UAB
Universitat Autònoma de Barcelona

Dipòsit digital de documents de la UAB

This is the **published version** of the bachelor thesis:

Planelles Samper, Bernat. *Anàlisi i visualització de patrons d'entrada/sortida a aplicacions Deep Learning*. Treball de Final de Grau (Universitat Autònoma de Barcelona), 2026 (Grau en Enginyeria de Dades)

This version is available at https://ddd.uab.cat/record/337357

under the terms of the  license.

Anàlisi i visualització de patrons d'entrada/sortida en aplicacions de Deep Learning)

Bernat Planelles Samper

Resum– Aquest treball aborda l'anàlisi i la visualització de patrons d'entrada/sortida en aplicacions de Deep Learning executades en entorns HPC. L'objectiu principal és facilitar la identificació de colls d'ampolla associats a la transferencia de dades en aplicaciones Deep Learning mitjançant el processament de traces generades amb Darshan. Per fer-ho, es defineix un model de dades a partir dels registres Perf, Parser i DXT de Darshan, i es desenvolupa un dashboard interactiu en Python per explorar mètriques globals, comptadors POSIX/LUSTRE i esdeveniments individuals d'I/O. Els resultats mostren que la visualització facilita la detecció d'accessos paral·lels, desequilibris entre processos i variacions de latència, evidenciant la seva utilitat per comprendre i optimitzar el rendiment d'I/O en càrregues de Deep Learning sobre HPC.
Paraules clau– Deep Learning, HPC, I/O, pipeline, Darshan, DXT, Lustre, visualització de dades, dashboard

Abstract– This work addresses the analysis and visualization of input/output patterns in Deep Learning applications executed in HPC environments. The main objective is to facilitate the identification of bottlenecks associated with data transfer in Deep Learning applications through the processing of traces generated with Darshan. To do this, a data model is defined from Darshan's Perf, Parser and DXT records, and an interactive Python dashboard is developed to explore global metrics, POSIX/LUSTRE meters and individual I/O events. The results show that visualization facilitates the detection of parallel accesses, imbalances between processes and latency variations, demonstrating its utility to understand and optimize the performance of I/O in Deep Learning loads on HPC.
Keywords– Deep Learning, HPC, I/O, pipeline, Darshan, DXT, Lustre, data visualization, dashboard

1 INTRODUCCIÓ

El creixement del Deep Learning en entorns de computació HPC ha transformat la forma en què es dissenyen, executen i optimitzen les aplicacions Deep Learning. Tradicionalment, els sistemes HPC s'han emprat per executar simulacions numèriques, resoldre models físics complexos o processar grans volums de dades mitjançant aplicacions paral·leles. No obstant això, en els últims anys, l'ús de tècniques d'aprenentatge profund ha introduït noves càrregues de treball que combinen entrenament distribuït, grans volums de dades i pipelines complexos d'entrada/sortida.

En aquest context, el rendiment d'una aplicació de Deep Learning no depèn únicament de la capacitat de càlcul dels seus processadors, que continuen sent essencials per a accelerar les operacions computacionals de l'entrenament, s'afegeix el sistema d'emmagatzematge i la transferència de dades com factors determinants del rendiment global. Una aplicació pot disposar de gran capacitat computacional, però si les dades no es transfereixen als nodes amb suficient velocitat, els recursos romanen limitats. Per això, l'anàlisi de les operacions d'entrada/sortida, o I/O, s'ha convertit en un element clau per a comprendre i optimitzar el comportament d'aplicacions de Deep Learning en HPC.

1.1 Context del Deep Learning en HPC

El Deep Learning aplicat a HPC sorgeix principalment de la necessitat d'entrenar models cada vegada més complexos sobre datasets massius i heterogenis. En molts casos, tant la mida dels models com el volum de dades superen la capacitat d'una màquina individual, la qual cosa obliga a distribuir l'entrenament entre múltiples nodes i processos.

Les aplicacions de Deep Learning presenten una naturalesa híbrida: d'una banda, hereten característiques pròpies de l'aprenentatge automàtic, com l'ús de mini-batches, shuffling, validació periòdica i checkpointing; per un altre, s'executen sobre infraestructures HPC amb xarxes d'al-

• E-mail de contacte: bernat.planelles@gmail.com
• Treball tutoritzat per: Betzabeth del Carmen Leon Otero (Departament d'Arquitectura de Computadors i Sistemes Operatius)
• Curs 2025/26

ta velocitat, sistemes d'arxius paral·lels i gestors de treballs com Slurm. Aquesta combinació genera un ecosistema complex en el qual frameworks com TensorFlow, PyTorch o Keras interactuen amb biblioteques tradicionals de HPC, instruccions POSIX, middleware MPI-IO o sistemes d'arxius com a Lustre. Aquesta interacció entre frameworks de Deep Learning i capes tradicionals d'I/O en HPC ha estat identificada com una de les causes principals de la complexitat en l'anàlisi del rendiment [1, 3].

El cicle de vida d'una aplicació de Deep Learning genera operacions d'I/O en pràcticament totes les seves fases. Durant la generació de dades, els datasets poden crear-se a partir de simulacions, sensors, experiments o fonts externes. En la preparació de dades, es fan tasques de neteja, transformació, normalització, partició, agregació i augment de dades. Durant l'entrenament, l'aplicació llegeix mostres, construeix mini-batches, randomitza les dades, valida el model i escriu checkpoints. Finalment, en la fase d'inferència, es carreguen models prèviament entrenats, es llegeixen noves mostres i s'emmagatzemen o transmeten prediccions.

Aquesta dinàmica converteix al pipeline de dades en un component tan rellevant com el mateix còmput. En un entrenament típic, cada iteració requereix llegir un mini-batch, transferir-lo a memòria, possiblement preprocesarlo, enviar-lo a la GPU, calcular la pèrdua, obtenir gradients i actualitzar els pesos del model. Si la lectura o preparació no segueix el ritme del càlcul, l'entrenament s'alenteix i els processadores queden infrautilitzats. Per tant, optimitzar el flux de dades és essencial per a aprofitar adequadament els recursos en sistemes HPC.

1.2 Problemes dels I/O en aquests entorns

Un dels principals problemes de les aplicacions de Deep Learning en HPC és que els seus patrons d'I/O difereixen significativament dels patrons tradicionals de les aplicacions científiques clàssiques. En HPC tradicional, moltes aplicacions generen grans escriptures seqüencials, checkpoints de simulació o fitxers de sortida relativament estructurats. En canvi, les càrregues de Deep Learning solen estar dominades per lectures repetides, petites, aleatòries i distribuïdes sobre grans datasets.

L'organització del dataset és un factor crític. En les aplicaciones de DL on cada arxiu representa una mostra, habitual en dominis com a imatge, àudio o uns certs formats NumPy, però pot generar milions d'arxius petits. Aquesta situació pressiona fortament als servidors de metadades, especialment en sistemes d'arxius paral·lels com a Lustre. Per contra, formats com HDF5 o TFRecord permeten emmagatzemar múltiples mostres en un mateix arxiu, reduint la pressió sobre metadades, encara que introdueixen altres reptes com l'accés a subconjunts i alineació de transferències. L'elecció del format de dades i de l'estratègia d'accés pot modificar significativament el nombre d'operacions, la pressió sobre metadades i l'amplada de banda efectiva [12].

Els sistemes d'arxius paral·lels tampoc es comporten igual davant tots els patrons d'accés. Lustre, per exemple, està optimitzat per a arxius grans i operacions contigües, però pot sofrir amb molts arxius petits, accessos aleatoris o càrregues intensives en metadades. NFS, encara que més simple, escala pitjor quan molts processos accedeixen de manera concurrent a datasets compartits. Per això, una aplicació que escala correctament des del punt de vista computacional pot deixar d'escalar quan el sistema d'emmagatzematge no és capaç d'absorbir la concurrència generada pels processos d'entrenament.

