Didim

AI · Big Data · Cloud Specialist

클라우드 대규모 LLM 환경 구축 및 운영

클라우드 대규모 LLM환경에 대한 Didim만의 노하우

2026년 4월 2일

목차

1 사례 개요

1.1 배경

정부에서 국내 AI 연구팀을 위한 고성능 GPU 인프라 구축을 목표로 하는 국가 AI 사업을 발표했습니다. 해당 사업에서 AWS 는 한국정보통신기술진흥협회(KAIT)가 주관하는 "AI for Research" 프로젝트의 GPU 자원 공급업체로 선정되어 8 개월 동안 30 개의 p5en.48xlarge 인스턴스(H200 GPU 240 개)를 제공했습니다. 그리고 그렇게 제공된 GPU 자원은 한국과학기술원(KAIST), 광주과학기술원(GIST), 서울대학교병원, 서울대학교, 한양대학교, 성균관대학교, 고려대학교 등 여러 기관의 AI 연구팀에서 8 개월 동안 사용했습니다.

이 프로젝트에서 디딤은 대규모 LLM 학습을 지원할 수 있도록 GPU 를 포함한 클라우드 인프라 환경을 구축하고, 해당 기관들이 안정적으로 GPU 자원을 사용할 수 있도록 운영 서비스도 제공했습니다.

1.2 도전 과제

이 프로젝트에서 필요한 대규모 LLM 학습을 지원하는 인프라 환경은 단순히 GPU 클러스터를 구성하는 것만으로는 충분하지 않았고, 다음과 같은 도전 과제들이 존재했습니다.

- GPU, 스토리지, 네트워크, 스케줄러의 간의 긴밀한 통합 필요
- 몇 주 간 지속되는 학습 환경에서의 안정적인 운영 보장
- 발생하는 다양한 상황들에 대한 신속한 대처 필요

1.3 해결 방안

그리고 위 도전 과제들의 해결을 위해 다음과 같은 도구들을 활용했습니다.

'2.1 AWS 의 ParallelCluster 활용': 클러스터 신속 배포 및 관리

'2.2 Slurm 파티션 및 Slurm Accounting 활용': 자원 공정성 보장, 자원 남용 방지, 운영 효율성 증대

'2.3 FSx for Lustre 활용': 고속 I/O 제공 및 저지연, 고대역폭 처리

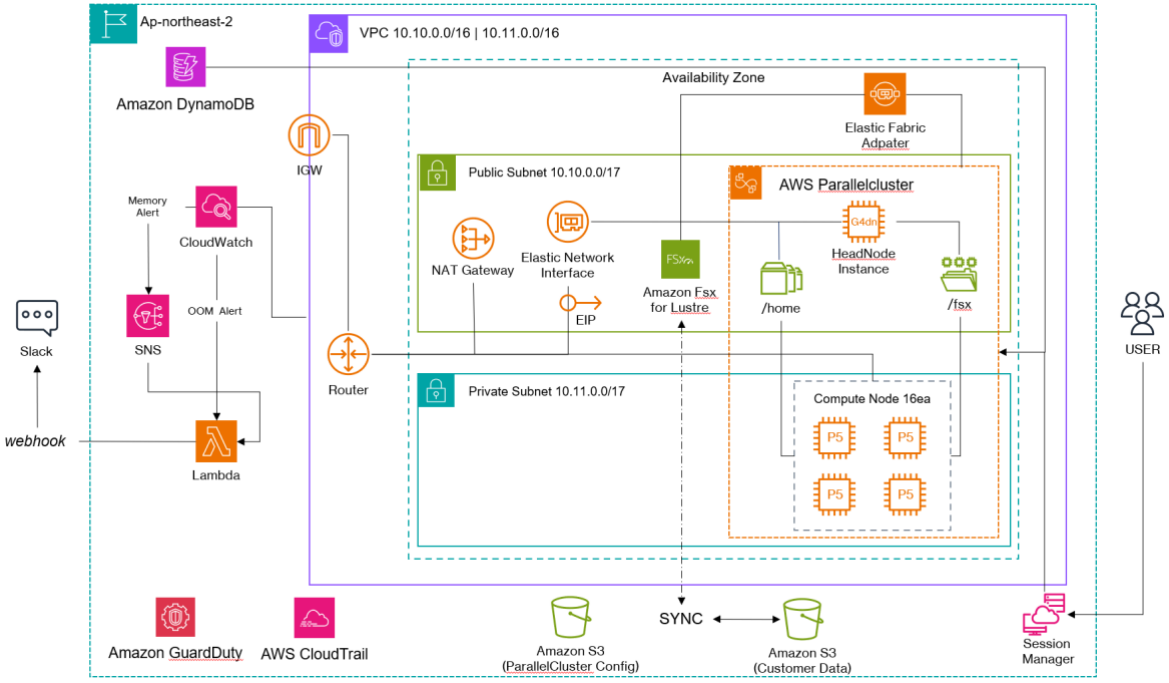
2 사례 세부 내용

2.1 ParallelCluster 활용

AWS ParallelCluster 는 AWS 에서 고성능 컴퓨팅(HPC) 클러스터를 신속하게 배포하고 관리하는 오픈 소스입니다.

2.1.1 ParallelCluster 아키텍처

ParallelCluster 를 사용한 아키텍처는 다음과 같습니다.



2.1.2 ParallelCluster 로 GPU 클러스터 구성하기

대규모 GPU 클러스터를 운영한다고 해서 무조건 복잡한 아키텍처가 필요한 것은 아닙니다.

AWS ParallelCluster 는 YAML 기반의 설정 파일(cluster-config.yaml) 하나만으로도 손쉽게 클러스터를 정의하고 확장할 수 있도록 지원합니다.

이번 환경에서는 특히 MinCount 와 MaxCount 값을 동일하게 고정해 항상 일정한 수량의 GPU 노드를 확보하는 방식을 채택했습니다.

일반적인 ParallelCluster 기본 동작은 HeadNode 만 띄워 두고, Slurm Job 제출 시 필요한 만큼의 Compute Node 를 일시적으로 생성하는 구조이지만, 본 프로젝트에서는 연구 특성상 항상 동일한 자원 규모를 안정적으로 제공하는 것이 더 중요했기 때문입니다.

이제부터 실제 사용한 cluster-config.yaml 을 기반으로, 각 주요 라인별로 어떤 의미가 있고 왜 해당 방식으로 설정했는지 간략히 설명하겠습니다.

