DNTS

Distributed Neural Training Simulation

Final Report for the Distributed Systems Course 2024/25

Autori: Colletta L., Giacobbi D. F., Fantilli G.

Panoramica e Obiettivi del Progetto

- **Cos'è:** Un framework per simulare, analizzare e visualizzare il training di Reti Neurali Profonde (DNN) in un ambiente completamente decentralizzato.
- **Paradigma P2P:** Il sistema abbandona l'approccio tradizionale basato su *Parameter Server* (centralizzato) a favore di un'architettura **leaderless**.
- **Protocollo Core:** Implementazione del **Gossip Learning** asincrono: i nodi elaborano dati locali e scambiano i pesi con peer selezionati casualmente per raggiungere il consenso globale.
- **Tecnologie Chiave:** Sviluppato in **Scala 3**, utilizzando il Modello degli Attori (**Akka Cluster**) e i principi della programmazione funzionale pura.

Interfacce di Sistema: CLI & GUI

Il sistema si presenta come un'applicazione ibrida con due interfacce complementari:

- **Command-Line Interface (CLI):**
  - Meccanismo di controllo primario per il bootstrap dei nodi.
  - Configurazione dei ruoli (Seed o Client) e dei parametri di binding di rete (IP/Porte).
- **Graphical User Interface (GUI):**
  - Dashboard di visualizzazione in tempo reale (plotting dinamico).
  - Monitoraggio dell'evoluzione del modello: Decision Boundary, Loss Function e metriche di Consenso.
  - Pannello operativo per gestire il ciclo di vita (Start, Pause, Resume) e iniettare guasti (simulazione di crash).

Simulazione di Training Decentralizzato

- **Obiettivo Operativo**: N nodi indipendenti risolvono un compito di classificazione apprendendo da dataset 2D etichettati e partizionati localmente.
- **Meccanismo di Convergenza**: I nodi scambiano e mediano i pesi locali per convergere verso un modello globale ottimizzato.
- **Target User**: Ricercatori, studenti di sistemi distribuiti e data scientist.
- **Resilienza Integrata**: Meccanismi di fault-tolerance (Passive Gossiping e snapshotting dello stato locale) per il recupero dai fallimenti.
- **Gestione Dati**: I dataset sintetici e lo stato delle reti sono salvati localmente su file system per garantire persistenza senza database centralizzati.

Perché è necessaria la distribuzione?

Eliminazione SPOF

Nessun Single Point of Failure. La rete P2P leaderless garantisce alta disponibilità; se un nodo crasha, il training prosegue tra i peer sopravvissuti.

Data Locality e Privacy

Modellazione dei vincoli del mondo reale (es. edge computing). I dataset non vengono centralizzati, preservando la privacy; il consenso si ottiene scambiando solo i pesi.

Parallelismo

I compiti di Deep Learning sono intensivi. Distribuendo i dati, si parallelizza il pesante carico computazionale della Backpropagation sull'intero cluster.

Requisiti di Sistema (1)

Requisiti Utente

- **Inizializzazione e Configurazione:** L'utente deve poter avviare il cluster specificando i parametri di rete (IP, porte, nodo seed) e definire la topologia della rete neurale (livelli, neuroni, iperparametri).
- **Gestione dei Dati:** Capacità di selezionare dataset sintetici 2D che il sistema partiziona automaticamente tra i nodi.
- **Controllo Operativo (GUI):** Interfaccia interattiva per gestire il ciclo di vita della simulazione (Start, Pause, ...).
- **Fault Injection:** Possibilità di iniettare guasti dinamici di un nodo per testare la resilienza del sistema.
- **Monitoraggio:** Visualizzazione in tempo reale del Decision Boundary, della Loss Function e della varianza del consenso tra i nodi.

Requisiti Funzionali

- **Membership P2P:** Gestione autonoma della scoperta dei nodi e del rilevamento dei fallimenti senza un registro centrale.
- **Gossip Learning Asincrono:** Alternanza tra cicli di computazione locale (Backpropagation) e cicli di sincronizzazione asincrona con peer casuali.
- **Metrica di Consenso:** Capacità di aggregare una metrica globale per misurare la divergenza tra i modelli locali e la convergenza del cluster.
- **Core Matematico Puro:** Implementazione della Backpropagation tramite funzioni pure e strutture dati immutabili per garantire la sicurezza nella concorrenza.
- **Reintegro e Recovery:** Utilizzo di snapshot locali e strategie di "Passive Gossiping" per reintegrare nodi guariti senza inquinare il consenso globale.