Un altre problema habitual és la grandària de les operacions d'I/O. Les operacions petites i freqüents augmenten el número de crides al sistema, eleven la latència acumulada i redueixen l'ampla de banda efectiu. En diverses càrregues de Deep Learning, transferències petites com 256 KiB poden generar un nombre excessiu d'operacions, mentre que transferències majors, per exemple d'1 MiB, poden alinearse millor amb el stripe size del sistema d'arxius i millorar el rendiment. Això demostra que no n'hi ha prou amb mesurar el volum total de dades transferit: també és necessari analitzar la granularitat de les operacions. En sistemes HPC amb emmagatzematge paral·lel, paràmetres com el nombre d'OSTs, el stripe count i el stripe size poden tenir un impacte directe sobre el rendiment de l'entrenament distribuït [2].

El *checkpointing* afegeix una altra font rellevant d'I/O, en aquest cas per les escriptures. A mesura que els models creixen, també augmenta la frequencia i mida dels *checkpoints*. Si aquests s'escriuen amb massa freqüència, poden introduir pauses significatives i afectar el rendiment global de l'entrenament. El checkpointing és una font rellevant d'escriptures en aplicacions de Deep Learning, especialment quan els models creixen i l'estat de l'entrenament s'ha de guardar periòdicament [3].

Finalment, augmentar el paral·lelisme no sempre resol el problema. Més processos poden incrementar el nombre d'operacions d'I/O encara que el volum total de dades no canviï. De la mateixa manera, augmentar el número de OSTs o stripes en Lustre pot millorar el rendiment en patrons regulars i seqüencials, però també pot empitjorar-lo quan els accessos són irregulars, desbalancejats o dominats per metadades. En conseqüència, l'optimització d'I/O requereix conèixer el patró real d'accés de l'aplicació.

1.3 Rellevància de l'anàlisi de patrons d'I/O

L'anàlisi de patrons d'I/O és fonamental perquè el rendiment d'una aplicació no depèn únicament de la quantitat de dades que llegeix o escriu. Dues aplicacions que transfereixen el mateix volum de dades poden tenir comportaments molt diferents si una realitza poques operacions grans i seqüencials, mentre que l'altra realitza moltes operacions petites, aleatòries i distribuïdes entre milers d'arxius. Per aquest motiu, és necessari estudiar tant la dimensió espacial com la dimensió temporal de les operacions d'I/O.

L'anàlisi espacial permet observar quines regions dels arxius són accedides per cada procés. A través d'offsets, longituds d'operació i distribució entre processos, és possible identificar si els accessos són contigus, aleatoris, solapats o fragmentats. També permet comprovar si les dades estan correctament alineats amb els stripes o OSTs. Aquesta informació és essencial per a determinar si el layout del dataset afavoreix o perjudica la concurrència [10, 11].

L'anàlisi temporal permet estudiar quan ocorren les operacions. Gràcies als timestamps d'inici i fi, es poden identificar ràfegues de lectura, fases d'escriptura de checkpoints, operacions de metadades i possibles períodes d'es-

pera. També permet analitzar si l'I/O se solapa amb el còmput o si totes dues fases s'executen de manera seqüencial.

Un avantatge important de l'anàlisi de patrons és que ajuda a distingir si el coll d'ampolla s'origina en l'aplicació o en el sistema d'emmagatzematge d'arxius. Si els processos accedeixen a regions varies i no solapades, el patró generat per l'aplicació pot considerar-se paralelizable. Si apareix serialització o baix rendiment, la causa pot estar en el sistema d'emmagatzematge, en la configuració de striping, en els servidors de metadades o en la capacitat del backend per a atendre sol·licituds concurrents. Aquesta distinció és rellevant perquè les solucions serien diferents: modificar el patró d'accés de l'aplicació, canviar el layout de dades o ajustar la configuració del sistema d'arxius.

Les eines de profiling i tracing permeten convertir aquestes observacions en mesuraments concrets. Darshan proporciona comptadors i mètriques agregades de manera lleugera i escalable; Darshan DXT permet obtenir traces detallades d'operacions POSIX; Vampir ofereix línies temporals completes per procés; TensorFlow Profiler permet observar parts del pipeline d'entrenament; i eines complementàries com seff, VaniDL o informes MPI-IO ajuden a relacionar I/O, CPU, memòria i configuració del sistema [6, 8].

En resum, l'anàlisi de patrons permet passar d'una optimització basada en intuïcions a una optimització basada en evidència. A partir de les traces i mètriques és possible decidir si convé canviar de format, augmentar la grandària de transferència, modificar el chunking, ajustar el batch size, reduir el número de OSTs, activar collective buffering, usar caching, aplicar prefetching o reorganitzar el dataset. Atès que no existeix una configuració universal òptima, mesurar el patró real d'accés és el primer pas per a millorar el rendiment.

1.4 Motivació i objectius

En aquest treball s'aborda el problema d'analitzar els patrons d'I/O d'aplicacions DL en entorns HPC. Es proposa un mètode per a processar i visualitzar les traces d'I/O generades per aquestes aplicacions, amb la finalitat d'identificar colls d'ampolla i comportaments ineficients de manera més clara i ràpida. La motivació principal és, donada la complexitat del programari d'I/O en DL sobre HPC, disposar d'eines visuals que sintetitzin els patrons temporals i espacials d'accés a dades per guiar als usuaris en l'optimització del rendiment I/O de les seves aplicacions. Els objectius inclouen:

- Dissenyar un model de dades per a organitzar aquesta informació.
- Desenvolupar una eina interactiva de visualització que mostri mètriques i patrons clau.
- Validar aquesta eines amb registres generats en entorns de HPC reals.

Aquest enfocament segueix la línia de treballs recents que proposen processar traces d'I/O i transformar-les en representacions analítiques i visuals per estudiar el rendiment de les aplicacions DL [11].

2 Fonaments teórics i estat de l'art

2.1 Què són les operacions d'I/O en aplicacions DL

En aplicacions de Deep Learning, les operacions d'I/O comprenen tots els moviments de dades entre emmagatzematge persistent, memòria principal, CPU, GPU, xarxa i sistema d'arxius. Aquestes operacions inclouen la lectura de datasets, l'obertura i tancament d'arxius, la consulta de metadades, l'escriptura de checkpoints, la càrrega de models, la lectura de mostres per a inferència i l'emmagatzematge de resultats o prediccions. Encara que moltes vegades s'associen únicament amb lectures i escriptures, en la pràctica l'I/O inclou una cadena més àmplia d'operacions que travessa múltiples capes del sistema i poden afectar directament la utilització de GPU i l'eficiència global de l'entrenament [3, 1].

Durant la fase de generació de dades, les operacions d'I/O estan relacionades amb la creació o recopilació de datasets. Aquestes dades poden procedir de simulacions científiques, sensors, experiments, scraping, bases de dades o repositoris públics. Un cop generats, les dades han d'emmagatzemar-se en sistemes d'arxius paral·lels, SSDs locals, emmagatzematge remot, data lakes o plataformes cloud.

En la fase de preparació, les aplicacions realitzen operacions de lectura, transformació i escriptura. Aquesta preparació pot ser offline, quan s'executa abans de l'entrenament, o en línia, durant l'entrenament o la inferència. Les tasques habituals inclouen neteja, filtrat, normalització, discretización, agregació, partició en conjunts d'entrenament/validació/prova i data augmentation. Cadascuna d'aquestes tasques pot modificar el patró d'I/O i afectar el rendiment posterior de l'entrenament.

