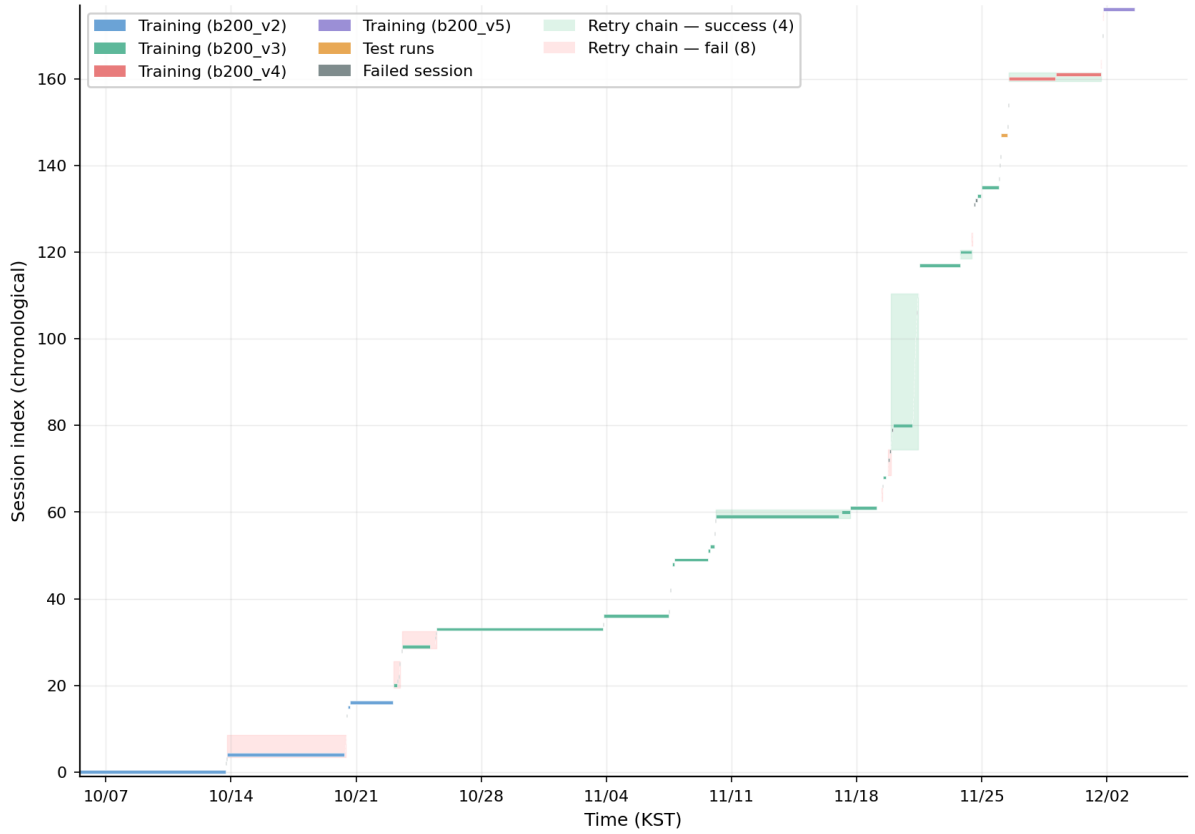

Figure 15: 73일간 작업 세션 타임라인. x축은 시각, y축은 작업 세션의 시작 시각 순서이다. 각 바는 하나의 60노드 작업 세션이며, 색상은 학습 버전(b200_v2–v5)을 구분한다. 배경 하이라이트는 자동 재시도 체인을 나타내며, 초록색은 재시도 후 학습 재개에 성공한 체인(4개), 분홍색은 실패한 체인(8개)이다.

4.3.3 재시도 간격의 예측 가능성

FastTrack의 재시도 지연 시간 설정의 직접적 효과는 재시도 간격의 일관성에서 나타난다(Figure 16). 자동 재시도의 작업 세션 간 간격은 중앙값 11분, IQR 10–11분으로, 설정된 재시도 지연 시간(10분)에 작업 세션 종료·재시작 오버헤드가 더해진 결과이다. 반면 수동 재시작은 중앙값이 2분이지만 범위가 0–430분으로 예측하기 어렵다. 이 차이는 야간이나 주말처럼 즉각 대응이 어려운 시간대에 더 두드러진다: 수동 재시작은 대응 가능한 시점까지 지연될 수 있지만, 자동 재시도는 시간대와 무관하게 약 10–11분의 일정한 간격으로 복구를 시도한다.

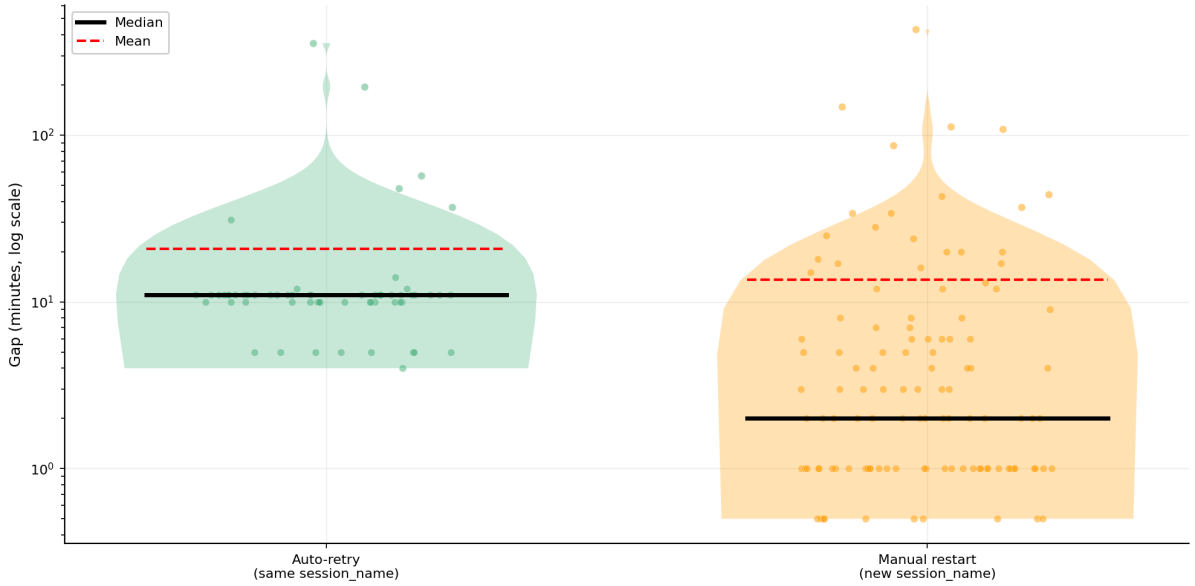

Figure 16: 자동 재시도와 수동 재시작의 작업 세션 간 간격 분포 비교. 자동 재시도는 중앙값 11분, IQR 10–11분으로 FastTrack의 재시도 지연 시간 설정에 대응한다. 수동 재시작은 중앙값이 2분이나 범위가 0–430분으로, 대응 시점에 따라 예측하기 어렵다.