Requisiti di Sistema (2)

Requisiti Non-Funzionali

- **Resilienza:** Alta disponibilità del sistema; il fallimento di uno o più nodi non deve interrompere il training globale.
- **Trasparenza della Distribuzione:** La complessità della rete (routing, serializzazione) deve essere nascosta alla logica matematica di dominio.
- **Efficienza di Rete:** Ottimizzazione del trasferimento di grandi matrici di pesi tramite serializzazione binaria a basso overhead.

Requisiti Implementativi

- Adozione rigorosa della **Programmazione Funzionale Pura** per garantire immutabilità e sicurezza dei tipi nel core matematico.
- Utilizzo di **Akka Typed** per la messaggistica asincrona e **Akka Cluster** per la gestione della membership P2P.
- Architettura a **strati** per isolare il dominio (stateless), il sistema ad attori (stateful) e l'interfaccia (boundary).

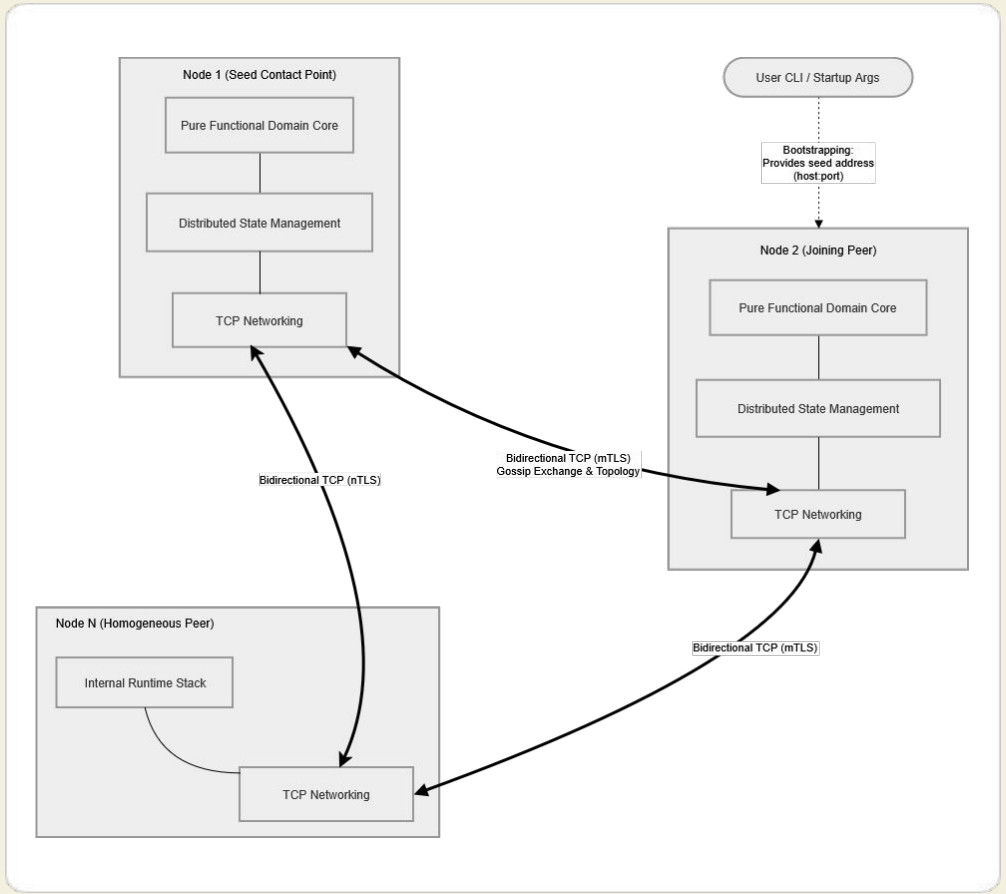
Proprietà Chiave del Sistema DNTS

Il progetto prioritizza i seguenti aspetti del design distribuito:

- **Consistenza Eventuale:** Si abbandona la consistenza forte per favorire la disponibilità (modello AP del Teorema CAP); i nodi convergono lentamente tramite media deterministica dei pesi (Gossip Averaging).
- **Scalabilità Orizzontale:** Il carico computazionale è perfettamente bilanciato tra i nodi (ciascuno elabora 1/N del dataset globale) e il traffico di rete è distribuito uniformemente dai protocolli epidemici.
- **Sicurezza:** Protezione del trasporto dati tramite crittografia nativa mTLS e controllo rigoroso degli accessi al cluster tramite autenticazione a step singolo e token JWT.
- **Fault Tolerance e Recoverability:** Il training globale sopravvive ai crash inaspettati dei nodi. I nodi superstiti continuano il ciclo di gossip in modo trasparente, mentre quelli riconnessi recuperano il proprio stato tramite snapshot locali su file system e si reintegrano senza inquinare il consenso.
- **Trasparenza della Distribuzione:** I dettagli di rete di basso livello (routing, physical location) sono totalmente astratti dal core matematico. Tuttavia, le dinamiche distribuite (ritardi, cambi di topologia, andamento della convergenza) rimangono volutamente visibili all'utente tramite la GUI per scopi di osservabilità.