Durant l'entrenament, l'operació dominant sol ser la lectura repetida de mostres. El model consumeix mini-batches durant múltiples èpoques, i les dades poden estar randomitzades per a millorar la qualitat estadística de l'entrenament. En entorns distribuïts, diversos processos accedeixen simultàniament a diferents particions del dataset. Segons el format i l'organització de les dades, aquests accessos poden dirigir-se a un arxiu compartit, a múltiples arxius independents o a una combinació de tots dos.

El checkpointing constitueix una operació d'escriptura característica de Deep Learning. El seu objectiu és guardar l'estat de l'entrenament per a permetre recuperació davant fallades, reutilització del model, fini-tuning o anàlisi posterior. Un checkpoint pot incloure pesos del model, estat del optimizador, metadades i fitxers auxiliars. En models grans, el cost d'escriptura pot ser considerable, per la qual cosa la freqüència de checkpointing ha de ser equilibarada.

En inferència també existeixen operacions d'I/O rellevants. El sistema ha de carregar el model entrenat, llegir o rebre noves mostres, realitzar possibles transformacions en línia i emmagatzemar o transmetre prediccions. En inferència offline poden predominar lectures grans i contigües, mentre que en inferència en línia o streaming la latència de xarxa i la velocitat de resposta poden ser més importants que l'amplada de banda agregada.

Des del punt de vista de baix nivell, moltes d'aquestes operacions es tradueixen en crides POSIX com open, clo-

se, read, write, stat, lseek, pread o pwrite. En aplicacions paral·leles, MPI-IO proporciona abstraccions addicionals perquè diversos processos accedeixin a un fitxer comú, defineixin vistes lògiques de l'arxiu i utilitzin operacions col·lectives o independents. Per sobre d'aquestes capes, formats com HDF5, NetCDF, TFRecord o NPZ defineixen com s'organitzen les dades i quins patrons d'accés es generen. Els formats de dades condicionen la granularitat dels accessos i la manera com les mostres són llegides pels processos d'entrenament [12].

2.2 Arquitectura dels sistemes HPC i generació d'I/O

Un sistema HPC modern està compost per nodes de còmput, xarxes d'interconnexió d'alta velocitat i sistemes d'emmagatzematge paral·lel. Els nodes de còmput executen els processos de l'aplicació, que poden estar distribuïts mitjançant MPI, Horovod o altres mecanismes de paral·lelisme. La xarxa connecta aquests nodes amb els servidors d'emmagatzematge, mentre que el sistema d'arxius paral·lel proporciona accés concurrent a grans volums de dades.

El stack d'I/O en HPC pot entendre's com una pila de capes. En la part superior es troba l'aplicació, que en el cas de Deep Learning sol estar desenvolupada sobre frameworks com TensorFlow o PyTorch. Per sota apareixen biblioteques d'alt nivell o formats científics com HDF5, NetCDF, ADIOS, pNetCDF o SIONLib. En una capa intermèdia se situa MPI-IO, que permet realitzar operacions paral·leles i aplicar optimitzacions. Més a baix es troba POSIX, que proporciona operacions bàsiques d'accés a fitxers. Finalment, el sistema d'arxius paral·lel i el maquinari d'emmagatzematge executen físicament les operacions.

La generació d'I/O en aplicacions paral·leles depèn de com es distribueixen les dades entre processos. En alguns casos, cada procés escriu o llegeix el seu propi arxiu. Aquesta estratègia és senzilla i evita coordinació complexa, però pot produir massa arxius i saturar els servidors de metadades. En altres casos, tots els processos accedeixen a un únic arxiu compartit, escrivint o llegint regions diferents. Aquesta estratègia redueix el nombre d'arxius, però requereix coordinació i un sistema d'arxius capaç de gestionar concurrència. També existeixen enfocaments intermedis, com l'ús d'escriptors col·lectius, on només un subconjunt de processos realitza les operacions físiques d'I/O.

Els sistemes d'arxius paral·lels es dissenyen per a escalar capacitat, amplada de banda i concurrència. Una tècnica central és el striping, que divideix les dades d'un arxiu entre diversos dispositius o servidors. En Lustre, per exemple, les metadades són gestionades per servidors específics, mentre que les dades s'emmagatzemen en Object Storage Targets. Paràmetres com stripe count, stripe size i stripe offset determinen com es distribueix un arxiu entre els OSTs. L'impacte d'aquests paràmetres depèn del patró d'accés, del nombre de processos i de la distribució física dels fitxers en el sistema d'emmagatzematge [2].

No obstant això, la configuració òptima depèn del patró d'accés. Per a arxius compartits grans i accessos seqüencials, pot ser beneficiós distribuir l'arxiu entre diversos OSTs. Per a molts arxius petits, sol ser preferible reduir el stripe count per a evitar sobrecàrrega innecessària. En aplicacions de Deep Learning amb patrons irregulars, shuffling o accessos petits, augmentar el número de OSTs no sempre millora el rendiment i pot fins i tot introduir més coordinació i contenció.

Els formats de dades també formen part de l'arquitectura efectiva d'I/O. HDF5 permet organitzar dades científiques en grups, datasets, metadades i chunks, i pot utilitzar MPI-IO per a accés paral·lel. TFRecord està orientat a pipelines TensorFlow i pot oferir bon rendiment amb transferències grans. NPZ és simple i còmode en entorns Python, però menys adequat per a entrenament distribuït intensiu en I/O. Formats com CSV, JSON, JPEG o PNG són fàcils d'utilitzar, però poden produir sobrecàrrega per text, compressió, descodificació o gran quantitat d'arxius petits.

2.3 Eines d'anàlisis: Darshan, strace, etc

L'anàlisi del rendiment d'I/O pot abordar-se mitjançant dues estratègies principals: profiling i tracing. El profiling recopila comptadors i mètriques agregades, com a nombre d'operacions, bytes transferits, temps total d'I/O o grandària mitjana d'accés. Sol ser menys intrusiu i més escalable. El tracing, d'altra banda, registra esdeveniments individuals amb marques temporals, la qual cosa permet reconstruir línies temporals i analitzar amb detall la seqüència d'operacions. El seu principal inconvenient és que genera més volum de dades i pot introduir major sobrecàrrega. El profiling proporciona una visió agregada i escalable del comportament d'I/O, mentre que el tracing permet reconstruir esdeveniments individuals amb informació temporal i espacial [10, 11].

Les eines de tracing destaca Darshan, àmpliament utilitzada per a instrumentar aplicacions HPC i registrar la seva activitat I/O; eines de monitoratge pròpies de sistemes d'arxius paral·lels (per exemple, eines de Lustre); i registres de treballs del scheduler (com els logs de Slurm, que reporten l'ús de recursos). Addicionalment, hi ha eines de profiling generals com PyTorch Profiler, TensorBoard o NVIDIA Nsight Systems, que es centren en el rendiment de l'aplicació (CPU/GPU), però poden oferir informació d'I/O d'alt nivell. A nivell de sistema operatiu, també existeixen utilitats clàssiques d'anàlisis de peticions i rendiment, entre elles strace (traça de crides al sistema, útil per a capturar totes les operacions d'arxiu realitzades per un procés), perf (perfilat d'esdeveniments de rendiment, incloses comptadors d'I/O), o dstat. Darshan és una eina àmpliament utilitzada en HPC per caracteritzar l'I/O d'aplicacions paral·leles sense necessitat de modificar el codi font [6].

Darshan permet capturar informació d'aplicacions paral·leles sense modificar el codi font. Pot interceptar operacions POSIX i MPI-IO i proporciona mètriques com el nombre de lectures i escriptures, bytes transferits, grandàries d'accés, temps dedicat a I/O, nombre d'arxius, accés a fitxers compartits o independents i variància entre processos.