[cluster-config.yaml]

```
1 Imds:
2   ImdsSupport: v2.0
3 Image:
4   Os: ubuntu2204
5   CustomAmi: ${CUSTOM_AMI}
6 HeadNode:
7   InstanceType: p5en.48xlarge
8   Ssh:
9     KeyName: ${SSH_KEY}
10  Networking:
11    SubnetId: ${PUBLIC_SUBNET_ID}
12    AdditionalSecurityGroups:
13      - ${SECURITY_GROUP}
14    ElasticIp: ${HEADIP}
15  LocalStorage:
16    RootVolume:
17      Size: 500
18      Iops: 16000
19      Throughput: 1000
20      DeleteOnTermination: true
21  Iam:
22    AdditionalIamPolicies:
23      - Policy: arn:aws:iam::aws:policy/AdministratorAccess
24  CustomActions:
```

```
25    OnNodeConfigured:
26      Script: s3://${S3_NAME}/script/main_headconf-script.sh
27 Scheduling:
28   Scheduler: slurm
29   SlurmSettings:
30     ScaledownIdletime: 60
31     QueueUpdateStrategy: DRAIN
32       EnableMemoryBasedScheduling: true
33       CustomSlurmSettingsIncludeFile: s3:// ${BUCKET_NAME}/script/slurm-settings.conf
34   SlurmQueues:
35     - Name: compute
36       CapacityReservationTarget:
37         CapacityReservationId: ${COM_CRID}
48       Networking:
49         SubnetIds:
50           - ${Private_SUBNET_ID}
51         PlacementGroup:
52           Enabled: false
53         AdditionalSecurityGroups:
54           - ${SECURITY_GROUP}
55       Iam:
56         AdditionalIamPolicies:
57          - Policy: arn:aws:iam::aws:policy/AdministratorAccess
58       ComputeSettings:
59         LocalStorage:
60           EphemeralVolume:
61             MountDir: /opt/dlami/nvme
62           RootVolume:
63             Size: 500
64       JobExclusiveAllocation: false
65       ComputeResources:
66         - Name: ai-gpu
67           InstanceType: p5en.48xlarge
68           MinCount: ${INSTANCES_NUM}
69           MaxCount: ${INSTANCES_NUM}
70           Efa:
```

```
71            Enabled: true
72        CustomActions:
73          OnNodeStart:
74            Script: s3://${S3_NAME}/script/main_comstart-script.sh
75 DevSettings:
76   Timeouts:
77     HeadNodeBootstrapTimeout: 3600
78 SharedStorage:
79   - MountDir: /fsx
80     Name: fsx
81     StorageType: FsxLustre
82     FsxLustreSettings:
83       FileSystemId: ${FSX_ID}
84 Monitoring:
85   DetailedMonitoring: true
86   Logs:
87     CloudWatch:
88       Enabled: true
89   Dashboards:
90     CloudWatch:
91       Enabled: true
92 Tags:
93   - Key: 'Grafana'
94     Value: 'true'
```

[주요 옵션 설명]

- **QueueUpdateStrategy: DRAIN**
  클러스터 설정을 업데이트할 때, 기존 Job 을 지켜주면서 점진적으로 노드를 교체하는 안전한 업데이트 방식 제공

- **EnableMemoryBasedScheduling: true**
  Slurm 스케줄러가 Job 을 배치할 때 기본으로 CPU 만 고려하지 말고, 메모리 사용량까지 고려하도록 활성화, 비활성화 시 메모리 부족 발생 가능성

- **CapacityReservationId**

항상 동일한 수량의 GPU 노드를 확보하기 위해 Targeted Capacity Reservation 을 적용

- **JobExclusiveAllocation: false**
  한 노드에 여러 Job 을 동시에 배치해 자원 활용도를 높이는 설정, 활성화 시 자원 여유가 있어도 하나의 노드에는 하나의 job 만 할당 가능하기 때문에, 자원 활용도를 위해서 비활성화 설정

- **MinCount:**
  MaxCount:
  HPC, 딥러닝 분산 학습에서 항상 정해진 수의 GPU 노드가 필요한 경우, 인스턴스 최대 수량과 최소 수량을 동일하게 해서 고정된 수량으로 클러스터 환경을 운영

- **Efa:**
  Enabled: true
  딥러닝 분산 학습 같이 노드 간 통신이 많은 워크로드에 더 낮은 레이턴시와 더 높은 대역폭을 위해서 컴퓨트 노드 그룹에 Elastic Fabric Adapter(EFA)를 활성화

- **CustomSlurmSettingsIncludeFile**
  Slurm 파티션 및 Accounting 세팅을 추가로 정의하기 위해 사용

2.2 Slurm 파티션 및 Slurm Accounting 활용

Slurm 파티션은 노드들의 논리적인 그룹핑 또는 대기열을 의미하고, Slurm Accounting 은 HPC 클러스터에서 작업 및 사용자가 어떤 리소스를 얼마나 사용했는지 기록하고 관리하는 기능입니다.

2.2.1 Slurm 파티션과 Slurm Accounting 도입 필요성 및 배경

Slurm 파티션은 작업을 그룹화하고 우선순위를 지정하는 데 유용하지만, 기본적으로는 단순한 “큐” 역할에 가까워서 사용자에 대한 통제력이 부족합니다.

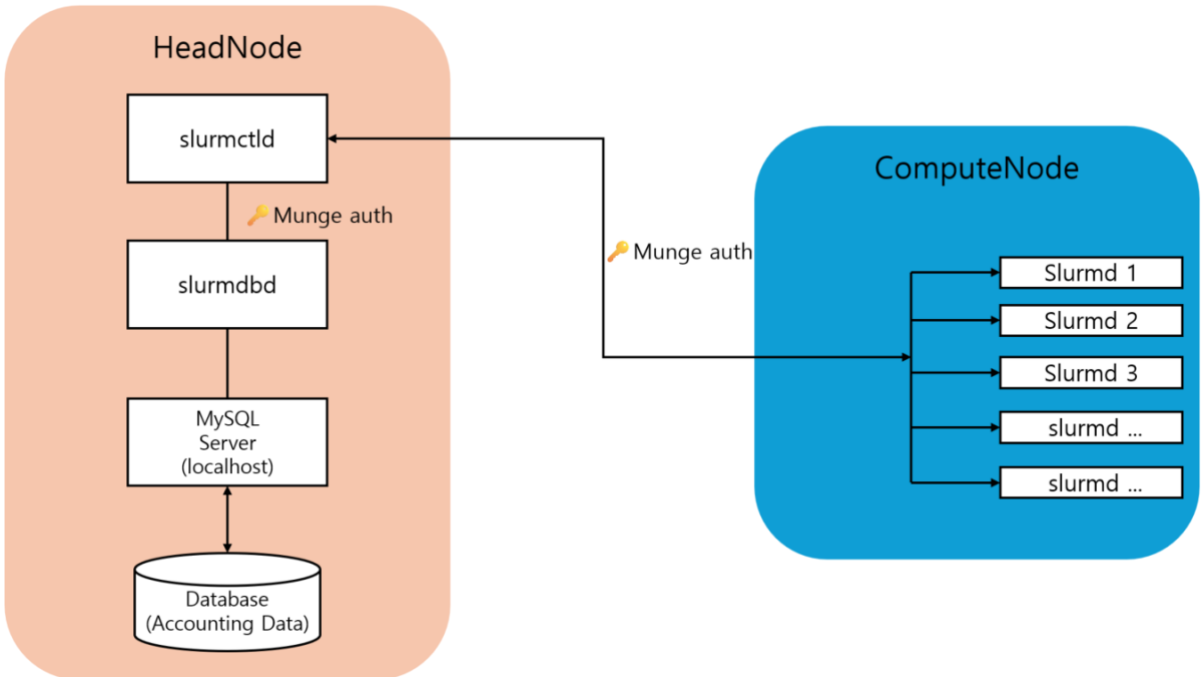
그 결과 여러 사용자가 동시에 환경을 공유할 때, 일부 사용자가 협의된 자원 제한을 무시하거나 과도하게 Job 을 제출하면 특정 연구자가 자원을 독점하게 되고, 다른 사용자의 작업이 지연되는 불공정한 상황이 발생할 수 있습니다.