Infrastruttura e Topologia di Rete

- **Nodi Omogenei:** Ogni nodo DNTS è un'unità autonoma che include l'intero stack (Dominio, Attori, Networking) senza dipendenze da server centrali.
- **Connettività:** Comunicazioni basate su connessioni TCP bidirezionali sicure tramite mutual TLS (mTLS).
- **Formazione del Cluster:** Discovery basata su nodi Seed (punti di contatto iniziali); una volta avviato il training, la topologia viene temporaneamente "congelata" per garantire la coerenza del dataset distribuito.
- **Flessibilità di Deployment:** Il design permette di eseguire i nodi sia localmente sulla stessa macchina (per test e simulazione visiva) sia distribuiti fisicamente su una rete LAN o macchine virtuali.
- **Formazione di una Rete Overlay:** Una volta stabilita la connessione iniziale con il nodo Seed, la topologia del cluster evolve come una rete overlay dinamica, dove la vista dei peer è mantenuta autonomamente tramite gossip.
- **Identificazione dei Nodi:** Ogni nodo è identificato in modo univoco sulla rete dalla combinazione host:port, evitando la necessità di registri di nomi o DNS centralizzati.



Architettura a Strati del Sistema

Separazione rigorosa tra logica matematica pura, gestione distribuita dello stato e interfacce utente.

1. Pure Functional Core

Incapsula l'intero dominio algoritmico e matematico (es. matrici, Gradient Descent, Backpropagation). Questo livello è completamente stateless, privo di side-effect e indipendente dalla rete, garantendo determinismo e massima testabilità.

2. Stateful Distributed Layer

Motore concorrente implementato in Akka Typed. Gestisce la concorrenza, la comunicazione P2P e l'incapsulamento dello stato mutabile. Isola le complessità della rete e garantisce aggiornamenti atomici tramite gli Attori.

3. Boundary Layer

Gestisce i confini del sistema verso l'utente. Include la CLI per il bootstrap della rete e la GUI (Java Swing) per il monitoraggio. Decoupla il rendering visivo dal loop di addestramento distribuito per evitare colli di bottiglia.

Livello Stateful e Message Passing

Gli attori incapsulano lo stato mutabile e comunicano esclusivamente tramite messaggi asincroni (ADTs).

Attori Chiave (Akka Typed)

- ModelActor: È il *Single Source of Truth* del nodo. Detiene i pesi della rete e aggiorna il proprio stato (Maturity) in modo strettamente atomico e sequenziale.
- TrainerActor: Motore di esecuzione locale (FSM). Delega i batch di dati al Core Puro per il calcolo dei gradienti e gestisce gli stati Idle/Training/Paused.
- GossipActor: Gestisce il protocollo P2P epidemico. Seleziona peer casuali e invia i modelli in background senza bloccare la computazione.

Interazione Asincrona

- Commands (Direttive): Messaggi imperativi per innescare azioni mirate. Esempio: UpdateModel inviato dal Trainer al ModelActor per aggiornare i pesi locali.
- Network Messages (Gossip): Payload binari scambiati via TCP. Esempio: GossipModelPush che trasferisce il modello remoto e innesca l'operazione di MergeWeights.
- Domain Events: Reazioni reattive ai cambiamenti del sistema. Esempio: TopologyChanged emesso dal Cluster per aggiornare le tabelle di routing in caso di crash.

Interaction Patterns: Ciclo di Vita e Training

- **Publish-Subscribe (Topology & Discovery):**
  - I nodi non usano registri centrali; all'avvio pubblicano i propri riferimenti (Service Keys) tramite l'**Akka Receptionist**.
  - Il DiscoveryActor si sottoscrive a questi topic per aggiornare in modo asincrono la tabella di routing locale ogni volta che un peer entra o esce dal sistema.
- **Point-to-Point Scatter (Initialization):**
  - Fase asimmetrica transitoria: il nodo Seed genera il dataset sintetico e lo diffonde inviando partizioni dirette (1/N) ai Client.
  - Una volta completata la distribuzione, l'interazione gerarchica viene smantellata a favore di una topologia puramente P2P.
- **Push-based Gossip (Training):**
  - Protocollo epidemico asincrono: il GossipActor seleziona un peer casuale e inoltra il modello aggiornato (GossipModelPush).
  - **Maturity come Logical Clock:** Il conflitto tra pesi remoti e locali è risolto tramite una media pesata sulla "maturità" del modello, che agisce da orologio logico scalare per ordinare causalmente gli aggiornamenti.
- **Periodic Request-Reply (Polling):**
  - Il MonitorActor interroga ciclicamente il ModelActor per snapshot delle metriche. Questo disaccoppia il loop computazionale ad alta velocità dal thread "lento" di rendering della GUI.