El modul DXT de Darshan amplia aquesta anàlisi mitjançant traces detallades d'operacions POSIX. Registra informació com a tipus d'operació, offset, longitud, temps d'inici, temps de fi, procés implicat i arxiu accedit. Això permet estudiar patrons espacials i temporals amb major precisió. Per exemple, pot revelar si diversos processos accedeixen a regions distintes d'un arxiu compartit, si les operacions se serialitzen, si hi ha ràfegues de lectura o si

predominen operacions petites.

Els móduls Parser i Perf de Darshan, permeten extreure mètriques agregades. Darshan Parser proporciona comptadors POSIX, bytes llegits i escrits, grandàries màximes d'accés i temps de lectura o escriptura. Darshan Perf resumeix informació de rendiment, com a temps d'I/O per a arxius únics o compartits i rendiment basat en el procés més lent. La combinació d'aquestes eines permet construir una visió integrada del comportament d'I/O.

En entorns gestionats per Slurm, **seff** pot complementar l'anàlisi proporcionant informació sobre l'eficiència del treball. Entre les seves mètriques s'inclouen ús de CPU, eficiència de CPU, ús de memòria, eficiència de memòria, nombre de nodes, nuclis per node i temps total d'execució. Aquesta informació és útil per a detectar si un problema d'I/O està afectant indirectament l'aprofitament dels recursos computacionals.

Strace permet inspeccionar les crides al sistema realitzades per un procés, incloent operacions com `open`, `read`, `write`, `close` o `stat` [7]. Encara que pot ser útil per a depuració puntual o anàlisi local, no és l'eina més adequada per a caracteritzar aplicacions HPC distribuïdes a gran escala.

2.4 Mètriques habituals i limitacions actuals

El rendiment d'I/O en aplicacions de Deep Learning sobre HPC s'avalua mitjançant un conjunt de mètriques complementàries. Una de les més bàsiques és el temps total d'I/O, que indica quant temps dedica l'aplicació a operacions de lectura o escriptura. Aquesta mètrica sol comparar-se amb el temps total d'execució per a estimar el percentatge de temps perdut o invertit en la transferencia de dades. Un percentatge elevat de temps en I/O pot indicar que l'emmagatzematge està limitant l'aplicació [11, 10].

L'**ampla de banda** mesura la taxa efectiva de transferència de dades i sol expressar-se en MiB/s, GiB/s o GB/s. És una mètrica fonamental per a comparar formats, sistemes d'arxius i configuracions. No obstant això, ha d'interpretarse amb cura: una amplada de banda alta pot ocultar desequilibris entre processos, i una amplada de banda baixa pot deure's a operacions petites, accessos aleatoris o sobrecàrrega de metadades, no necessàriament a una limitació física de l'emmagatzematge.

El **nombre total d'operacions de lectura i escriptura** permet detectar granularitat excessiva. Si una aplicació realitza milions d'operacions petites per a transferir un volum relativament moderat de dades, probablement està desaprofitant el sistema d'arxius paral·lel.

Els **bytes llegits i escrits** permeten mesurar el volum total de transferència. Aquesta mètrica pot comparar-se amb la grandària física del dataset per a detectar sobrelectura, accessos repetits o transferència addicional de metadades. En checkpointing, els bytes escrits depenen de la grandària del model i de la freqüència de guardat. Aquesta mètrica és especialment important en models grans, on els checkpoints poden ocupar diversos gigaoctets o més.

Els **offsets** i les **longituds** d'operació són essencials per a l'anàlisi espacial. Permeten identificar si els processos accedeixen a regions contigües, aleatòries, solapades o fragmentades.

Les mètriques temporals permeten reconstruir l'ordre de les operacions. Entre elles s'inclouen **temps d'inici, temps de fi, durada de cada operació i distribució d'esdeveniments al llarg de l'execució.** Aquestes mètriques ajuden a identificar ràfegues d'I/O, fases separades de còmput i lectura, escriptura periòdica de checkpoints o períodes d'espera.

Una altra mètrica important és la distribució d'I/O per procés. En aplicacions distribuïdes, el rendiment global pot quedar limitat pel procés més lent. Per això, és necessari analitzar la variància entre processos, la quantitat de dades transferida per cadascun i el temps dedicat a I/O de manera individual. Un desbalance pot indicar particionado incorrecte del dataset, accés desigual a arxius o problemes de configuració.

Les operacions de metadades son especialment rellevant per evaluar els rendiments de les aplicacions DL. Inclouen obertura i tancament d'arxius, creació, eliminació, consulta d'atributs i llistat de directoris. En datasets amb molts arxius petits, aquestes operacions poden saturar el servidor de metadades abans que l'amplada de banda de dades arribi al seu límit.

En sistemes Lustre, també han d'analitzar-se mètriques específiques del sistema d'arxius, com a número de OSTs, stripe count, stripe size, stripe offset, distribució d'accessos entre OSTs i temps associat a metadades.

Encara que existeixen nombroses mètriques per a avaluar el rendiment d'I/O, l'estat actual de les eines presenta limitacions importants: molts benchmarks de DL es centren en el rendiment computacional i no caracteritzen amb suficient detall els patrons d'accés a dades. La majoria d'estudis s'enfoca en la fase d'entrenament, mentre que la inferència en HPC, especialment en escenaris en línia o de *streaming*, requereix analitzar aspectes com a latència, xarxa, balanceig de càrrega, càrrega de models i escriptura de prediccions. A això se suma que moltes conclusions depenen de l'entorn experimental, provocant que no existeix una única mètrica ni configuració òptima per a tots els casos: l'avaluació ha de combinar temps d'I/O, amplada de banda, IOPS, grandària d'operació, metadades, distribució per procés, offsets, timestamps, configuració del sistema d'arxius i eficiència de còmput, seguint una metodologia basada en primer identificar el patró real d'accés i ajustar després format del dataset.

3 Disseny metodològic i model de dades

La metodologia seguida en aquest treball es centra en transformar les traces generades per Darshan en estructures de dades adequades per a la seva anàlisi posterior. L'objectiu no és únicament llegir els fitxers produïts per l'eina, sinó reorganitzar-ne el contingut en un model tabular que permeti comparar mètriques, filtrar operacions i representar patrons d'I/O de manera sistemàtica.

El flux metodològic parteix de tres registres principals: *Perf*, *Parser* i *DXT* [6]. Aquests fitxers comparteixen una capçalera comuna amb metadades de l'execució, però difereixen en el nivell de detall i en el tipus d'informació que proporcionen. Per aquest motiu, cada registre es processa de manera diferenciada, conservant únicament les variables necessàries per a l'anàlisi del rendiment d'I/O en aplicaci-

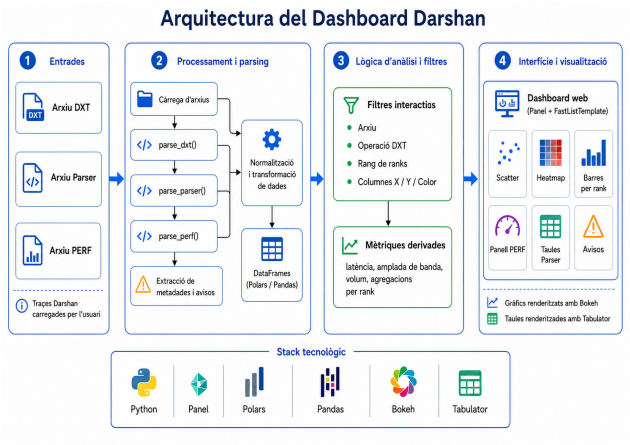

Fig. 1: Arquitectura de l'aplicació

ons de *Deep Learning* executades en entorns HPC.