4.3.4 성공률 비교 및 다운타임 단축

체인 성공률(동일 작업명의 연속 재시도 중 최소 한 번이라도 학습에 도달한 비율)은 33.3%(12개 체인 중 4개 성공)였다. 비교를 위해, 수동으로 시작된 개별 작업 세션이 학습에 도달한 비율은 12.5%(104개 작업 세션 중 13개 성공)로, 자동 재시도 체인이 약 2.7배 높았다. 체인은 복수 시도의 기회를 가지므로 단일 작업 세션보다 유리하지만, 이 차이는 자동 재시도의 구조적 이점을 반영한다. 성공한 4건 중 3건은 1회 재시도만으로 복구되었다. 이 중 1건은 XID 94(ECC 오류)로, Table 3에서 `RESTART_APP`(작업 세션 재시작)으로 해결 가능한 유형에 해당하며, 자동 재시도가 운영자 개입 없이 복구에 성공한 사례이다.

성공률 향상은 다운타임 단축으로 직결된다. 22개 학습 작업 세션 간의 21건 복구 에피소드를 분석한 결과(Figure 17), 자동 재시도를 통해 학습이 재개된 4건의 다운타임 중앙값은 1.9시간인 반면, 수동 복구 17건의 중앙값은 3.3시간으로 약 1.8배의 차이를 보였다. 수동 복구의 분산(0–53시간)이 큰 것은 야간이나 주말 등 즉각 대응이 어려운 시간대에 장애가 발생한 경우가 포함되어 있기 때문이다.

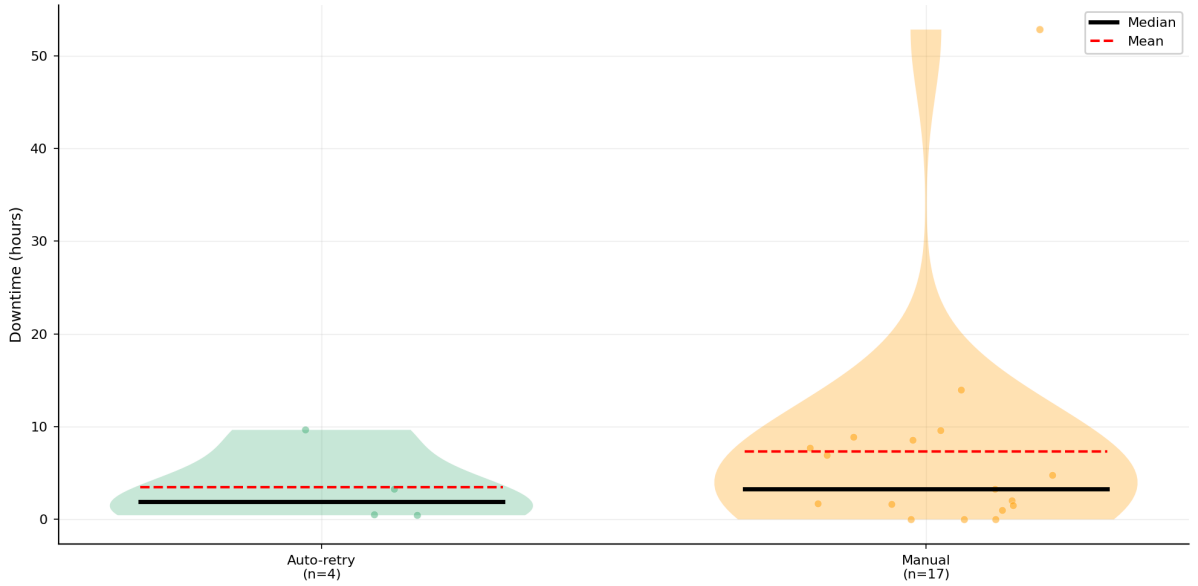

Figure 17: 자동 재시도와 수동 복구의 다운타임 비교(21건 학습 간 복구 에피소드). 자동 재시도로 복구된 4건(초록)은 중앙값 1.9시간, 수동 복구 17건(주황)은 중앙값 3.3시간. 검은 실선 = 중앙값, 빨간 점선 = 평균.

4.3.5 한계 및 향후 개선 방향

12개 체인 중 8개(67%)가 최종적으로 실패하였다. 대부분의 실패는 소프트웨어/네트워크 수준의 문제(NCCL 통신 오류 등)로, 단순 재시작만으로는 해결되지 않는 유형이었다.

또한 자동 재시도 복구 에피소드 중 다운타임이 긴 사례(9.65시간, 3.25시간)는 자동 재시도 메커니즘 자체의 한계가 아닌 인프라 수준의 문제에 기인한다. 하드웨어 교체 후 GPU 라이선스가 갱신되지 않아 가용 자원 풀에 편입되지 못한 노드가 있었으며, 이로 인해 60노드 요구 조건을 충족하지 못해 재시도가 수 시간 동안 지연되었다. 해당 문제는 이후 플로팅 라이선스 방식으로 전환하여 노드 교체 시에도 동일 문제가 재발하지 않도록 해결하였다.

실패한 재시도의 비용은 약 35 GPU-hours(전체 학습 시간의 약 2.7%)이다. 특히 한 체인은 25.4시간 학습에 성공한 뒤 30회 연속으로 실패하며 끝났는데, 문제를 해결하지 않은 채 같은 조건으로 반복 재시도하면 GPU-hours만 소모할 뿐임을 보여준다.

위 분석을 바탕으로 다음 개선 방향을 제시한다.

- **지수 백오프(exponential backoff)**: 현재 고정 간격(10분) 대신 10분 → 20분 → 40분으로 간격을 점진 증가시켜, 후반 재시도의 자원 소모를 줄이면서 일시적 장애에 대한 초기 복구 속도는 유지할 수 있다.
- **XID 기반 분기 대응**: Table 3의 해결 유형에 따라 재시도 전략을 차별화한다. `RESTART_APP` 유형(XID 31, 43, 94)은 즉시 재시도, `RESET_GPU` 유형(XID 119, 145, 149)은 GPU 리셋 후 재시도, `CONTACT_SUPPORT` 유형(XID 79)은 재시도를 중단하고 운영자에게 알림을 발송하는 방식이다.
- **우선순위 기반 작업 세션 선점**: 현재 여분 노드가 3대뿐이어서(Section 4.3.1), 단일 노드 작업 세션이 점유 중이면 60노드 갱 스케줄링 요구 조건을 충족하지 못해 자동 재시도가 지연될 수 있다. 멀티노드 학습에 더 높은 우선순위를 부여하여, 재시도 시 낮은 우선순위의 단일 노드 작업 세션을 자동으로 선점하거나, 예비 노드 풀을 확대하는 방안을 통해 가용 상태를 개선할 수 있다.