Interaction Patterns: Resilienza, Sicurezza e Dati

- **Heartbeating (Failure Detection):**
  - Monitoraggio continuo della liveness tramite Akka Cluster; il silenzio di un peer oltre una soglia prefissata innesca l'aggiornamento dinamico delle tabelle di routing.
- **Request-Reply Proxying (Authentication):** L' AuthActor sul Seed valida le credenziali e rilascia token **JWT**. Senza questo passaggio, i nodi non possono accedere al dataset o partecipare all'overlay.
- **Custom Marshalling (Presentation):** Ottimizzazione della banda per lo scambio di matrici tramite **serializzazione binaria ad-hoc** (Scala Type Classes), evitando l'overhead di formati testuali come JSON.

Reattività, FSM e Stato Immutabile

- **Message-Driven e Purezza Funzionale:** Sistema interamente reattivo e privo di memoria condivisa inter-nodo. I compiti computazionali pesanti (backpropagation, averaging) sono isolati in funzioni pure e deterministiche nel Domain Layer, garantendo una thread-safety nativa.
- **ModelActor (Single Source of Truth):** Unica autorità locale responsabile dello stato della Rete Neurale (pesi, bias, maturity). Sfrutta gli *Akka Typed Behaviors:* invece di mutare variabili interne l'attore evolve restituendo un nuovo comportamento che incapsula lo stato aggiornato, garantendo mutazioni atomiche e sequenziali.
- **Attori come FSM**: Modellazione del ciclo di vita di un attore, come nel caso del TrainerActor, tramite una Macchina a Stati Finiti dinamica:
  - Idle: In attesa della configurazione iniziale dei parametri.
  - Training: Loop di ottimizzazione attivo sui batch di dati locali.
  - Paused: Sospensione immediata del calcolo con *stashing* (accantonamento) dei messaggi di rete residui per evitare perdite di stato.
- **Gossip Timer-Driven:** Il comportamento del GossipActor è guidato da eventi temporizzati asincroni. A ogni impulso del timer, l'attore seleziona in autonomia un peer casuale dalla tabella di routing ed esegue il push del modello, regolando la diffusione epidemica senza bloccare gli altri attori.

Fault Tolerance e Disponibilità

- **Failure Detection e Adattamento:** Il sistema sfrutta l'heart-beating continuo di Akka Cluster per monitorare lo stato dei nodi. Se un peer non risponde entro le soglie previste, viene dichiarato irraggiungibile e il DiscoveryActor lo rimuove in modo asincrono dalle tabelle di routing.
- **Isolamento dei Crash:** I guasti hardware o di rete non bloccano la simulazione; i nodi superstiti ignorano i peer caduti e continuano i cicli locali di Gradient Descent e Gossip.
- **Snapshotting e Fast Recovery:** Lo stato del sistema (NNModel e partizione dati) viene salvato periodicamente in formato binario sul file system locale. In caso di riavvio, il nodo deserializza lo snapshot bypassando la fase di cold-start, per riprendere dall'ultima epoca salvata.
- **Passive Gossiping:** Per reintegrare i nodi riavviati senza corrompere il consenso globale, la Maturity (orologio logico) impone un merge fortemente pesato che penalizza i modelli obsoleti.
- **Load Sharing Intrinseco:** Eliminando i load balancer centrali, il sistema divide naturalmente il carico: ogni macchina processa solo 1/N del dataset e la banda viene ottimizzata dall'approccio randomico del protocollo epidemico.

Sicurezza, Crittografia e Controllo Accessi

- **Gatekeeping sul Nodo Seed:** L'ingresso nell'overlay P2P è subordinato all'autenticazione delle credenziali (protette da algoritmi di hashing unidirezionale nel modulo Crypto), gestita centralmente dall'AuthActor situato sul nodo Seed.
- **Token Stateless (JWT):** A login completato, l'AuthActor utilizza un JwtCodec custom per rilasciare un JSON Web Token firmato crittograficamente. I client usano questo token per validare le azioni successive senza l'uso di database di sessione.
- **Demarcazione dei Privilegi**: Il sistema impone confini operativi stretti tra i Seed (Amministratori con privilegi di setup, avvio e arresto globale) e i Client (Worker confinati esclusivamente al fetch del dataset, calcolo e gossip).
- **Crittografia in Transito (mTLS):** Tutte le comunicazioni TCP inter-nodo operano su Akka Artery Remoting e sono nativamente crittografate tramite mutual TLS (mTLS).