El procés d'anàlisi es divideix en tres fases principals. La primera fase correspon a la càrrega dels arxius Darshan i a l'extracció de metadades comunes de l'execució. Aquesta informació permet contextualitzar cada traça mitjançant dades com l'identificador del treball, el nombre de processos, el temps total d'execució i les marques temporals d'inici i fi. La segona fase consisteix en el processament específic de cada tipus de registre. Pel fet que *Perf*, *Parser* i *DXT* descriuen el comportament I/O des de nivells de granularitat diferentes, es defineix un procediment de *parsing* independent per a cadascun. Aquesta separació permet conservar la semàntica pròpia de cada arxiu i evita barrejar mètriques globals, comptadors agregats i esdeveniments individuals en una mateixa etapa de processament. La tercera fase transforma les dades extretes en estructures tabulars. Aquestes estructures permeten aplicar operacions de filtrat, agregació i selecció de variables, servint com a base per a les visualitzacions descrites posteriorment. El flux complet d'aquesta metodologia, des dels arxius d'entrada fins a la generació de *DataFrames* i mètriques derivades.

3.1 Processament del registre Perf

El registre *Perf* s'utilitza per obtenir una visió global del rendiment d'entrada/sortida. Aquest fitxer resumeix el comportament per interfície d'I/O i permet identificar mètriques agregades associades principalment als mòduls POSIX i STDIO.

Durant el processament d'aquest registre s'extreuen les metadades generals de l'execució i les mètriques principals de rendiment. Entre aquestes mètriques s'inclouen el volum total de *bytes* transferit, els temps d'I/O associats a fitxers únics i compartits, el temps agregat calculat sobre el *procès* més lent i l'ampla de banda estimat.

El resultat d'aquest processament s'organitza com una estructura de dades jeràrquica, on les mètriques queden agrupades per mòdul. Aquesta organització permet que el dashboard mostri un resum compacte del comportament global de l'aplicació, sense barrejar aquesta informació amb els esdeveniments individuals del registre *DXT*.

3.2 Processament del registre Parser

El registre *Parser* conté comptadors i mètriques agregades per mòdul, fitxer i procès. En aquest treball es consideren principalment els mòduls POSIX i LUSTRE, ja que proporcionen informació rellevant sobre les operacions d'entrada/sortida i sobre la distribució física dels fitxers en el sistema d'emmagatzematge paral·lel.

El processament d'aquest fitxer es basa en la seva estructura tabular. Cada línia conté informació sobre el mòdul, el procès, l'identificador del registre, el nom del comptador, el valor del comptador, el nom del fitxer, el punt de muntatge i el tipus de sistema de fitxers. A partir d'aquests camps es construeix una taula en format llarg, on cada fila representa un comptador concret associat a un fitxer i a un mòdul determinat.

Durant aquesta etapa s'exclouen fitxers que no formen part de l'anàlisi principal de dades, com ara mòduls auxiliars o fitxers generats per l'entorn d'execució. Aquesta decisió redueix el soroll en l'anàlisi i permet centrar l'exploració en els fitxers rellevants per a l'aplicació de *Deep Learning*.

En el cas del mòdul LUSTRE, el registre *Parser* permet recuperar informació relacionada amb la configuració del sistema de fitxers, com ara la mida del *stripe* i els OST associats. Aquestes dades són especialment rellevants per analitzar com es distribueixen físicament els accessos sobre el sistema d'emmagatzematge.

3.3 Processament del registre DXT

El registre *DXT* proporciona el nivell de detall més alt, ja que conté esdeveniments individuals d'I/O amb informació temporal i espacial. A diferència de *Perf* i *Parser*, que ofereixen mètriques agregades, *DXT* permet reconstruir la seqüència d'operacions realitzades pels processos durant l'execució.

El processament d'aquest fitxer s'organitza per blocs de fitxer i *procès*. Cada bloc conté informació contextual, com ara l'identificador del fitxer, el nom del fitxer, el *procès*, el node de compút, el nombre de lectures i escriptures, el punt de muntatge, el tipus de sistema de fitxers, la configuració de LUSTRE i els OST implicats. A continuació, es processen les files corresponents a cada operació individual.

Per a cada esdeveniment s'extreuen variables com el tipus d'operació, el segment, l'*offset*, la mida transferida, l'instant d'inici, l'instant de finalització i l'OST associat. A partir d'aquests camps es calculen mètriques derivades, com la durada de l'operació, la latència en mil·lisegons, la mida transferida en KiB o MiB i l'ampla de banda.

Aquest registre permet analitzar patrons espacials, com els rangs d'*offset* accedits i les mides de transferència, així com patrons temporals, com la distribució de les operacions al llarg de l'execució o el solapament entre processos. Per aquest motiu, *DXT* constitueix la font principal per a les visualitzacions dinàmiques del dashboard.

3.4 Disseny del model de dades

El model de dades es dissenya a partir de tres estructures principals, una per a cada tipus de registre processat. La primera emmagatzema les mètriques globals procedents de

Perf. La segona conté els comptadors extrets del *Parser*. La tercera representa els esdeveniments individuals del *DXT*.

Aquesta separació respon a una diferència fonamental en la granularitat de les dades. *Perf* resumeix l'execució completa, *Parser* descriu comptadors per fitxer, mòdul i procès, mentre que *DXT* registra operacions individuals. Mantenir aquestes fonts separades evita duplicitats i permet aplicar a cadascuna el tipus d'anàlisi més adequat [11]..

El *DataFrame* derivat de *DXT* inclou variables orientades a l'anàlisi temporal i espacial de les operacions d'I/O. El *DataFrame* derivat del *Parser* s'orienta a l'anàlisi de comptadors POSIX i LUSTRE. L'estructura procedent de *Perf* s'utilitza com a font per als indicadors generals del rendiment de l'aplicació.

Sobre aquestes estructures s'apliquen filtres comuns, principalment per fitxer, operació i rang de processos. D'aquesta manera els objectes visuals del dashboard poden actualitzar-se de manera coherent a partir d'una mateixa selecció realitzada per l'usuari.

4 Implementació de l'eina de visualització de patrons en I/O

L'aplicació desenvolupada consisteix en un dashboard web interactiu per explorar les dades processades a partir dels registres Darshan. El seu disseny respon a la necessitat d'integrar en una mateixa interfície la càrrega de fitxers, la selecció de filtres, la visualització gràfica i la consulta tabular de mètriques.

L'arquitectura general del software es mostra a la Figura 1. En aquesta figura s'observa com els fitxers *DXT*, *Parser* i *Perf* són carregats per l'usuari, processats mitjançant funcions específiques de *parsing* i transformats en estructures tabulars. Aquestes estructures alimenten la lògica d'anàlisi, els filtres interactius i els components de visualització.

4.1 Arquitectura de la aplicació

L'arquitectura de la aplicació s'organitza en quatre capes principals. La primera capa correspon a les entrades del sistema, formades pels tres fitxers Darshan requerits. La segona capa realitza el processament i el *parsing* dels registres. La tercera capa conté la lògica d'anàlisi, on s'apliquen filtres i es calculen mètriques derivades. La quarta capa correspon a la interfície d'usuari i a les visualitzacions.

Aquesta divisió permet separar les responsabilitats de l'aplicació. El processament de dades queda desacoblat de la representació gràfica, mentre que els filtres actuen com a enllaç entre les dades processades i les vistes mostrades a l'usuari. Aquesta organització facilita el manteniment del codi i permet estendre l'app amb noves visualitzacions o mètriques sense modificar completament el flux de processament.