5 한계

본 보고서는 종단 간 학습 효율성보다 인프라 수준의 오류와 복수 시스템을 평가했다. Section 4.2의 체크포인트 I/O 분석은 체크포인트 오버헤드(δ = 18–31.7초), 장애당 손실 시간(평균 0.98시간), 재시작 로딩 시간(평균 33분), RPC 병목 메커니즘을 정량화함으로써 이 공백을 일부 보완하고자 했다. 그러나 이러한 효과가 전체 학습 효율로 어떻게 이어지는지를 평가하려면 학습 프레임워크 내부 계측이 필요하며, 이는 본 연구의 범위를 벗어난다고 할 수 있다.

전조 분석(Section 4.1) 역시 사후적 평가에 해당한다. 프로덕션 환경에 실시간 적용하려면 false positive 분포와 알림이 초래하는 운영 부담을 별도로 검증해야 한다. 또한 10건의 장애 사례만으로는 통계적 검정력이 제한적이다.

마지막으로 자동 재시도 분석은 12개 체인(73회 시도)만을 다룬다. 이 결과는 자동 복구의 구조적 특성을 드러내기에는 충분하지만, 표본 규모에는 여전히 한계가 있다.

6 관련 연구

GPU 클러스터 스케줄링. GPU 클러스터 스케줄링 연구는 서비스 달성량 기반 우선순위(Tiresias [22])와 관찰 기반 타임 슬라이싱(Gandiva [23])에서 시작해, Pollux [24], Sia [25] 같은 goodput 적응형 시스템으로 발전해 왔다. 최근에는 공정성 [26], 네트워크 토폴로지 [27], 지리 분산 스케줄링 [28], 클라우드 자원 관리 [29]로 주제가 확장되고 있다. 이들 시스템은 워크로드에 적응적인 제어로 JCT나 goodput을 최적화한다. 반면 본 보고서가 다루는 Sokovan은 힌트 기반 폴링으로 스케줄링 지연을 예측 가능하게 유지하는 데 초점을 둔다.

분산 학습 시스템. 분산 학습 연구는 그동안 모델 병렬화(Megatron-LM [30, 31]), 파이프라인 병렬화(GPipe [32]), 메모리 효율적 샤딩(DeepSpeed ZeRO [33], PyTorch FSDP [34]), 자동 전략 선택(Alpa [35]), 10,000+ GPU 규모 확장(MegaScale [5]) 등 병렬화 전략 자체에 초점을 맞춰 왔다. 본 보고서가 다루는 Backend.AI는 그 아래에서 세션 생명주기와 자원 할당을 관리한다.

장애 내성 및 체크포인팅. 체크포인트 기반 복구는 장기 실행 학습의 핵심이다. 관련 연구는 저장 빈도 최적화(CheckFreq [36]), 예측적 체크포인팅 [37], 인메모리 체크포인트(Gemini [38], ByteCheckpoint [39]), 체크포인트 없는 복원력(Oobleck [40], Bamboo [41])에 이르기까지 폭넓게 다뤄져 왔다. 종단 간 장애 내성 시스템으로는 탄력적 스팟 VM 학습(Varuna [42]), 빠른 장애 감지(TRANSOM [43]), MoE 특화 복원력(Lazarus [44])이 있다. 이들 연구는 주로 체크포인팅 자체나 학습 파이프라인 내부의 장애 내성을 최적화한다. 본 보고서의 자동 재시도 메커니즘은 오케스트레이션 계층에서 작업 세션 재시작과 자원 재할당을 수행하며, 체크포인트의 생성과 복원은 학습 프레임워크에 위임한다. 이때 발생하는 재시작 로딩 시간은 Section 4.2.3에서 정량화한다.

7 결론

본 보고서에서는 63개 노드 B200 프로덕션 GPU 클러스터에서 수집한 55일간의 모니터링 데이터와 73일간의 운영 로그를 바탕으로 장애 전조 탐지, 체크포인트 I/O, 자동 복구를 분석하였다.

7.1 주요 발견 요약

먼저, Section 4에서 분석한 결과들은 조직 간 공동 운영 환경(Section 3.5)에서 도출된 것이라는 대전제가 필요하다. 조직 경계를 가로지르는 공유 지표 파이프라인이 없었다면, 여기에 정리한 운영 규모의 현상은 어느 한 팀도 직접 관찰하기 어려웠을 것이다.

장애는 대규모 학습의 구조적 특성. 클러스터 규모와 장애 빈도 간의 수학적 관계, 그리고 운영 현장에서 확인된 증거(16K GPU에서 54일간 419회 중단과 63노드 클러스터에서의 집중된 노드 제외 패턴)는 수 시간마다 발생하는 하드웨어 장애가 대규모 학습의 근본적 특성임을 보여준다.

작업 세션 수준 추상화를 요구하는 학습 워크로드. 컨테이너 오케스트레이션은 무상태의 단기 프로세스를 가정한다. 학습 워크로드는 상태를 가지며 장기 실행되므로, 재시작이 아닌 체크포인트 진행 상황을 추적하고 재개를 가능하게 하는 추상화가 필요하다. Backend.AI의 작업 세션 추상화는 학습 진행을 컨테이너 생명주기로부터 분리한다(Section 3.2).

GPU 스케줄링과 스토리지의 공동 설계 필요성 CPU 중심 자원 모델은 GPU 토폴로지나 전부 또는 전무(all-or-nothing) 할당 요구사항을 포착하지 못한다. 조직 간 공동 운영 환경에서 관찰된 스토리지 I/O 병목(Section 3.5)이 그 한계를 잘 보여준다. 이에 상응하는 스토리지 대역폭 없이 GPU 용량만 프로비저닝하면 운영 규모에서만 나타나는 성능 절벽이 발생한다. 체크포인트 I/O의 전체 계층 프로파일링(Section 4.2.5)은 200 Gbps RoCE 대역폭의 1.4–10.4%만 활용되는 "대역폭 패러독스"의 원인이 NFS RPC 프로토콜 계층의 128 slot 포화에 있음을 밝혔으며, 따라서 네트워크 대역폭 증설(200 Gbps→400 Gbps)보다 RPC slot 한도 증대나 서버 측 응답 시간 단축이 더 중요하다는 점을 보여준다. Sokovan은 GPU 우선 할당과 갱 스케줄링으로 GPU 측 해법을 제공한다.

7.2 향후 연구

본 연구의 여러 한계는 관측 기간 동안 충분한 계측이 갖춰지지 않았기 때문에 발생했다. 향후 학습 운영 기간에서는 다음과 같은 계측 확장이 필요하다.