이를 방지하고자 Slurm Accounting 을 도입했습니다. Slurm Accounting 은 사용자, 계정 단위로 Job 제출 개수, 동시 실행 가능 개수, 최대 실행 시간 등을 강제적으로 제어할 수 있으며, 모든 사용 이력을 데이터베이스에 기록해 자원 사용 관리를 가능하게 합니다.

즉, Slurm 파티션이 "자원 배분을 위한 틀"이라면, Slurm Accounting 은 이를 실제로 강제하고 추적하는 관리 체계라고 할 수 있습니다. 이 두 기능을 결합함으로써 공정성을 보장하고, 자원 남용을 방지하며, 운영 효율성을 높일 수 있었습니다.

구분	Slurm Partition	Slurm Accounting
제어 단위	파티션(Partition)	사용자(User), 계정(Account), QoS
가능한 제약	- 파티션별 실행 시간(MaxTime) - 파티션별 노드 수(MaxNodes) - 파티션 우선순위(Priority) - Preempt 설정(PreemptMode)	- 사용자별 동시 실행 Job 수 (MaxJobsPerUser) - 사용자별 제출 Job 수 (MaxSubmitJobsPerUser) - 계정/그룹 단위 GPU·CPU·메모리 한도 (GrpTRES) - 팀별 총 Job 수/리소스 제한
우선순위 관리	파티션 Priority 값	QoS Priority + Weight 조합으로 정밀 제어
정책 강제 여부	Slurm 기본 동작 (권장 수준)	AccountingStorageEnforce 로 강제 적용
활용 목적	단순 자원 분리 / Job 이 어떤 파티션에서 실행될지 관리	팀/사용자별 세밀한 자원 통제 및 사용량 추적

[기본 파티션 기능 vs Slurm Accounting 기능 비교]



[Slurm Accounting 다이어그램]

[Slurm 주요 옵션 설명: slurm-settings.conf]

- **PreemptType=preempt/partition_prio**
  파티션 우선순위(Partition Priority)를 기준으로 선점(preemption) 동작을 결정

- **PreemptMode=REQUEUE**
  밀려난(Job preempted) 작업은 강제로 종료되지 않고 대기열(Pending Queue)로 재등록

- **UnkillableStepTimeout=300**
  Slurm 이 특정 Job Step 을 강제 종료할 때, 300 초(5 분)까지 안전하게 종료 대기

- **PartitionName=debug Nodes=compute-st-kait-gpu-1 Priority=100 MaxTime=00:30:00 Default=NO State=UP AllowQOS=debug_qos**
  짧은 Job(최대 30 분)을 빠르게 테스트하는 파티션. Priority=100 으로 낮은 우선순위, debug 파티션에 debug_qos 를 매핑 적용

- **PartitionName=normal Nodes=compute-st-kait-gpu-1 Priority=200 MaxTime=1-00:00:00 Default=YES State=UP AllowQOS=normal_qos**
  일반 Job 제출용 (최대 1 일). -p 옵션을 지정하지 않으면 자동으로 이 파티션에 제출.
  normal 파티션에 normal_qos 를 매핑 적용

- **PartitionName=priority Nodes=compute-st-kait-gpu-[1-4] Priority=300 MaxTime=5-00:00:00 Default=NO State=UP AllowQOS=priority_qos**
  여러 노드(1~4 번) 사용 가능, 최대 5 일. Priority=300 으로 높은 우선순위, priority 파티션에 priority_qos 를 매핑 적용

- **PartitionName=exclusive Nodes=compute-st-kait-gpu-[1-4] Priority=1000 MaxTime=14-00:00:00 Default=NO State=UP PreemptMode=REQUEUE AllowQOS=exclusive_qos**
  여러 노드(1~4 번)를 독점적으로 사용, 최대 14 일. Priority=1000 으로 최우선순위,
  exclusive 파티션에 exclusive_qos 를 매핑 적용, 높은 우선순위 Job 이 오면 기존 Job 을 중단 후 대기열로 되돌림

- **AccountingStorageTRES=cpu,mem,node,gres/gpu**
  기록할 리소스 종류(TRES, Trackable RESources)를 지정

- **AccountingStorageEnforce=limits.qos.associations**
  QOS 와 사용자(Account) 단위로 설정된 제한 사항을 강제 적용

- **PriorityWeightPartition=100000**
  Partition 우선순위(PartitionName 에 정의된 Priority 값)을 얼마나 크게 반영할지를 정하는 가중치

- **PriorityWeightQOS=1000**
  QOS(서비스 품질, Quality of Service) Priority 를 얼마나 반영할지에 대한 가중치

- **PriorityWeightFairshare=100**
  Fairshare(공정 배분, Share 값 기반) 점수를 얼마나 반영할지를 정하는 가중치

2.2.2 Slurm Accounting 을 이용한 사용자별 QOS 설정

앞서 소개한 Slurm 파티션의 중요 설정들을 이용해서 자율성 있게 파티션 단위의 제약을 제공했지만, 각 연구자가 동시에 Slurm 클러스터를 활용하는 환경에서 공정하고 효율적인 자원 운영을 하기 위해서 Slurm Accounting 을 추가로 구성했습니다. 아래는 연구팀의 자원제어를 위한 요청 사항입니다.

파티션	접근 권한	WallTime	동시 실행 제한	동시 제출 제한	실행 노드	우선순위 (파티션 기준)	Default QOS	Sharefair
debug	일반 + 우선	30 분	1 개	1 개	Node1	낮음 (100)	X	전역설정
normal	일반 + 우선	24 시간	4 개	8 개	Node1	기본 (200)	O	
priority	우선 사용자만	120 시간	제한 없음	제한 없음	Node1-4	높음 (300)	O	
exclusive	우선 사용자만	336 시간	1 개	2 개	Node1-4	최우선 (1000)	X	

[Job Scheduling Policy Table]

이 정책은 각 파티션마다 접근 권한, 최대 실행 시간(WallTime), 동시 실행 및 제출 제한, 우선순위 등을 정의한 것으로, 연구자들이 자원을 협의된 방식대로 사용할 수 있도록 기준을 마련한 것입니다.

예를 들어, debug 파티션은 짧은 테스트 용도로 일반 사용자도 접근 가능하지만 최대 30 분, 동시 1 개 Job 만 실행 가능하도록 제한했습니다.