La Figura (1) resumeix aquesta arquitectura d'extrem a extrem.

L'eina està implementada en Python i utilitza biblioteques especialitzades per al processament de dades, el desenvolupament d'interfícies i la visualització interactiva.

Panel s'utilitza com a framework principal per construir l'aplicació web. Mitjançant Panel es defineixen els widgets de càrrega de fitxers, els selectors, els filtres, les pestanyes, els panells laterals i la plantilla general del dashboard [14].

Polars i Pandas s'utilizan per tabular de les dades i transformacions necesaries per les visualitzacions [15].

Bokeh s'empra per generar les visualitzacions interactives [13]. Els gràfics construïts amb aquesta biblioteca permeten incorporar eines de navegació, zoom, desplaçament, escales de color i *tooltips* amb informació detallada de cada operació.

4.2 Panel del dashboard

La interfície del dashboard s'organitza en tres zones principals, tal com es mostra a la Figura 2.

Al panell esquerre se situa la zona de càrrega de fitxers. Des d'aquesta secció l'usuari selecciona els fitxers *DXT*, *Parser* i *Perf*. També es mostra informació sobre l'estat de la càrrega, indicant si els fitxers han estat processats correctament o si s'ha produït algun error.

La zona central concentra els elements principals d'anàlisi. A la part superior es troben els filtres interactius. Aquests filtres permeten seleccionar els fitxers a analitzar, restringir el tipus d'operació *DXT* i escollir els eixos X i Y de les gràfiques principals.

Sota els filtres se situa la gràfica principal. Aquesta zona està dissenyada per mostrar la visualització més important de l'anàlisi *DXT*, i pot alternar entre un grafic de dispersió o un mapa de calor.

La zona central també inclou gràfiques secundàries orientades a l'anàlisi per procés. Aquestes visualitzacions permeten resumir mètriques com el volum total transferit i la latència mitjana. A la part inferior se situa una taula interactiva amb informació procedent del *Parser*, centrada en comptadors POSIX i LUSTRE.

El panell dret agrupa etiquetes i mètriques de rendiment. En aquesta zona es presenta informació procedent del registre *Perf* i mètriques associades al módul POSIX. La seva funció és mantenir visible un resum de l'execució mentre l'usuari explora els gràfics i taules de l'àrea central.

4.3 Visualitzacions implementades

El dashboard inclou visualitzacions orientades a diferents nivells d'anàlisi.

El graáfic de dispersió representa operacions individuals d'entrada/sortida. Cada punt correspon a un esdeveniment *DXT*, i l'usuari pot seleccionar quines variables es representen als eixos i quina s'utilitza com a color. Aquesta visualització és adequada per analitzar relacions entre temps, procès, *offset*, mida de l'operació, latència i amplada de banda.

El mapa de calor agrega una mètrica numèrica sobre dues dimensions seleccionades per l'usuari. Aquest tipus de visualització permet detectar concentracions d'activitat, zones amb major volum de transferència o regions temporals i espacials amb major intensitat d'operacions.

Les gràfiques secundàries resumeixen el comportament per procès. Una d'elles mostra el volum total d'entrada/sortida per procés, mentre que una altra representa la latència mitjana per procès. Aquestes visualitzacions permeten identificar desequilibris entre processos i diferències

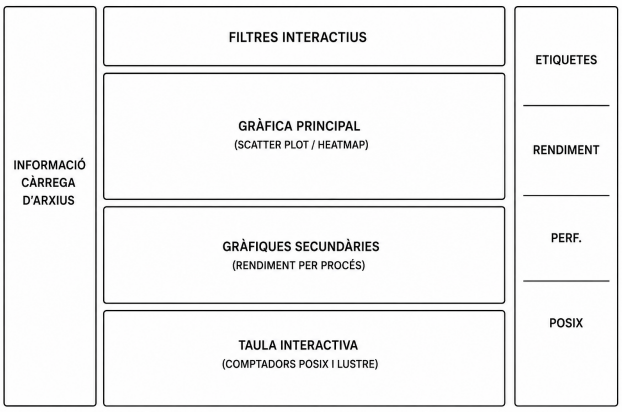

Fig. 2: Esquema del dashboard

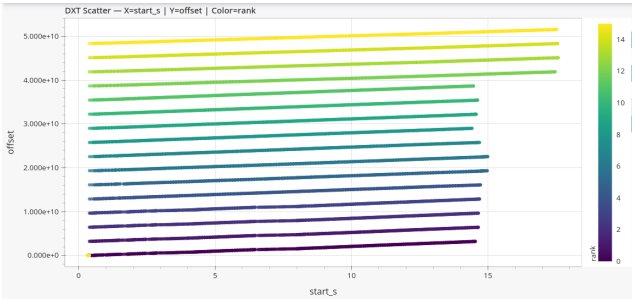

Fig. 3: Gràfic de dispersió: Patró temporal i espeacial de lectura per procès

en la càrrega d'I/O. S'afegeix en node de comput en la llegenda, per indentifcar si els comportampents dels processos son semblats en un mateix node.

Les taules interactives complementen les visualitzacions gràfiques. La taula detallada permet examinar comptadors POSIX i LUSTRE a nivell de fitxer i procès, mentre que la taula resum facilita una visió agregada per mòdul i comptador.

5 VALIDACIÓ DE L'EINA I ANÀLISI DE RESULTATS

S'ha avaluat l'eina amb diverses traces darshan generades en diferents contextos, per a verificar la funcionalitat en diferents casuístiques. L'aplicació analitzada es DLIO Benchmark, configurada per a simular la càrrega de E/S sobre un dataset d'un arxiu HDF5 en format únic mitjançant accés compartit i amb emmagatzematge en Lustre.

Realitzem una comparació amb 5 execucions de 3, 8, 16, 32 i 48 processos augmentant la distribució de les dades en Lustre en 1,2,4,8 i 12 OSTs respectivament. En la taula 1 obtenim una comparativa dels resultats:

Observem que tant per l'augment dels processos com per l'augment de les OST de Luste augmenta el rendiment de l'aplicació considerablement ja que amb 48 processos obtindrem un augment de 11.8x en el temps d'execució. Aquestes dades es poden identificar en l'apartat d'etiquetes de la figura (2)

5.1 Analisis temporals de patrons per accès

El gràfic en la figura 3 representa l'evolució temporal de les operacions de lectura registrades en la traça DXT, mostrant en l'eix X l'instant d'inici de cada operació, en l'eix I l'offset de l'arxiu accedit i mitjançant el color el procés que realitza l'operació.

El patró observat indica un accés paral·lel i majoritàriament ordenat a l'arxiu, ja que cada procés descriu una trajectòria ascendent gairebé lineal, cosa que significa que els processos llegeixen regions d'offset creixents a mesura que avança l'execució. La separació vertical entre les línies suggereix que cada procés treballa sobre una regió distinta de l'arxiu, seguint una distribució per blocs relativament equilibrada. Aquest comportament és coherent amb un accés a un arxiu compartit en el qual el dataset es reparteix entre diversos processos, de manera que cada procés llegeix una zona concreta de l'arxiu.

La superposició temporal de les línies mostra que les lectures es realitzen de manera concurrent, per la qual cosa no s'aprecia una serialització completa de la E/S entre processos. A més, la similitud dels pendents indica que els ranks avancen a velocitats comparables, encara que s'observen petites diferències en la durada d'algunes trajectòries, la qual cosa podria reflectir cert desbalance temporal o variacions puntuals en el rendiment de lectura. En la zona inicial del gràfic apareixen accessos a offsets baixos realitzats per diversos ranks, la qual cosa probablement correspon a lectures de capçaleres o metadades de l'arxiu HDF5 abans d'accedir a les regions principals del dataset.