학습 효율성 지표. 현재 분석은 체크포인트 간격, 재시작 시간, 장애율 같은 인프라 수준 수치를 측정하지만, 학습 프레임워크 내부 지표는 포함하지 못한다. 반복당 처리량(tokens/sec)을 기록하면 MFU를 산출할 수 있고, 장애, 재시작, 노드 교체 같은 인프라 이벤트가 실제 학습 진행에 미치는 영향을 직접 정량화할 수 있다. 이는 학습 시작 전 프레임워크 로거 설정만으로도 수집 가능하다.

RPC 병목 최적화 Section 4.2.5에서 재시작 로딩의 병목이 네트워크 대역폭이 아닌 NFS RPC 프로토콜 계층임을 규명하였다. 저장 시 WRITE 요청은 약 92%의 시간을 RPC slot 대기에 소비하며(요청당 약 1,495 ms), 로딩 시에는 READ 요청이 약 23분간 8,000~9,000 req/s/node로 지속되며 시작 직후와 24분 부근에서 약 160 ms/req의 짧은 응답 지연 스파이크를 보인다. NFS 클라이언트의 RPC slot 한도 증대, 서버 측 응답 시간 단축, 또는 체크포인트 전용 대용량 I/O 경로 도입이 향후 개선 방향이다.

전조 기반 예측적 장애 관리 Section 4.1의 통계 기반 다중 신호 탐지에서는 사전 검출 비율이 낮은 점이 주된 한계이다. 본 클러스터의 장애 다수에서 신호가 점진적으로 악화되지 않고 XID 시점에 갑작스럽게 나타나는 특성이 사전 검출을 어렵게 한다. 이를 보완하기 위해 다변량 시계열 패턴과 지표 간 상관관계의 변화를 함께 학습하는 ML 모델링을 후속 작업으로 진행하고 있다. 운영에 적용하려면 실시간 추론에서도 false positive 수준이 유지되는지 다시 검증해야 하고, 탐지 결과가 자동 재시도 의사결정에 반영되도록 하는 통합 설계도 함께 마련해야 한다.

지능형 자원 조정 및 로그 분석 향후 연구로, OOM 발생 시 자원 자동 조정 [45], 자원 과할당 감지, 이기종 로그 통합 분석을 결합한 자동 운영 설계를 검토할 필요가 있다.

FP8 및 축소 정밀도 학습. FP8 및 MXFP8과 같은 축소 정밀도 형식은 처리량 향상을 약속하지만, 소프트웨어 스택과 수치 안정성 양쪽에서 장애 모드를 도입한다. NVIDIA cuDNN 릴리스에는 다수의 FP8 관련 결함이 문서화되어 있으며 [46], Fishman et al. [47]은 약 2,000억 토큰 이후 치명적 불안정성을, Lee et al. [48]은 현재 FP8 방법의 범용적 견고성 부족을 보였다. Solar Open 프로젝트는 동일 B200 클러스터에서 FP8 + bfloat16 혼합 정밀도를 채택했다 [3]. 인프라 관점에서, 학습 발산이 cuDNN 버그, 수치 한계, 하드웨어 결함 중 어디에서 비롯되었는지 구별하는 자동화된 장애 원인 판별 메커니즘 개발이 미해결 과제로 남아 있다.

감사의 글

본 연구는 과학기술정보통신부(MSIT)가 주관하고 정보통신산업진흥원(NIPA)이 지원하는 독자 AI 파운데이션 모델 프로젝트(GPU 트랙)의 일환으로 수행되었다(PJT-25-080041).

Section 3.5의 스토리지 I/O 디버깅 사례는 여러 조직의 협업 덕분에 가능했다. 클라우드 인프라 및 운영 지원을 제공한 SKT, 하드웨어 전문성과 드라이버 수준 진단을 제공한 NVIDIA Korea, 스토리지 시스템 분석과 구성 최적화를 수행한 VAST Data, 워크로드 특성을 공유하고 공동 디버깅 세션에 참여한 Upstage에 감사드린다.

또한 Backend.AI가 구축된 기반이 되는 도구 및 프레임워크(PyTorch, NCCL, Linux 커널 네트워킹 계층 포함)를 제공한 오픈소스 커뮤니티에도 감사를 표한다.

데이터 가용 상태

본 보고서에서 분석한 운영 데이터(Prometheus 시계열 지표, 노드 제외 로그, 자동 재시도 기록, GPU 사용률 추적 포함)는 독자 AI 파운데이션 모델 프로젝트하에서 운영되는 프로덕션 클러스터에서 수집되었으며, 독점적 워크로드 정보를 포함한다. 따라서 계약 및 기밀 유지 제약 때문에 공개할 수 없다. Backend.AI 플랫폼 자체는 https://github.com/lablup/backend.ai에서 오픈소스 소프트웨어로 제공된다. `all-smi` 모니터링 도구는 https://github.com/lablup/all-smi에서 제공된다. 본 보고서에 제시된 분석을 재현하기에 충분한 집계 통계는 본문의 표와 그림으로 제공된다.

References

[1] Aaron Grattafiori, Abhimanyu Dubey, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[2] Apostolos Kokolis, Michael Kuchnik, John Hoffman, Adithya Kumar, Parth Malani, Faye Ma, Zachary DeVito, Shubho Sengupta, Kalyan Saladi, and Carole-Jean Wu. Revisiting reliability in large-scale machine learning research clusters. In *Proceedings of the 2025 IEEE International Symposium on High-Performance Computer Architecture* (*HPCA*). IEEE, 2025.

[3] Upstage Solar Team. Solar open technical report. Technical report, Upstage, January 2026. arXiv:2601.07022. 102B bilingual MoE (12B active) trained on 20T tokens. Also available at https://huggingface.co/upstage/Solar-Open-100B.

[4] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. Analysis of Large-Scale Multi-Tenant GPU clusters for DNN training workloads. In *2019 USENIX Annual Technical Conference* (*USENIX ATC 19*), pages 947–960, Renton, WA, July 2019. USENIX Association.

[5] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, et al. Megascale: Scaling large language model training to more than 10,000 gpus. In *Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation* (*NSDI*). USENIX Association, 2024.

[6] Alexander Erben and Ege Erdil. Hardware failures won't limit AI scaling. Epoch AI, 2024.

[7] Yangtao Deng, Xiang Shi, Zhuo Jiang, Xingjian Zhang, Lei Zhang, Zhang Zhang, Bo Li, Zuquan Song, Hang Zhu, Gaohong Liu, Fuliang Li, Shuguang Wang, Haibin Lin, Jianxi Ye, and Minlan Yu. Minder: Faulty machine detection for large-scale distributed model training. In *Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation* (*NSDI*), pages 505–521. USENIX Association, 2025.