반면, exclusive 파티션은 우선 사용자 전용으로 최대 14 일간 단독 실행이 가능하며, 높은 우선순위로 다른 작업보다 먼저 배치되도록 구성했습니다.

이러한 Job Scheduling Policy 는 단순히 문서화된 규칙이 아니라, Slurm Accounting 및 QOS 설정과 연계되어 실제 운영 환경에서 강제 적용됩니다.

이러한 정책을 실질적으로 구현하기 위해 사용한 sacctmgr 기반의 계정, QOS 생성 명령과 Slurm 설정 방법을 구체적으로 소개합니다.

2.2.3 Slurm Accounting 사용자 등록 절차

Slurm Accounting 사용자 등록 절차는 다음과 같습니다.

1. OS 계정 생성

Slurm Accounting 에 등록하려는 사용자는 반드시 Linux OS User 가 먼저 생성되어 있어야 합니다.

2. slurm accounting 에 사용자 등록을 위해 사용자 Account 생성

```
# 일반 사용자 그룹 Account
sacctmgr --immediate add account general Description="general user group"
# 우선 사용자 그룹 Account
sacctmgr --immediate add account priority Description="priority user group"
```

3.Slurm Accounting 에 사용자 등록 (sacctmgr)

OS 에 생성한 사용자를 Slurm Accounting DB 에 등록해야 Slurm 이 해당 사용자의 작업을 추적할 수 있습니다.

```
# 일반 사용자 그룹 Account User 등록
sacctmgr --immediate add user name=$USER account=general
# 우선 사용자 그룹 Account User 등록
sacctmgr --immediate add user name=$USER account=priority
```

4. 파티션마다 Mapping 시킬 QoS 생성

```
sacctmgr --immediate add qos debug_qos
sacctmgr --immediate add qos normal_qos
sacctmgr --immediate add qos priority_qos
sacctmgr --immediate add qos exclusive_qos
```

5. QoS 에 동시 실행, 동시 제출 및 우선순위 조건 설정

```
sacctmgr --immediate modify qos debug_qos set MaxJobsPerUser=1 MaxsubmitJobsPerUser=1 Priority=1000

sacctmgr --immediate modify qos normal_qos set MaxJobsPerUser=4 MaxsubmitJobsPerUser=8 Priority=2000

sacctmgr --immediate modify qos priority_qos set Priority=3000

sacctmgr --immediate modify qos exclusive_qos set MaxJobsPerUser=1 MaxSubmitJobsPerUser=2 Priority=10000
```

Name	Priority	MaxJobsPU	MaxSubmitPU
normal	0		
debug_qos	1000	1	1
normal_qos	2000	4	8
priority_qos	3000		
exclusive_qos	10000	1	2

[Slurm QOS Policy Table]

6. 생성한 Account 에 QOS 매핑

```
#priority 계정의 유저는 전부 액세스 할 수 있도록 설정

sacctmgr -i modify account priority set qos+=debug_qos,normal_qos,priority_qos,exclusive_qos


#general 계정의 모든 유저는 debug_qos, normal_qos만 사용 가능

sacctmgr -i modify account general set qos+=debug_qos,normal_qos
```

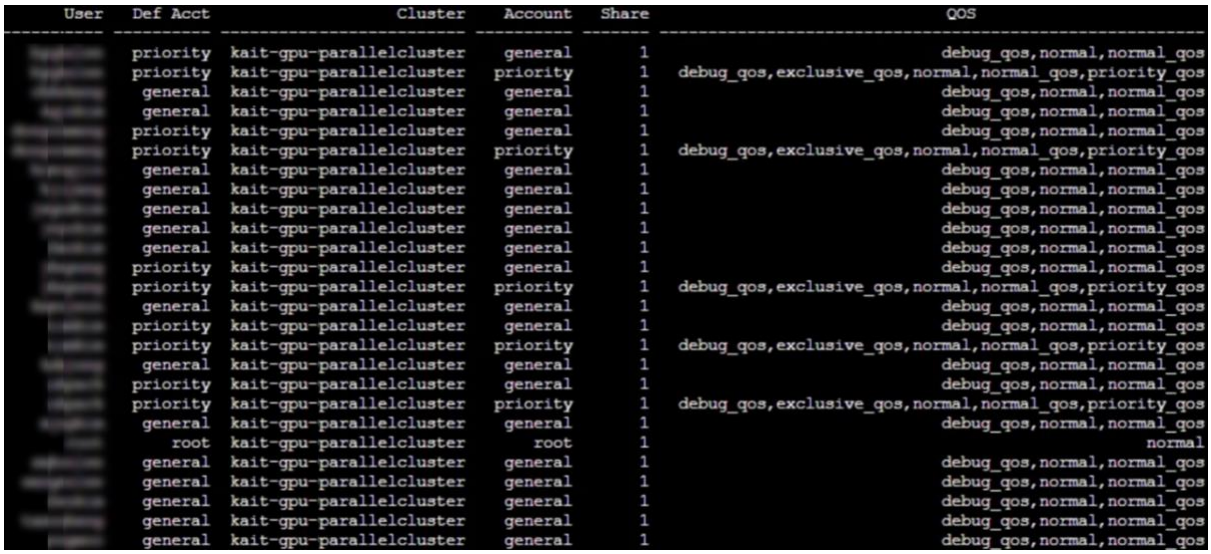
7. DefaultQoS 설정 (QoS 미 지정 시 기본 할당할 Qos 정책)

```
#우선 사용자는 pirority_qos 할당

sacctmgr -i modify account priority set DefaultQOS=priority_qos


# 일반 사용자는 general_qos 할당

sacctmgr -i modify account general set DefaultQOS=normal_qos
```

8. 우선 사용자의 경우 DefaultAccount 를 지정 (Account Fix Setting)

```
sacctmgr -i modify user $USER set DefaultAccount=priority
```