5.2 Anàlisi temporal de l'ampla de banda

El mapa de calor en la figura (4) representa l'evolució temporal de l'amplada de banda de lectura per procès, utilitzant en l'eix X l'instant d'inici de les operacions agrupades en intervals temporals, en l'eix Y el procès que realitza l'operació i com a valor de color l'ampla de banda agregat en cada contenidor. Aquest mapa de calor permet analitzar la intensitat i variabilitat del rendiment de lectura al llarg de l'execució. És important considerar que els valors representats no corresponen necessàriament a operacions individuals, sinó a dades agregades dins de cada interval temporal, per la qual cosa cada cel·la resumeix el comportament d'un procès durant una finestra temporal concreta.

El patró observat confirma l'existència d'una fase de lectura paral·lela, ja que la majoria dels processos presenten activitat durant una finestra temporal comuna. Això indica que els processos realitzen operacions de E/S de manera concurrent i no de manera completament serialitzada. No obstant això, l'amplada de banda no és completament homogeni entre procès ni al llarg del temps.

S'observen variacions de color dins d'un mateix procès i entre distints procès, la qual cosa reflecteix canvis en el rendiment de lectura durant l'execució. Alguns intervals presenten valors d'ampla de banda més elevats, mentre que uns altres mostren un rendiment menor, la qual cosa pot estar associat a diferències en la mida de les operacions, variacions en la latència, contenció en el sistema d'arxius o efectes derivats de l'accés concurrent a l'arxiu compartit.

També s'aprecia certa disparitat temporal entre processos. Alguns procesos finalitzen la seva activitat abans, mentre que uns altres continuen realitzant lectures durant inter-

TAULA 1: TAULA COMPARATIVA RENDIMENT

Procs	OSTs	Runtime app	I/O slowest rank	total read	Bandwith	GiB/rank
4	1	74.38 s	51.97 s	48.0 GiB	946 MiB/s	12.0
8	2	30.84 s	11.89 s	48.0 GiB	4,135 MiB/s	6.0
16	4	17.88 s	8.11 s	48.0 GiB	6,057 MiB/s	3.0
32	8	8.72 s	4.09 s	48.0 GiB	12,023 MiB/s	1.5

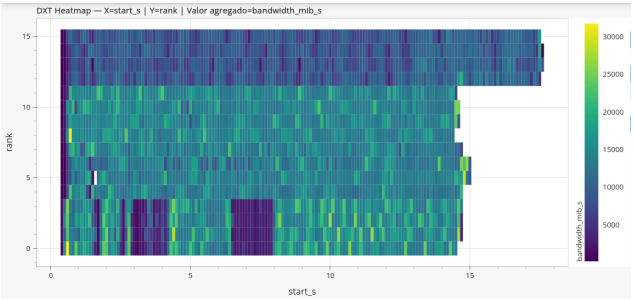

Fig. 4: Mapa de calor: Evolució temporal de l'ampla de banda

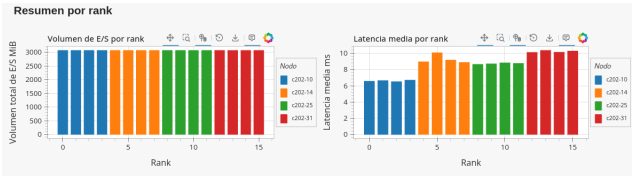

Fig. 5: Resumen del volumen i latència d'I/O per procès

vals posteriors. Aquest comportament suggereix que, encara que l'accés espacial observat en el gràfic de dispersió es regular i està distribuït per regions de l'arxiu, la durada efectiva de la fase de lectura no és idèntica per a tots els processos.

Les zones sense color han d'interpretar-se com a intervals en els quals no es van registrar operacions de lectura iniciades per a aquest procès, i no necessàriament com a amplada de banda nul·la. Aquesta distinció és rellevant a causa de l'ús de contenidors temporals per a agregar la informació.

5.3 Anàlisi entre processos

Les gràfiques en la figura (5) resum el comportament de la E/S a nivell de processos i permeten complementar l'anàlisi temporal i espacial realitzat prèviament en les figures (3 i 4. En el primer gràfic s'observa que el volum total de E/S llegit per cada procés és pràcticament constant, amb valors pròxims als 3000 MiB per a tots els processos. Això indica que la distribució del treball està equilibrada en termes de quantitat de dades transferides, ja que no s'aprecien diferències significatives entre processos. Aquesta observació és coherent amb el patró identificat en la Figura (3), on cada procès accedeix a una regió diferenet de l'arxiu, la qual cosa suggereix un repartiment regular del dataset entre els processos.

No obstant això, la segona gràfica mostra que aquest equilibri en volum no es tradueix en una latència homogènia. Encara que tots els processos transfereixen una quantitat de dades similar, la latència mitjana varia de manera apreciable entre processos.

Els processos agrupats en un mateix node mostren comportaments similars, per tant la diferència en la latencia pot derivar de factors locals del node, com el client Lustre, la ruta cap als OSTs, la càrrega del node o la interacció entre els processos d'aquest node i el sistema d'arxius.

Aquesta interpretació també és consistent amb la figura (4), on s'observaven variacions temporals de l'amplada de banda per proces.

En conjunt, aquests resultats indiquen que la partició espacial del dataset està ben equilibrada, però que el rendiment efectiu de E/S presenta un component dependent del node. Per tant, el possible coll d'ampolla no sembla estar associat a una mala distribució del volum de dades entre processos, sinó a diferències en la latència d'accés durant la lectura paral·lela.

6 DISCUSSIÓ

Els resultats obtinguts mostren que l'anàlisi visual de traces d'I/O permet interpretar amb més detall el comportament d'una aplicació de Deep Learning en un entorn HPC. En el cas analitzat, les traces de Darshan indiquen que els processos accedeixen a regions diferents de l'arxiu compartit i que les lectures es produeixen de manera concurrent. Aquest comportament suggereix una distribució espacial del dataset relativament equilibrada i sense una serialització completa de les operacions d'entrada/sortida. Per tant, el patró d'accés generat per l'aplicació és, en general, adequat per a una execució paral·lela.

La principal aportació pràctica de l'eina desenvolupada és que integra en una mateixa interfície diferents nivells d'informació: mètriques globals del registre Perf, comptadors agregats del Parser i esdeveniments detallats del DXT. Aquesta integració facilita passar d'una visió general del rendiment a una anàlisi més específica dels patrons temporals, espacials i per procés. Això pot ajudar usuaris i administradors de sistemes HPC a detectar desequilibris, identificar accessos irregulars, comparar configuracions i prendre decisions sobre el format de dades, el nombre de processos, la configuració de Lustre o l'estratègia de lectura.

Destacar que la visualització interactiva aporta valor perquè redueix la complexitat de les traces. Els fitxers generats per eines com Darshan poden contenir una gran quantitat d'informació difícil d'interpretar manualment. Mitjançant gràfics de dispersió, mapes de calor, filtres i taules resum, el dashboard permet explorar les dades de manera més ràpida i detectar patrons que podrien passar desapercebuts en informes agregats.

Una limitació d'aquesta eina es que permet observar correlacions i patrons, però no determina automàticament la causa exacta d'un problema de rendiment. Per exemple, una latència elevada pot estar associada a uun coll de bote-

lla en el sistema de fitxers, desequilibri entre nodes, configuració de striping o característiques internes de l'aplicació. Per confirmar aquestes causes seria necessari complementar l'anàlisi amb altres fonts d'informació, com mètriques del sistema, monitoratge de Lustre, informació del scheduler o perfiladors del framework de Deep Learning.