[8] Tianyuan Wu, Wei Wang, Yinghao Yu, Siran Yang, Wenchao Yang, Qinkai Duan, Guodong Yang, Jiamang Wang, Lin Qu, and Liping Zhang. FALCON: Pinpointing and mitigating stragglers for large-scale hybrid-parallel training. *arXiv preprint arXiv:2410.12588*, 2024.

[9] Jinkun Lin, Ziheng Jiang, Zuquan Song, Sida Zhao, Menghan Yu, Zhanghan Wang, Chenyuan Wang, Zuocheng Shi, Xiang Shi, Wei Jia, Zherui Liu, Shuguang Wang, Haibin Lin, Xin Liu, Aurojit Panda, and Jinyang Li. Understanding stragglers in large model training using what-if analysis. *arXiv preprint arXiv:2505.05713*, 2025.

[10] NVIDIA Corporation. Analyzing XID errors — GPU deployment and management documentation. https://docs.nvidia.com/deploy/xid-errors/analyzing-xid-catalog.html, 2026. Accessed: 2026-02-15.

[11] NVIDIA Corporation. NVIDIA Blackwell architecture technical brief: Powering the new era of generative AI and accelerated computing. Technical brief, NVIDIA Corporation, 2024. Version 1.1. Per Blackwell GPU: up to 192 GB HBM3e, up to 8 TB/s. Accessed: 2026-04-21.

[12] NVIDIA Corporation. DGX SuperPOD: Next generation scalable infrastructure for AI leadership reference architecture featuring DGX B200. https://docs.nvidia.com/dgx-superpod/reference-architecture-scalable-infrastructure-b200/latest/, 2025. Document RA-11334-001. Accessed: 2026-02-18.

[13] Marcelo Amaral, Jordà Polo, David Carrera, Seetharami Seelam, and Malgorzata Steinder. Topology-aware gpu scheduling for learning workloads in cloud environments. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (*SC*), pages 1–12. ACM, 2017.

[14] Jeongkyu Shin and Joongi Kim. Sokovan: Container orchestrator for accelerated AI/ML workloads and massive-scale GPU computing. Presented at OpenInfra Summit Vancouver, June 2023. Slides available at https://www.backend.ai/ko/video/2023-06-11-openinfra-summit.

[15] NVIDIA Corporation. Dcgm-exporter: NVIDIA GPU monitoring tool for Prometheus. https://github.com/NVIDIA/dcgm-exporter, 2026. Accessed: 2026-02-15.

[16] Prometheus Authors. Node exporter: Prometheus exporter for hardware and OS metrics. https://github.com/prometheus/node_exporter, 2026. Accessed: 2026-02-15.

[17] Lablup Inc. all-smi: Cross-platform ai accelerator monitoring tool. https://github.com/lablup/all-smi, 2025. Accessed: 2026-02-05.

[18] NVIDIA Corporation. NVIDIA GPU memory error management. https://docs.nvidia.com/deploy/a100-gpu-mem-error-mgmt/index.html, 2025. v590. Accessed: 2026-03-24.

[19] John W. Young. A first order approximation to the optimum checkpoint interval. *Communications of the ACM*, 17(9):530–531, 1974.

[20] John T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Computer Systems*, 22(3):303–312, 2006.

[21] Leonardo Bautista-Gomez, Anne Benoit, Sheng Di, Thomas Herault, Yves Robert, and Hongyang Sun. A survey on checkpointing strategies: Should we always checkpoint à la Young/Daly? *Future Generation Computer Systems*, 161:315–328, 2024.

[22] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. Tiresias: A gpu cluster manager for distributed deep learning. In *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation* (*NSDI*), pages 485–500. USENIX Association, 2019.

[23] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, Fan Yang, and Lidong Zhou. Gandiva: Introspective cluster scheduling for deep learning. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation* (*OSDI*), pages 595–610. USENIX Association, 2018.

[24] Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R. Ganger, and Eric P. Xing. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In *Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation* (*OSDI*), pages 1–18. USENIX Association, 2021.

[25] Suhas Jayaram Subramanya, Daiyaan Arfeen, Shouxu Lin, Aurick Qiao, Zhihao Jia, and Gregory R. Ganger. Sia: Heterogeneity-aware, goodput-optimized ml-cluster scheduling. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles* (*SOSP*), pages 642–657. ACM, 2023.

[26] Pengfei Zheng, Rui Pan, Tarannum Khan, Shivaram Venkataraman, and Aditya Akella. Shockwave: Fair and efficient cluster scheduling for dynamic adaptation in machine learning. In *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation* (*NSDI*). USENIX Association, 2023.

[27] Sudarsanan Rajasekaran, Manya Ghobadi, and Aditya Akella. Cassini: Network-aware job scheduling in machine learning clusters. In *Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation* (*NSDI*). USENIX Association, 2024.

[28] Arnab Choudhury, Yang Wang, Tuomas Pelkonen, Kutta Srinivasan, Abha Jain, Shenghao Lin, Delia David, Siavash Soleimanifard, Michael Chen, Abhishek Yadav, Ritesh Tijoriwala, Denis Samoylov, and Chunqiang Tang. Mast: Global scheduling of ml training across geo-distributed datacenters at hyperscale. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation* (*OSDI*). USENIX Association, 2024.

[29] Qinlong Wang, Tingfeng Lan, Yinghao Tang, Ziling Huang, Yiheng Du, Haitao Zhang, Jian Sha, Hui Lu, Yuanchun Zhou, Ke Zhang, and Mingjie Tang. DLRover-RM: Resource optimization for deep recommendation models training in the cloud. *Proceedings of the VLDB Endowment*, 17, 2024.

[30] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[31] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kasber, Matei Zaharia, and Bryan Catanzaro. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (*SC*). ACM, 2021.

[32] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Proceedings of*

the 33rd International Conference on Neural Information Processing Systems (*NeurIPS*), pages 103–112, 2019.

[33] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (*SC*). IEEE, 2020.

[34] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Les Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. Pytorch fsdp: Experiences on scaling fully sharded data parallel. *Proceedings of the VLDB Endowment*, 16(12):3848–3860, 2023.

[35] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. Alpa: Automating inter- and intra-operator parallelism for distributed deep learning. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation* (*OSDI*), pages 559–578. USENIX Association, 2022.

[36] Jayashree Mohan, Amar Phanishayee, and Vijay Chidambaram. Checkfreq: Frequent, fine-grained dnn checkpointing. In *Proceedings of the 19th USENIX Conference on File and Storage Technologies* (*FAST*), pages 203–216. USENIX Association, 2021.

[37] Tanmaey Gupta, Sanjeev Krishnan, Rituraj Kumar, Abhishek Vijeev, Bhargav Gulavani, Nipun Kwatra, Ramachandran Ramjee, and Muthian Sivathanu. Just-in-time checkpointing: Low cost error recovery from deep learning training failures. In *Proceedings of the Nineteenth European Conference on Computer Systems* (*EuroSys*), pages 1110–1125. ACM, 2024.