위와 같은 설정 절차를 통해서, 사용자에 대한 QOS 정책이 정상 반영된 것을 확인할 수 있습니다.



```
      User   Def Acct                   Cluster    Account     Share                                                 QOS
---------- ---------- ------------------------- ---------- --------- ---------------------------------------------------
[redacted]   priority kait-gpu-parallelcluster     general         1                             debug_qos,normal,normal_qos
[redacted]   priority kait-gpu-parallelcluster    priority         1  debug_qos,exclusive_qos,normal,normal_qos,priority_qos
[redacted]    general kait-gpu-parallelcluster     general         1                             debug_qos,normal,normal_qos
[redacted]    general kait-gpu-parallelcluster     general         1                             debug_qos,normal,normal_qos
[redacted]   priority kait-gpu-parallelcluster     general         1                             debug_qos,normal,normal_qos
[redacted]   priority kait-gpu-parallelcluster    priority         1  debug_qos,exclusive_qos,normal,normal_qos,priority_qos
[redacted]    general kait-gpu-parallelcluster     general         1                             debug_qos,normal,normal_qos
[redacted]    general kait-gpu-parallelcluster     general         1                             debug_qos,normal,normal_qos
[redacted]    general kait-gpu-parallelcluster     general         1                             debug_qos,normal,normal_qos
[redacted]    general kait-gpu-parallelcluster     general         1                             debug_qos,normal,normal_qos
[redacted]    general kait-gpu-parallelcluster     general         1                             debug_qos,normal,normal_qos
[redacted]   priority kait-gpu-parallelcluster     general         1                             debug_qos,normal,normal_qos
[redacted]   priority kait-gpu-parallelcluster    priority         1  debug_qos,exclusive_qos,normal,normal_qos,priority_qos
[redacted]    general kait-gpu-parallelcluster     general         1                             debug_qos,normal,normal_qos
[redacted]   priority kait-gpu-parallelcluster     general         1                             debug_qos,normal,normal_qos
[redacted]   priority kait-gpu-parallelcluster    priority         1  debug_qos,exclusive_qos,normal,normal_qos,priority_qos
[redacted]    general kait-gpu-parallelcluster     general         1                             debug_qos,normal,normal_qos
[redacted]   priority kait-gpu-parallelcluster     general         1                             debug_qos,normal,normal_qos
[redacted]   priority kait-gpu-parallelcluster    priority         1  debug_qos,exclusive_qos,normal,normal_qos,priority_qos
[redacted]    general kait-gpu-parallelcluster     general         1                             debug_qos,normal,normal_qos
[redacted]       root kait-gpu-parallelcluster        root         1                                                  normal
[redacted]    general kait-gpu-parallelcluster     general         1                             debug_qos,normal,normal_qos
[redacted]    general kait-gpu-parallelcluster     general         1                             debug_qos,normal,normal_qos
[redacted]    general kait-gpu-parallelcluster     general         1                             debug_qos,normal,normal_qos
[redacted]    general kait-gpu-parallelcluster     general         1                             debug_qos,normal,normal_qos
[redacted]    general kait-gpu-parallelcluster     general         1                             debug_qos,normal,normal_qos
```

[Slurm User-Account-QOS Association Table]

2.3 FSx for Lustre 활용

FSx for Lustre 는 고성능 컴퓨팅(HPC), 머신러닝, 영상 처리 등 대규모 데이터 처리에 최적화된 AWS 의 고속 공유 파일 시스템입니다.

2.3.1 대규모 학습 환경, 왜 Lustre 를 선택했나?

대규모 분산 학습에서는 수십 개 노드가 동시에 TB~PB 단위 데이터를 읽고 쓰기 때문에, 일반적인 NFS/EFS 로는 성능이 부족합니다. Lustre 는 데이터를 여러 OST 에 병렬로 분산해 고속 I/O 를 제공하고, EFA 네트워크와 결합해 저지연, 고대역폭 처리가 가능합니다. 또한 S3 연동과 File Release 기능으로 확장성과 비용 효율성까지 확보할 수 있어, 대규모 학습 환경에 최적의 선택이라고 볼 수 있습니다.

2.3.2 대규모 LLM 학습을 위한 FSx for Lustre 성능 최적화

대규모 LLM 학습 환경에서는 수십~수백 개의 GPU 노드가 동시에 Lustre 에 접근하기 때문에, 기본 설정만으로는 I/O 병목이나 메타데이터 지연이 발생할 수 있습니다. 이를 개선하기 위해 Lustre 의 주요

파라미터를 조정하여 최적화했습니다.

```
#시스템이 유지할 수 있는 Lock의 최대 개수를 설정
sudo lctl set_param ldlm.namespaces.*.lru_size=192

# OST(Object Storage Target)에 대한 동시 RPC 요청 수를 32개로 설정
sudo lctl set_param osc.*OST*.max_rpcs_in_flight=32

# MDT(Metadata Target)에 대한 일반적인 동시 RPC 요청 수를 64개로 설정
sudo lctl set_param mdc.*.max_rpcs_in_flight=64

# MDT에 대한 수정 작업 관련 동시 RPC 요청 수를 50개로 설정
sudo lctl set_param mdc.*.max_mod_rpcs_in_flight=50
```

2.3.3 S3 + FSx for Lustre 기반 Traning Data 준비 프로세스(Lazy Loading)

대규모 LLM 학습 환경에서는 데이터 준비와 업로드 시간이 곧 비용과 직결됩니다.

GPU 클러스터를 제공받았더라도 학습용 데이터셋이 아직 공유 스토리지에 업로드되지 않았다면, 고가의 GPU 인스턴스가 한동안 Idle 상태로 낭비될 수 있습니다. 따라서 데이터를 사전에 준비해 두고, GPU 자원 할당과 동시에 학습을 시작할 수 있는 환경을 갖추는 것이 핵심입니다.

이번 프로젝트에서는 연구팀에 GPU 를 제공하기 전에 원본 저장소로 Amazon S3 를 활용해 학습 데이터를 업로드하도록 했습니다. 이후 Lustre 와 S3 를 연동해 동기화했으며, Lazy Loading 의 특성을 활용해 미리 데이터 Preloading 을 마무리했습니다. 덕분에 연구팀은 GPU 제공과 동시에 Lustre 를 마운트해 곧바로 학습을 시작할 수 있었습니다.

1. Lustre 의 원본 저장소 S3 로 사용

A. 학습 데이터셋은 수 TB 단위로 커질 수 있으며, 버전 관리나 백업도 필요

B. S3 를 데이터 저장소로 두면, 무제한에 가까운 확장성과 안정적인 관리가 가능

2. Lazy Loading 의 장점

A. 전체 데이터셋을 한 번에 Lustre 로 복사하지 않고, 필요한 파일만 액세스 시점에 가져오는 방식

B. GPU 클러스터가 준비되자마자 학습이 시작 가능

C. 초기 스토리지 부담 최소화 → Lustre 용량 관리에도 유리

3. 비용 효율성

A. GPU 사용 시작부터 바로 training 이 가능 → 불필요한 리소스 사용을 크게 줄일 수 있음

2.3.4 정기적인 File Release on Lustre Task 수행으로 용량 관리

대규모 학습 환경에서는 FSx for Lustre 용량이 빠르게 소모됩니다.

이미 학습이 끝났거나 당분간 사용하지 않을 데이터까지 그대로 남겨두면 스토리지 부족과 I/O 성능 저하를 초래할 수 있습니다. 이를 방지하기 위해 Lustre 의 File Release 기능을 정기적으로 수행해서 용량을 확보합니다.

※Lustre에서의 File Release란?