Finalment, el dashboard se centra en l'anàlisi posterior a l'execució i no en el monitoratge en temps real. Això és adequat per estudiar experiments ja finalitzats i comparar configuracions, però limita el seu ús en escenaris on es vulgui detectar un problema durant l'execució de l'aplicació. Malgrat aquestes limitacions, els resultats obtinguts mostren que l'eina és útil com a suport per comprendre el comportament d'I/O i com a punt de partida per optimitzar aplicacions de Deep Learning en entorns HPC.

7 Conclusions

Aquest treball ha abordat l'anàlisi i la visualització de patrons d'entrada/sortida en aplicacions de Deep Learning executades en entorns HPC. L'objectiu principal era transformar traces d'I/O generades amb Darshan en una representació més accessible i útil per a l'estudi del rendiment. Per assolir aquest objectiu, s'ha definit un model de dades basat en els registres Perf, Parser i DXT, s'ha desenvolupat una eina interactiva de visualització i s'ha validat el seu funcionament amb traces generades a partir del benchmark DLIO sobre un sistema de fitxers Lustre.

Els objectius plantejats s'han complert de manera satisfactòria. En primer lloc, s'ha dissenyat un procés de parsing capaç d'extreure metadades, comptadors agregats i esdeveniments individuals d'I/O. En segon lloc, aquestes dades s'han organitzat en estructures tabulars que permeten aplicar filtres, calcular mètriques derivades i alimentar diferents visualitzacions. Finalment, s'ha implementat un dashboard que integra gràfics de dispersió, mapes de calor, gràfiques per procés, taules de comptadors i indicadors globals de rendiment.

Els resultats obtinguts confirmen que l'enfocament visual facilita la interpretació de traces d'I/O complexes i complementa les mètriques agregades tradicionals. La representació conjunta de patrons temporals, espacials i per procés permet obtenir una visió més completa del comportament de l'aplicació i dona suport a la presa de decisions per optimitzar el rendiment en entorns HP

La principal aportació del treball és, proporcionar una eina que facilita la interpretació de traces complexes i dona suport a l'anàlisi del rendiment d'I/O. Aquesta aproximació pot ser útil tant per a desenvolupadors d'aplicacions de Deep Learning com per a usuaris i administradors de sistemes HPC, ja que permet comparar configuracions, detectar comportaments ineficients i orientar possibles decisions d'optimització relacionades amb el format de dades, el nombre de processos, la distribució del dataset o la configuració de Lustre.

Com a treball futur, una primera línia de millora seria ampliar la integració amb altres eines de profiling i tracing. A més dels registres de Darshan, es podrien incorporar mètriques procedents d'eines com seff, perf, strace, monitoratge de Lustre o perfiladors específics de frameworks de Deep Learning. Això permetria relacionar els patrons d'I/O amb altres dimensions del rendiment, com l'ús de CPU, memòria, GPU, eficiència del treball o comportament del sistema de fitxers. Una segona línia de treball seria crear una base de dades relacional per emmagatzemar de manera estructurada la informació analitzada. Aquesta base de dades podria incloure entitats com models de Deep Learning, datasets, execucions, configuracions del sistema, traces, fitxers, processos i mètriques. D'aquesta manera, seria possible conservar l'històric d'experiments, consultar execucions anteriors i facilitar estudis comparatius més sistemàtics. Finalment, una millora rellevant seria afegir a l'eina una funcionalitat per comparar diverses execucions dins d'una mateixa visualització. Aquesta extensió permetria analitzar de manera directa l'impacte de modificar el nombre de processos, el nombre d'OSTs, el format del dataset, la mida de les operacions o altres paràmetres de configuració. Amb aquesta funcionalitat, el dashboard passaria de ser una eina d'anàlisi individual de traces a una plataforma de comparació experimental per estudiar el rendiment d'I/O en aplicacions de Deep Learning sobre HPC.

Agraïments

Vull agrair el suport de la meva tutora, Betzabeth, per haverme ofert ajuda i orientació sempre que ho he necessitat, així com per la flexibilitat mostrada davant les dificultats de compaginar l'horari laboral amb l'horari acadèmic.

També vull agrair al professor Edixson Parraga el material addicional proporcionat i la seva disponibilitat per respondre els dubtes que m'han sorgit durant el desenvolupament del treball.

Finalment, vull donar les gràcies a la meva família, que sempre m'ha donat suport, especialment durant les etapes més difícils d'aquest treball.

Referències

[1] S. Lewis, J. Bez, and S. Byna, *I/O in Machine Learning Applications on HPC Systems: A 360-degree Survey*, arXiv preprint arXiv:2404.10386, 2024. Available: https://arxiv.org/abs/2404.10386

[2] T. Fukai, K. Sato, and T. Hirofuchi, *Analyzing I/O Performance of a Hierarchical HPC Storage System for Distributed Deep Learning*, arXiv preprint arXiv:2301.01494, 2023. Available: https://arxiv.org/abs/2301.01494

[3] S. W. D. Chien, S. Markidis, C. Peng, I. B. Peng, and E. Laure, *Characterizing Deep-Learning I/O Workloads in TensorFlow*, arXiv preprint arXiv:1810.03035, 2018. Available: https://arxiv.org/abs/1810.03035

[4] A. Khan et al., *HVAC: Removing I/O Bottleneck for Large-Scale Deep Learning Applications*, 2022. Available: https://www.osti.gov/servlets/purl/1902810

[5] H. Devarajan et al., *DLIO: A Data-Centric Benchmark for Scientific Deep Learning Applications*, 2021. Available: https://www.cs.iit.edu/~scs/assets/files/devarajan2021dlio.pdf

[6] Darshan, *Documentation*. Available: https://www.mcs.anl.gov/research/projects/darshan/documentation/

[7] M. Kerrisk, *strace(1) – Linux manual page*. Available: https://man7.org/linux/man-pages/man1/strace.1.html

[8] S. W. D. Chien, A. Podobas, I. B. Peng, and S. Markidis, *tf-Darshan: Understanding Fine-Grained I/O Performance in Machine Learning Workloads*, in *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 359–370, IEEE Computer Society, 2020. doi: https://doi.org/10.1109/CLUSTER49012.2020.00046

[9] Argonne Training Program on Extreme-Scale Computing, *Agenda 2024*, 2024. Available: https://extremecomputingtraining.anl.gov/agenda-2024/

[10] S. Mendez, *Analyzing the I/O Patterns of Deep Learning Applications*, Presentation slides, July 2, 2021. Available: https://hps.vi4io.org/_media/events/2021/iodc21-1030-mendez.pdf

[11] E. Parraga, B. Leon, S. Mendez, D. Rexachs, R. Suppi, and E. Luque, *An Empirical Method for Processing I/O Traces to Analyze the Performance of DL Applications*, in M. Naiouf, L. De Giusti, F. Chichizola, and L. Libutti, Eds., *Cloud Computing, Big Data and Emerging Topics: 12th Conference, JCC-BD and ET 2024, Revised Selected Papers*, pp. 74–90, Springer, 2024. doi: https://doi.org/10.1007/978-3-031-70807-7_6

[12] E. A. Parraga Pinzon, B. León, S. Mendez, D. I. Rexachs del Rosario, D. Franco Puntes, and E. Luque, *Deep Learning Data Handling: Exploring File Formats and Access Strategies*, *Cluster Computing*, vol. 28, Article 613, 2025. doi: https://doi.org/10.1007/s10586-025-05283-3

[13] Bokeh Development Team, *Bokeh Documentation*. Available: https://docs.bokeh.org/en/latest/

[14] HoloViz, *Panel Documentation*. Available: https://panel.holoviz.org/

[15] Polars, *Polars User Guide*. Available: https://docs.pola.rs/