[38] Zhuang Wang, Zhen Jia, Shuai Zheng, Zhen Zhang, Xinwei Fu, T. S. Eugene Ng, and Yida Wang. Gemini: Fast failure recovery in distributed training with in-memory checkpoints. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles* (*SOSP*), pages 364–381. ACM, 2023.

[39] Borui Wan, Mingji Han, Yiyao Sheng, Yanghua Peng, Haibin Lin, Mofan Zhang, Zhichao Lai, Menghan Yu, Junda Zhang, Zuquan Song, Xin Liu, and Chuan Wu. Bytecheckpoint: A unified checkpointing system for large foundation model development. In *Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation* (*NSDI*). USENIX Association, 2025.

[40] Insu Jang, Zhenning Yang, Zhen Zhang, Xin Jin, and Mosharaf Chowdhury. Oobleck: Resilient distributed training of large models using pipeline templates. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles* (*SOSP*), pages 382–395. ACM, 2023.

[41] John Thorpe, Pengzhan Zhao, Jonathan Eyolfson, Yifan Qiao, Zhihao Jia, Minjia Zhang, Ravi Netravali, and Guoqing Harry Xu. Bamboo: Making preemptible instances resilient for affordable training of large dnns. In *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation* (*NSDI*). USENIX Association, 2023.

[42] Sanjith Athlur, Nitika Saran, Muthian Sivathanu, Ramachandran Ramjee, and Nipun Kwatra. Varuna: Scalable, low-cost training of massive deep learning models. In *Proceedings of the Seventeenth European Conference on Computer Systems* (*EuroSys*), pages 472–487. ACM, 2022.

[43] Baodong Wu, Lei Xia, Qingping Li, Kangyu Li, Xu Chen, Yongqiang Guo, Tieyao Xiang, Yuheng Chen, and Shigang Li. TRANSOM: An efficient fault-tolerant system for training LLMs. *arXiv preprint arXiv:2310.10046*, 2023.

[44] Yongji Wu, Wenjie Qu, Xueshen Liu, Tianyang Tao, Yifan Qiao, Zhuang Wang, Wei Bai, Yuan Tian, Jiaheng Zhang, Z. Morley Mao, Matthew Lentz, Danyang Zhuo, and Ion Stoica. Lazarus: Resilient and elastic training of mixture-of-experts models. *arXiv preprint arXiv:2407.04656*, 2024.

[45] Jungsuk Kang and Joongi Kim. Method and apparatus for automatic recovery of tasks using execution failure-based resource requirement adjustment. Lablup Inc., 2026. Korean Patent Application 10-2026-0024429.

[46] NVIDIA Corporation. cuDNN backend release notes. https://docs.nvidia.com/deeplearning/cudnn/backend/latest/release-notes.html, 2024. Cumulative release notes covering cuDNN 9.x; multiple FP8/MXFP8/NVFP4-related fixes across versions. Accessed: 2026-04-21.

[47] Maxim Fishman, Brian Chmiel, Ron Banner, and Daniel Soudry. Scaling FP8 training to trillion-token LLMs. In *Proceedings of the Thirteenth International Conference on Learning Representations* (*ICLR*), 2025. Spotlight. arXiv:2409.12517.

[48] Joonhyung Lee, Jeongin Bae, Byeongwook Kim, Se Jung Kwon, and Dongsoo Lee. To FP8 and back again: Quantifying the effects of reducing precision on LLM training stability. *arXiv preprint arXiv:2405.18710*, 2024.

A 시스템 아키텍처 상세

본 부록에서는 Section 3에서 요약한 Backend.AI 인프라의 상세 구현을 기술한다. Sokovan 스케줄러의 핵심 설계(2계층 스케줄링, NUMA 인식 배치, 갱 스케줄링)는 Section 3.3에서 다루었으며, 여기서는 헬스 체크와 스토리지 아키텍처를 기술한다.

A.1 다중 계층 헬스 체크

Table 15는 Backend.AI의 헬스 체크 아키텍처를 보여준다.

Table 15: 다중 계층 헬스 체크 아키텍처

계층	메커니즘	타임아웃 / 임계값
인프라 (etcd)	주기적 활성 프로브	5.0 s
인프라 (Valkey/Redis)	컴포넌트별 핑	컴포넌트당 2.0 s, 전체 5.0 s
인프라 (PostgreSQL)	주기적 활성 프로브	2–5 s
에이전트 RPC	Manager → Agent 핑	5.0 s
에이전트 활성	하트비트 + 스윕	300 s 타임아웃, 600 s 스윕
에이전트 상태	Manager → Agent 하트비트	기본 40 s
작업 세션 행	상태별 허용 시간	PREPARING: 1 h, TERMINATING: 30 min
GPU 하드웨어	PCI 버스 열거 (`lspci`)	Rev `ff`/`00` = 결함
GPU 지표	`all-smi` Prometheus 엔드포인트	Alertmanager 내 임계값

A.2 통합 스토리지 아키텍처

Backend.AI는 프록시 기반 추상화를 통해 스토리지를 작업 세션 생명주기에 통합하여, 다양한 스토리지 백엔드(NFS, Ceph, 클라우드 스토리지 등)를 쿼타 및 운영 정책을 적용하면서도 네트워크 볼륨으로 일관되게 노출한다(Figure 18).