A. Lustre의 Hierarchical Storage Management(HSM) 기능 중 하나

B. 파일의 메타데이터만 Lustre에 유지하고, 실제 데이터 블록은 S3로 반환

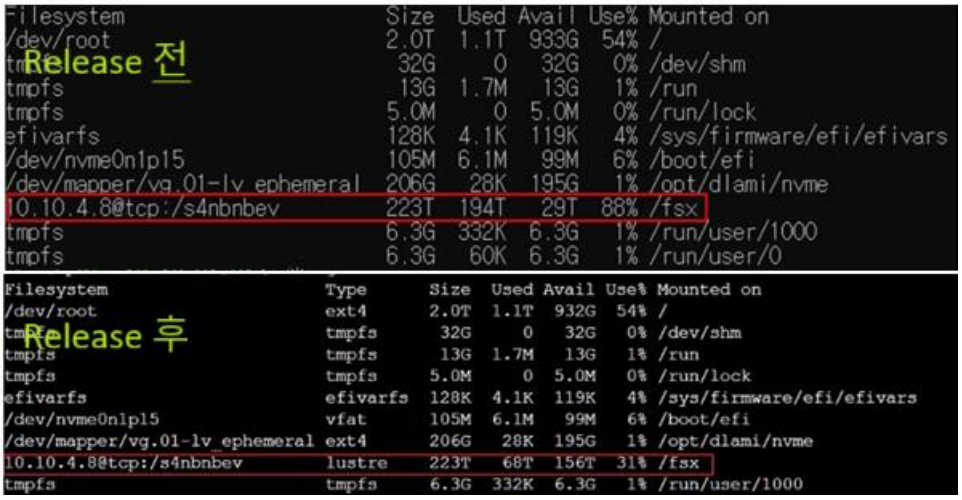
C. 사용자가 파일에 다시 접근할 경우, 필요한 시점에 S3에서 자동으로 데이터를 불러옴

File Release on Lustre Task의 운영상의 장점

A. 학습이 끝난 데이터는 Lustre 용량을 차지하지 않으면서도, 파일 목록은 그대로 유지되어 "존재하는 것처럼" 보이므로, Lustre의 데이터 저장 비용을 획기적으로 절감 가능

B. I/O 리소스를 불필요하게 점유하지 않으므로, 전체 워크로드 성능 저하를 예방.

즉, File Release 는 Lustre 를 "캐시 역할"로 유지하게 하여, 대규모 LLM 학습 워크로드에서 안정적인 스토리지 운영을 가능하게 합니다. File Release on Lustre Task 는 경로를 지정 후 일회성으로 동작하므로, 반복 작업이 필요한 경우 EventBridge 를 활용해서 원하는 시점에 Release 가 되도록 설정하면 됩니다.



[Release 이후 194T 에서 68T 로 Lustre 저장 용량 감소 확인]

2.3.5 Lustre 에서 EFA 활성화가 중요한 이유

EFA(Elastic Fabric Adapter)는 AWS 가 제공하는 저지연·고대역폭 네트워크 인터페이스로, HPC 와 ML 워크로 등을 위해 설계되었습니다. Lustre 에 EFA 를 활성화하면 다음과 같은 이점이 있기 때문에, 대규모 학습 환경에서는 EFA 를 꼭 활성화하도록 추천합니다.

FSx for Lustre 에 EFA 를 활성화하면 다음과 같은 장점을 얻을 수 있습니다.

- **낮은 지연 시간 (Low Latency)**

  A. Lustre 메타데이터 조회나 소규모 파일 접근과 같은 짧고 빈번한 I/O 요청에서 응답 속도를 크게 단축

  B. 파일 시스템 전반의 응답성을 높여, 애플리케이션 동작 시 체감 속도를 개선

- **높은 처리량 (High Throughput)**

  A. Lustre 의 특성상 데이터가 여러 OST(Object Storage Target)에 분산 저장됨

  B. EFA 는 이런 대규모 병렬 I/O 를 병목 없이 처리하여, 대용량 데이터셋을 빠르게 읽고 쓰는 데 최적화

- **대규모 분산 학습에 최적화**

  A. LLM 학습 환경에서는 GPU 노드 간 파라미터 동기화(NCCL 등) 와 Lustre 기반 데이터 접근이 동시에 발생

  B. EFA 는 이 두 경로의 네트워크 지연을 최소화하여, 전체 학습 속도를 안정적으로 유지하고 스케일 아웃 효율성을 보장

2.3.6 FSx for Lustre 메타데이터 Iops 성능 모니터링

대규모 학습에서 GPU 의 성능만큼이나 중요한 것이 스토리지입니다. 아무리 GPU 의 성능이 좋더라도 소스를 호출하는 스토리지에서 동시에 발생하는 처리량을 감당하지 못하면, 지연이 발생할 수밖에 없습니다. 이 때문에 상시로 Lustre 의 성능 모니터링을 하는 것은 대규모 학습 환경에서 중요한 운영 포인트입니다.

Lustre 의 성능 모니터링을 설명하기 앞서 먼저 Lustre 에서 처리량(Throughput)과 메타데이터 IOPS 가 별개로 나뉘어진 이유에 대해서 설명이 필요합니다.

Lustre 파일시스템은 **데이터 저장소(OST)** 와 **메타데이터 저장소(MDT)**를 분리해 관리하는 구조로 설계되어 있습니다.

1. OST (Object Storage Target)

A. 실제 파일 데이터 블록을 저장하는 장치

B. 대규모 학습 시 발생하는 연속적인 대용량 읽기/쓰기(Throughput)를 담당

C. OST 수와 성능이 늘어나면 전체 I/O 처리량도 비례적으로 증가

2. MDT (Metadata Target)

A. 파일 이름, 경로, 권한, 디렉토리 구조 같은 메타데이터를 관리

B. 대규모 학습 환경에서는 “수십만 개 파일을 동시에 열고 닫는” 메타데이터 요청이 매우 많음

C. 따라서 IOPS(초당 입출력 요청 수) 성능이 중요

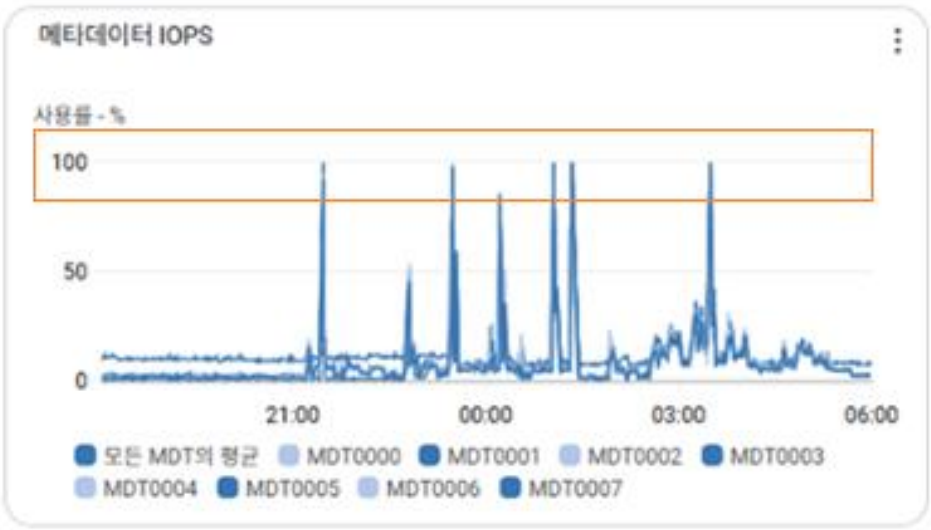
즉, 데이터 자체의 대역폭(Throughput) 과 메타데이터 처리 속도(IOPS)는 서로 다른 리소스를 사용하고, HPC 환경에서는 둘 다 병목이 될 수 있기 때문에 별도로 확장/최적화할 수 있게 분리된 것입니다.