Impl.: Rete, Serializzazione e Storage

- **Trasporto TCP via Akka Artery:** Per supportare il protocollo Gossip senza perdita di dati, il sistema sfrutta connessioni TCP incapsulate in Akka Artery Remoting, garantendo la consegna affidabile e ordinata necessaria per le matrici di pesi.
- **Custom Binary Serialization:** I formati testuali (come JSON) sono stati scartati per evitare la saturazione della banda. I dati ad alta dimensionalità vengono trasformati in array di byte minimi e ultra-ottimizzati tramite Type Classes native di Scala 3.
- **Assenza di Database:** Nessun motore SQL o NoSQL per evitare colli di bottiglia e dipendenze esterne.
- **FileNodeRepository:** La persistenza di dataset, snapshot dei modelli e credenziali è gestita in modo custom tramite deserializzazione sequenziale diretta di file binari sul file system locale, massimizzando le performance di I/O.

Impl.: Tech Stack e Sicurezza Applicativa

- **Scala 3 (Pure Functional Core):** Modellazione della matematica tramite strutture immutabili per eliminare i side-effect durante il Gradient Descent. Protocolli di rete e di dominio protetti tramite Algebraic Data Types (ADT) e pattern matching.
- **Ecosistema Akka:**
  - **Akka Typed:** Garantisce l'incapsulamento sicuro dello stato senza lock e race conditions.
  - **Akka Cluster:** Fornisce l' infrastruttura P2P leaderless e la failure detection.
- **Autorizzazione Stateless:** L'autenticazione dei componenti evita di interrogare un database locale. Al contrario, AuthActor utilizza un JwtCodec personalizzato per generare, firmare crittograficamente e verificare i token Web JSON (JWT) in modo nativo all'interno del sistema.
- **Disaccoppiamento della GUI (Java Swing):** Il rendering visivo è isolato dal sistema ad attori. Il MonitorActor invia le metriche direttamente all'Event Dispatch Thread (EDT) di Swing per non rallentare l'addestramento distribuito.

Validazione: Testing Automatico e Isolamento

- **Strategia di Isolamento:** La netta separazione architetturale permette di testare la logica matematica pura in modo completamente indipendente dalla gestione dello stato concorrente.
- **Pure Functional Core (ScalaTest):** I componenti del livello di dominio (ad esempio, primitive di algebra lineare, algoritmi di retropropagazione) sono completamente senza stato e puri. Vengono sottoposti a test unitari esaustivi per garantire la correttezza matematica dei gradienti e della media dei modelli prima di essere distribuiti nell'ambiente distribuito.
- **Actor System (Akka TestKit):** Le interazioni asincrone e le macchine a stati vengono validate simulando peer e interfacce remote tramite l'uso di oggetti TestProbe, evitando di instanziare complessi cluster TCP reali per ogni test.
- **Automazione e CI:** L'intera suite di test è integrata nel ciclo di vita SBT (sbt test) e pronta per essere inserita in pipeline di Continuous Integration (es. GitHub Actions) per prevenire regressioni.

Validazione: Fault Injection e Test Coverage

- **Acceptance Testing Manuale:** Utilizzato per gli scenari intrinsecamente difficili da testare in modo deterministico (come il rendering visivo e i fallimenti fisici dell'infrastruttura).
- **Visual Convergence (GUI):** Validazione della fluidità della dashboard Java Swing, assicurandosi che il plotting dinamico (Decision Boundary, Loss) si aggiorni in tempo reale senza bloccare l'esecuzione distribuita.
- **Fault Injection Testing:** Test di resilienza su hardware reale (nodi in LAN) simulando crash improvvisi (kill del processo via OS) e partizioni di rete (disconnessione Wi-Fi), verificando l'aggiornamento dinamico delle routing table e la sopravvivenza del training.
- **Degree of Coverage (sbt-scoverage):** Elevata copertura nei moduli critici (core matematico, logica gossip e cluster); fisiologicamente minore solo sui livelli di confine (GUI) e negli scenari di errore di rete a bassissimo livello.

Grazie per l'attenzione

Fine della presentazione

Corso: Sistemi Distribuiti 2025/26

Team DNTS: Lorenzo Colletta, Domenico Francesco Giacobbi, Giorgio Fantilli