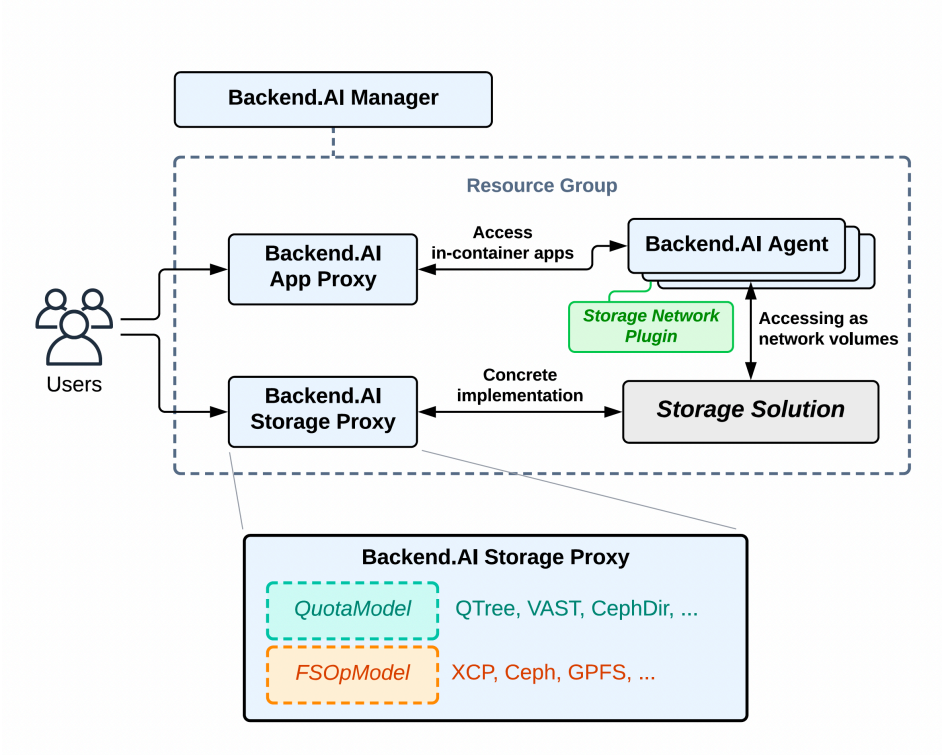

Figure 18: Backend.AI 스토리지 아키텍처 및 프록시 기반 통합. 스토리지 자원은 네트워크 볼륨으로 작업 세션에 노출되며, 쿼타 적용과 파일시스템 운영은 시스템 수준 모델로 관리된다.

B 용어집

Table 16: 용어 정의

용어	정의
Agent	Backend.AI의 노드별 구성요소로, 컨테이너 또는 가상 머신을 직접 관리하고, 물리적 자원(GPU, CPU, 메모리)를 할당하며, Manager에 상태를 보고한다
AOC	Active Optical Cable; 양 끝에 광-전기 변환 회로를 내장한 광 케이블. 수동형 구리 케이블(DAC)의 길이 한계를 넘어 GPU 클러스터의 노드–스위치 장거리 연결에 사용된다
Auto-retry	Backend.AI FastTrack의 자동 장애 복구 메커니즘으로, 작업 세션 장애를 감지하고 작업 세션을 재시작하여 학습 프레임워크가 마지막 체크포인트에서 재개할 수 있도록 한다; 구성 가능한 재시도 횟수 및 지연을 지원한다
cuDNN	CUDA Deep Neural Network 라이브러리; 컨볼루션, 정규화, 어텐션 연산 등 딥 뉴럴 네트워크를 위한 NVIDIA의 GPU 가속 기본 연산 라이브러리
DCGM	Data Center GPU Manager; 클러스터 환경에서 GPU를 모니터링하고 관리하기 위한 NVIDIA의 도구 모음
DGX	NVIDIA의 GPU 서버 플랫폼. 본 클러스터는 DGX B200 노드 63대로 구성된다

용어	정의
ECC	Error-Correcting Code; 비트 오류를 감지하고 정정하는 메모리 보호 메커니즘으로, GPU의 ECC 오류는 하드웨어 수준의 메모리 결함을 의미한다
etcd	클러스터 시스템에서 서비스 검색 및 구성 관리에 사용되는 분산 키-값 저장소
FastTrack	Backend.AI의 MLOps 오케스트레이션 계층으로, 자동 재시도, 작업 세션 모니터링, 장애 복구 워크플로를 통한 자동화된 학습 관리를 제공한다
FP8 / MXFP8	학습용 8비트 부동소수점 정밀도 형식. FP8은 메모리 및 연산 비용을 절감하며; MXFP8(Microscaling FP8)은 블록별 스케일링 팩터를 추가하여 수치 범위를 개선한다
FSDP	Fully Sharded Data Parallel; 모델 파라미터, 그래디언트, 옵티마이저 상태를 워커 간에 샤딩하는 PyTorch 분산 학습 전략
Gang Scheduling	전부 또는 전무(all-or-nothing) 스케줄링: 필요한 모든 자원을 한 번에 할당하거나, 전혀 할당하지 않는다
Goodput	체크포인팅, 통신 지연, 장애 복구 등의 오버헤드를 제외한 단위 시간당 완료된 유용한 학습 작업량
GSP	GPU System Processor; GPU 펌웨어를 실행하는 RISC-V 마이크로컨트롤러로, 호스트 드라이버와 RPC로 통신한다. RPC 타임아웃 시 XID 119로 보고된다
HBM	High Bandwidth Memory; GPU를 위한 고대역폭, 고용량 메모리를 제공하는 적층 DRAM. HBM3e는 NVIDIA B200 GPU에 사용되는 변형(디바이스당 192 GB)이다
HDFS	Hadoop Distributed File System; Hadoop 생태계의 대용량 분산 파일 시스템. 블록 기반의 쓰기 1회 · 읽기 N회(write-once-read-many) 모델로, 빅데이터 배치 워크로드에 최적화되어 있다
HSDP	Hybrid Sharded Data Parallel; 노드 그룹 내에서의 FSDP 샤딩과 그룹 간 데이터 병렬화를 결합한 분산 학습 전략
InfiniBand	HPC 및 AI 클러스터에서 노드 간 GPU 통신에 사용되는 고속, 저지연 인터커넥트 패브릭. NDR은 400 Gbps 세대를 나타낸다
IOPS	Input/Output Operations Per Second; 랜덤 접근 워크로드에 대한 스토리지 시스템 처리량의 측정 단위
IQR	Interquartile Range(사분위 범위); 25^{th}–75^{th} 백분위 사이의 범위로, 분포의 중간 50%를 나타낸다
JCT	Job Completion Time(작업 완료 시간)
Manager	Backend.AI의 중앙 제어 구성요소로, 클러스터 전체의 스케줄링 결정을 조율하고, 작업 세션 생명주기 상태를 관리하며, RPC를 통해 Agent와 통신한다
MFU	Model FLOPs Utilization; 하드웨어의 이론적 최대 FLOPS 대비 관측된 처리량의 비율
MoE	Mixture of Experts; 입력 토큰당 파라미터의 일부만 활성화하여 더 낮은 토큰당 연산 비용으로 더 큰 총 모델 용량을 가능하게 하는 모델 아키텍처
MTBF	Mean Time Between Failures(평균 고장 간격)
NCCL	NVIDIA Collective Communications Library; 다중 GPU 및 다중 노드 학습을 위한 최적화된 집합 연산(all-reduce, broadcast 등)을 제공한다