UUID	bytes	Used	Available	Use%	Mounted on
s4nbnbev-MDT0000_UUID	549.9G	6.6G	543.3G	2%	/fsx[MDT:0]
s4nbnbev-MDT0001_UUID	549.9G	6.3G	543.5G	2%	/fsx[MDT:1]
s4nbnbev-MDT0002_UUID	549.9G	6.2G	543.7G	2%	/fsx[MDT:2]
s4nbnbev-MDT0003_UUID	549.9G	6.4G	543.5G	2%	/fsx[MDT:3]
s4nbnbev-MDT0004_UUID	549.9G	6.3G	543.6G	2%	/fsx[MDT:4]
s4nbnbev-MDT0005_UUID	549.9G	6.1G	543.8G	2%	/fsx[MDT:5]
s4nbnbev-MDT0006_UUID	549.9G	6.4G	543.5G	2%	/fsx[MDT:6]
s4nbnbev-MDT0007_UUID	549.9G	5.9G	544.0G	2%	/fsx[MDT:7]
s4nbnbev-MDT0008_UUID	549.8G	3.4G	546.4G	1%	/fsx[MDT:8]
s4nbnbev-MDT0009_UUID	549.8G	2.2G	547.6G	1%	/fsx[MDT:9]
s4nbnbev-MDT000a_UUID	549.8G	2.3G	547.5G	1%	/fsx[MDT:10]
s4nbnbev-MDT000b_UUID	549.8G	2.3G	547.6G	1%	/fsx[MDT:11]
s4nbnbev-MDT000c_UUID	549.9G	2.3G	547.5G	1%	/fsx[MDT:12]
s4nbnbev-MDT000d_UUID	549.8G	2.4G	547.4G	1%	/fsx[MDT:13]
s4nbnbev-MDT000e_UUID	549.8G	2.3G	547.5G	1%	/fsx[MDT:14]
s4nbnbev-MDT000f_UUID	549.8G	2.5G	547.4G	1%	/fsx[MDT:15]
s4nbnbev-OST0000_UUID	37.0T	29.1T	7.9T	79%	/fsx[OST:0]
s4nbnbev-OST0001_UUID	37.0T	29.1T	7.9T	79%	/fsx[OST:1]
s4nbnbev-OST0002_UUID	37.0T	28.8T	8.2T	78%	/fsx[OST:2]
s4nbnbev-OST0003_UUID	37.0T	29.1T	7.9T	79%	/fsx[OST:3]
s4nbnbev-OST0004_UUID	37.0T	29.2T	7.8T	79%	/fsx[OST:4]
s4nbnbev-OST0005_UUID	37.0T	30.7T	6.3T	84%	/fsx[OST:5]

[Lustre 에서 MDT 와 OST 영역의 분리와 사용률 현황]

실제 데이터가 저장되는 장치는 OST 이지만 메타데이터를 관리하는 MDT 에서 충분한 IOPS 를 보이지 않는다면, 스토리지 성능에 전체에 영향을 줄 수 있으므로, Lustre 의 OST 와 MDT 의 각 분리된 성능에 대해서 모니터링을 하는 것은 대규모 학습 환경에서 중요한 포인트입니다.

아래와 같이 메타데이터의 IOPS 가 순간 100%까지 발생하는 경우가 있다면, OST 의 처리량이 아닌 MDT 의 IOPS 를 증설해서 문제 해결을 해야 하며, 그 외에도 Lustre 의 성능 지표는 상시 모니터링이 필요합니다.



[Lustre 의 MDT IOPS 모니터링, 100% 사용률 탐지]

3 대규모 HPC 운영 노하우 및 운영 성과

3.1 대규모 HPC 인프라 운영에서 얻은 실전 노하우

3.1.1 ParallelCluster OOM issue 대응 방식

대규모 학습 중 메모리가 100%까지 차오르면 커널 OOM-Killer 가 동작하면서 임의의 프로세스를 강제 종료하게 되고, 이 과정에서 HeadNode–ComputeNode 간 Health Check 가 실패해 노드 전체가 재생성되는 문제가 발생할 수 있습니다. 이를 방지하기 위해 Ubuntu 환경에 earlyoom 패키지를 설치하여 메모리 사용률이 98%에 도달하기 전에 oom_score 기준으로 우선순위가 높은 프로세스를 선제적으로 종료하도록 설정했습니다.

> ※ **earlyoom:** Linux 환경에서 메모리 사용률이 임계치에 도달하기 전에 우선순위가 높은 프로세스를 선제적으로 종료하여 시스템 Hang을 예방하는 경량 데몬

기존 커널 OOM-Killer 는 메모리가 모두 소진된 후에야 동작하기 때문에 시스템이 Hang 에 빠질 위험이 있었지만, earlyoom 은 임계치 도달 전에 순차적으로 SIGTERM → SIGKILL 을 보내 안정성을 확보합니다. 실제 운영 과정에서도 여러 노드에서 메모리 사용량이 폭주하는 현상이 빈번하게 발생했는데, 이에 대한 대응으로 earlyoom 을 설치한 결과, 문제가 되는 프로세스들이 OS 가 멈추기 전에 사전에 종료되어 OS 자체의 장애를 예방할 수 있었습니다.

또한 CloudWatch + Lambda 연동을 통해 syslog 를 모니터링하고, earlyoom 이 프로세스를 종료하면 로그를 참조해서 해당 PID, 프로세스명, UID 정보를 Slack 으로 알림을 발송하게 되며, 연구원들은 문제 프로세스를 즉시 확인하고 설정해 둔 학습 Checkpoint 를 참조해서 학습을 빠르게 재개할 수 있도록 대응 체계를 마련했습니다.

3.1.2 클러스터 운영 시 자동 업데이트 비활성화 권장

Unattended-Upgrade 는 Ubuntu/Debian 계열에서 보안 패치나 패키지 업데이트를 자동으로 설치해주는 기능입니다. 운영자가 직접 명령을 내리지 않아도 정해진 주기에 따라 시스템이 알아서

업데이트를 수행합니다. 보안 측면에서는 관리 편의성이 크지만, HPC 클러스터 환경에서는 이 옵션을 비활성화할 것을 권장합니다.

GPU 환경에서 비활성화해야 하는 이유

- GPU 클러스터와 같은 HPC 환경에서는 예상치 못한 커널 업데이트 시 문제 발생 가능성이 높음
- 커널이나 드라이버가 자동으로 갱신되면 CUDA, NVIDIA 드라이버, 라이브러리 간 버전 충돌이 발생
- 분산 학습 환경에서는 모든 노드의 버전이 동일해야 하는데, 자동 업데이트는 이 일관성을 깨뜨릴 위험

따라서 GPU 기반 HPC 환경에서는 Unattended-Upgrade 를 비활성화하고, 관리자가 직접 검증된 시점에 수동으로 업데이트를 적용하는 것이 가장 안전합니다. 혹시 해당 설정이 활성화됐다면, 비활성화를 추천합니다.

3.1.3 인스턴스 스토어의 효율적인 활용 제안

이번 프로젝트에서 활용한 P5en 인스턴스는 노드당 30.4TB NVMe SSD 인스턴스 스토어를 제공합니다.

인스턴스 스토어는 인스턴스 로컬에 직접 연결된 초고속 스토리지로, 지연 시간과 처리량 측면에서 Lustre 보다 뛰어난 I/O 성능을 발휘합니다.

다만, 인스턴스를 중지 후 다시 시작하거나 종료할 경우 데이터가 모두 삭제된다는 특성이 있습니다.

따라서 인스턴스 스토어는 캐시 성격의 데이터를 저장하는 용도로 사용하는 것이 적합합니다.

원본 학습 데이터셋, 체크포인트, 최종 산출물과 같이 반드시 보존해야 하는 데이터는 Lustre 나 S3 같은 영구 스토리지에 저장해야 하고,

토크나이징 등 데이터셋 전처리 캐시, 학습 과정에서 생성되는 Dataloader 캐시처럼 필요 시 재생성 가능한 데이터는 인스턴스 스토어에 저장하는 것이 효율적입니다.

이렇게 운영하면 Lustre 의 I/O 병목을 줄이고, 인스턴스 스토어의 초고속 NVMe 성능을 캐시 용도로 활용하여 학습 성능과 비용 효율성을 동시에 높일 수 있습니다.

3.1.4 AWS PHD 이벤트 기반 인스턴스 장애 대응 체계

AWS Personal Health Dashboard(PHD)는 계정에 직접 영향을 미치는 이벤트(예: EC2 인스턴스 리타이어먼트, 하드웨어 장애 가능성, 예정된 유지보수 등)를 알려줍니다. 운영팀은 이를 단순히 콘솔에서 확인하는 데 그치지 않고, EventBridge + SNS 연동을 통해 별도의 관리 이메일로도 자동 수신할 수 있도록 구성했습니다.

특히 GPU 클러스터 환경에서는 아래의 이유로 해당 설정이 더욱 중요합니다.

- GPU 인스턴스는 고온·고전력 상태로 장시간 동작하기 때문에 CPU 기반 인스턴스보다 물리적 장애 발생 가능성이 높습니다.
- LLM 학습은 수일~수주 단위로 실행되므로, 장애 발생 시 학습 Job 전체가 중단될 수 있습니다.
- PHD 이벤트를 실시간 알림으로 받아 즉시 대응하지 않으면, 수천 GPU-시간 이상의 자원 낭비로 이어질 수 있습니다.

이 구성을 통해 인스턴스 이상 이벤트가 감지되면 관리자 이메일로 즉시 알림을 받아 빠르게 대응할 수 있었고, 장시간 학습 환경에서 불필요한 중단을 최소화할 수 있었습니다.

3.1.5 Lustre Maintenance Window 시간 지정의 필요성

Lustre 는 AWS 가 자동으로 보안 패치와 업데이트를 적용하기 때문에, Maintenance Window 를 지정하지 않으면 예기치 못한 시간대에 작업이 발생할 수 있습니다. 이를 방지하기 위해 관리자가 직접 유지보수 시간을 지정하는 것이 안전합니다. 연구 환경의 특성에 따라, Job 이 몰리지 않는 주말이나 새벽 시간대를 선택하기도 하고, 반대로 문제 발생 시 즉시 대응할 수 있도록 평일 업무 시간에 맞춰 두는 경우도 있습니다.

3.2 운영 성과

이번 프로젝트에서 중요한 목표는 단순히 GPU 클러스터를 빠르게 구축하는 것이 아니라, 대규모 LLM 학습을 장기간 안정적으로 운영할 수 있는 환경을 제공하는 것이었습니다.

이를 위해 ParallelCluster 의 여러 옵션을 고정 값으로 설정하여 자원을 안정적으로 확보하고, 필요할 때만 잠깐 사용하는 AutoScaling 기반의 유동적 방식 대신 항상 동일한 수량의 GPU 노드를 운영하도록

구성했습니다.

운영 과정에서 특히 중점을 둔 부분은 다음과 같습니다.

- Slurm 스케줄링 최적화: 단순 파티션만으로는 강제력이 부족하기 때문에, 사용자·계정 단위로 자원 사용량을 추적하고 QOS 정책을 강제 적용할 수 있도록 Slurm Accounting 을 도입
  > 사용자별 Job 제출 개수, 동시 실행 개수, 최대 실행 시간 등을 제한하여 공정성 보장
  > DB 기반 기록을 통해 자원 사용 내역을 투명하게 관리하고, 향후 리포트/분석에도 활용
- Lustre 성능 최적화 및 EFA 활용: OST 와 MDT 성능을 분리 관리하고, EFA 활성화를 통해 저지연, 고대역폭 네트워크 성능을 확보하여 대규모 데이터셋 I/O 병목 최소화
- S3 + File Release 기반 용량 관리: 불필요한 데이터는 File Release 로 S3 에 반환하고, Lazy Loading 을 통해 GPU 자원 할당과 동시에 학습을 시작할 수 있도록 지원

운영 안정성 확보:

- earlyoom 도입으로 메모리 폭주 시 OS Hang 을 사전에 차단
- PHD + EventBridge + SNS 연동으로 인스턴스 이상 이벤트를 실시간 감지 및 대응
- Maintenance Window 지정으로 예기치 못한 패치에 따른 이슈 대응 체계 수립

대규모 LLM 학습 환경에서는 단순히 GPU 성능만 확보한다고 안정적인 학습이 보장되지는 않습니다.

네트워크, 스토리지, 스케줄러, 운영 자동화까지 함께 고려해야만 장시간 이어지는 학습을 안정적으로 완수할 수 있습니다.

이번 운영 사례를 통해, ParallelCluster 의 고정 자원 운영, Lustre 최적화, S3 연계, 모니터링 체계가 결합된 결과 대규모 학습 환경에서의 안정성을 달성할 수 있었습니다.

대규모 LLM 학습과 다양한 연구 환경에 대한 지원이 필요하시면 디딤에 문의해 주시기 바랍니다.
이러한 운영 경험과 노하우를 바탕으로 안정적이고 효율적인 서비스를 제공해 드리겠습니다.

- **문의하기:** https://didimgpu.com/consulting/

Didim

AI · Big Data · Cloud Specialist

디딤 문의하기

Tel 02-857-0365

Email cloud@didim.com

[08389] 서울시 구로구 디지털로26길 43, R동 12층 (구로동, 대륭포스트 8차)

Tel | 02-857-0365 **E-mail** | cloud@didim.com **Web** | didim.com