용어	정의
NFS	Network File System; 클러스터 노드 간 공유 스토리지 접근을 가능하게 하는 분산 파일 시스템 프로토콜
NIC	Network Interface Card; 네트워크 인터페이스 카드. 본 클러스터는 노드당 GPU 통신용 InfiniBand NIC과 스토리지용 RoCE NIC을 별도로 구성한다
NUMA	Non-Uniform Memory Access; 다중 소켓 시스템에서 메모리 접근 지연이 CPU와 메모리의 상대적 위치에 따라 달라지는 메모리 아키텍처
NVLink	노드 내 GPU 간 직접 통신을 위한 NVIDIA의 고대역폭 인터커넥트
OOM	Out of Memory; 프로세스가 가용 메모리(일반적으로 GPU 메모리)보다 많은 메모리를 요청할 때 발생하는 런타임 오류
PCIe	Peripheral Component Interconnect Express; 시스템 내 GPU, NIC, 스토리지 디바이스를 호스트와 연결하는 고속 직렬 버스
Prometheus	시계열 지표 수집 및 쿼리를 위한 오픈소스 모니터링 시스템. 본 보고서의 751개 지표는 Prometheus 호환 형식으로 수집되어 VictoriaMetrics에 저장되었다
Resource group	Backend.AI의 클러스터 자원에 대한 논리적 파티셔닝 단위. 스케줄러는 각 자원 그룹을 독립적으로 처리하여 메모리 사용을 제한하고 장애를 격리한다
RoCE	RDMA over Converged Ethernet; 이더넷 위에서 RDMA(Remote Direct Memory Access)를 구현하는 네트워크 프로토콜. 본 클러스터에서는 200 Gbps RoCE NIC을 스토리지 전용 네트워크로 사용한다
RPC	Remote Procedure Call; 다른 프로세스 또는 호스트의 함수를 호출하는 통신 메커니즘. Backend.AI의 Manager–Agent 통신, NFS의 클라이언트–서버 요청 등에 사용된다
Session	Backend.AI에서 여러 노드의 복수 컨테이너에 걸칠 수 있는 논리적 학습 작업 단위로, 스토리지, 구성, 생명주기 상태를 단일 엔티티로 묶는 작업 세션을 뜻한다
Sokovan	Backend.AI의 오케스트레이션 계층. 작업 세션 스케줄링(NUMA 인식 배치, 갱 스케줄링), 배포 관리, 라우트 관리를 통합하며, 힌트 기반 듀얼 루프로 이벤트에 반응한다. 본 보고서에서는 학습 작업 세션 스케줄링 기능을 중심으로 기술한다
Temporal occupancy	관측 기간 중 학습 작업 세션이 클러스터를 점유한 비율로, 누적 작업 세션 경과 시간을 관측 기간으로 나누어 산출한다
XID	NVIDIA GPU 오류 식별자; 하드웨어 및 소프트웨어 오류를 분류하기 위해 GPU 드라이버가 보고하는 숫자 코드

C GPU 모니터링 대시보드

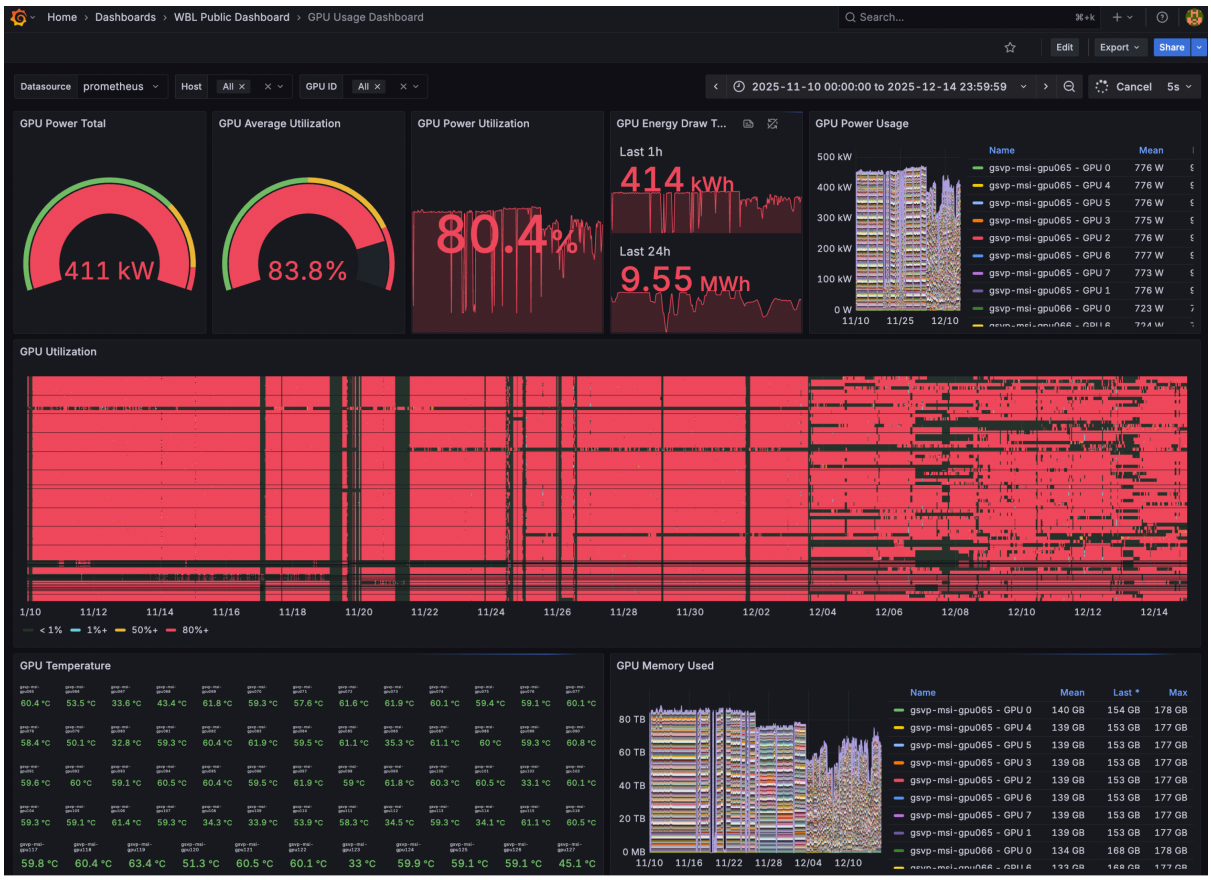

Figure 19: Backend.AI와 함께 배포된 Grafana 기반 GPU 모니터링 대시보드. 이 대시보드는 클러스터 전체 및 노드별 GPU 전력 소비, 활용률, 온도, SM 클럭, 메모리 사용량, 에너지 소비를 실시간으로 시각화하며, NVIDIA DCGM 및 `all-smi` 지표를 Prometheus를 통해 집계한다. 이 원격 계측 데이터는 Section 4에 제시된 장애 분석의 관측 기반이 된다.

D 저자 목록

다음은 Backend.AI의 개발 및 운영과 본 보고서에 기술된 인프라에 기여한 저자 목록이다. 저자명은 이름을 기준으로 알파벳 순으로 나열되었다.

Daemyung Kang, Eunjin Hwang, Hanjeong Lee*, HyeokJin Kim, Hyunhoi Koo, Jeongkyu Shin, Jeongseok Kang, Jihyun Kang, Joongi Kim, Junbum Lee, Jungseung Yang, Kyujin Cho, Youngsook Song

* 래블업 주식회사의 인턴십 기간 중 수행한 작업임.