**Appunti Bruscolini**

**LEZIONE 1**

**🔹 Cos'è la Macchina di Turing?**

La **macchina di Turing (MdT)** è un **modello teorico di calcolatore** ideato da Alan Turing nel 1936. È una rappresentazione **astratta** che serve a formalizzare il concetto di **algoritmo** e di **calcolabilità**.

**🔹 Cosa dimostra la macchina di Turing?**

1. **Universalità**: ogni algoritmo eseguibile su un qualsiasi computer può essere simulato da una MdT. È un modello **equivalente** a qualsiasi sistema di calcolo moderno.
2. **Non decidibilità**: esistono **problemi non risolvibili** da alcun algoritmo. Il più famoso esempio è il **problema dell'arresto**:
   * "Data una descrizione di un algoritmo e un input, esiste un metodo per decidere se l’algoritmo si fermerà oppure no?" → **Turing ha dimostrato che no, non è possibile.**

**🔷 Che cos’è il teorema di Turing?**

Il **teorema di Turing** (1936-1937) stabilisce che **esistono problemi che non possono essere risolti da nessun algoritmo**, cioè **non decidibili**. Questo significa che **non esiste alcun procedimento automatico (algoritmo o macchina) che possa risolvere tutte le istanze di questi problemi in un tempo finito**.

Turing lo dimostrò introducendo un **modello astratto di calcolatore**: la **Macchina di Turing**.

**🔹 Cosa vuol dire *problema non decidibile*?**

Un problema è **non decidibile** quando:

* non è possibile costruire un algoritmo che restituisca una risposta **sì/no** (vero/falso) **per ogni input** del problema;
* o l’algoritmo dovrebbe andare avanti **all’infinito** su alcuni input.

**🔹 Implicazioni pratiche**

Anche se le MdT non esistono fisicamente, esse:

* Forniscono **le fondamenta teoriche dell’informatica**.
* Permettono di definire in modo rigoroso i concetti di **algoritmo**, **calcolabilità** e **complessità computazionale**.
* Sono alla base dei linguaggi di programmazione e dei compilatori.

Vediamo ora in dettaglio **cosa compone un’architettura AI&HPC (High Performance Computing)** di ultima generazione, suddividendo i vari blocchi principali. Questo tipo di infrastruttura è progettato per offrire **prestazioni elevate**, **scalabilità** e **flessibilità** per calcoli scientifici complessi, intelligenza artificiale, simulazioni, ecc.

**🔧 1. Elaborazione (Computer)**

La potenza di calcolo è fornita da un **insieme di nodi**, cioè computer collegati tra loro:

* **Nodi single o dual CPUs**:  
  → Ogni nodo contiene 1 o 2 processori. Questi sono adatti a elaborazioni generali (simulazioni, modelli matematici, ecc.).
* **Nodi oltre dual CPUs**:  
  → Architetture più avanzate con più di 2 CPU per nodo, usate per lavori molto intensivi (come simulazioni fisiche complesse).
* **Nodi con acceleratori GPU**:  
  → Integrano **schede grafiche (GPU)**, molto più efficienti per carichi di lavoro paralleli, come:
  + addestramento di reti neurali (AI),
  + simulazioni molecolari o fluidodinamiche (CFD),
  + elaborazione immagini e video.

**🌐 2. Comunicazione**

I nodi devono **scambiarsi dati velocemente e con bassa latenza**. Quindi:

* **Rete ad alta efficienza e bassa latenza**:  
  → Tecnologie come **InfiniBand** o **RoCE (RDMA over Ethernet)** permettono comunicazioni ultraveloci tra nodi.
* **Topologie di rete scalabili**:  
  → La rete può avere diverse forme (torus, fat tree, dragonfly…) per garantire **scalabilità orizzontale**: più nodi, più potenza, senza colli di bottiglia.
* **Rete di gestione**:  
  → Una seconda rete, basata su **Ethernet**, è dedicata a operazioni di servizio (monitoraggio, accensione/spegnimento nodi, aggiornamenti software, ecc.).  
  Non è usata per il calcolo; quindi, non ha bisogno di altissime prestazioni.

**💾 3. Sottosistema Storage (Archiviazione)**

Serve per leggere/salvare grandi quantità di dati, spesso con accesso parallelo da più nodi:

* **Varie tipologie di storage**:  
  → **Flash (SSD)**: molto veloce  
  → **HDD**: capacità elevata, più economico  
  → **Tape**: usato per archiviazione a lungo termine (backup, dati “freddi”)
* **Storage multi-livello (Multi-tier)**:  
  → I dati sono gestiti in diversi livelli (tier) per ottimizzare prestazioni e costi:
  + Tier 0: dati “caldi” su SSD
  + Tier 1: dati meno usati su HDD
  + Tier 2: dati archiviati su tape

**🛠️ 4. Sistemi di servizio**

Non fanno calcolo diretto, ma sono **essenziali per il funzionamento e la gestione del sistema**:

* **Nodi di gestione**:  
  → Supervisionano l’infrastruttura, monitorano lo stato dei nodi, raccolgono log, diagnosticano problemi hardware/software.
* **Nodi per gestione carico di lavoro** (workload management):  
  → Eseguono funzioni come:
  + assegnazione dei job (scheduler, es. Slurm),
  + gestione code e priorità,
  + bilanciamento del carico di lavoro tra i nodi.

**🇮🇹 Leonardo @ CINECA – Supercomputer italiano di classe mondiale**

**🏢 Dove si trova?**

* Presso il **CINECA**, il consorzio interuniversitario italiano di supercalcolo.
* Installato a **Bologna**, fa parte dell’iniziativa **EuroHPC** (European High Performance Computing Joint Undertaking).

Immagine che contiene testo, schermata, Carattere, numero

Il contenuto generato dall'IA potrebbe non essere corretto.

**FLOP** = **Floating Point Operation**  
Significa: una **singola operazione matematica** su numeri decimali (a virgola mobile)

Quando parliamo di potenza di calcolo, misuriamo **quante operazioni** il sistema riesce a fare **in un secondo**.

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

**Misure prestazionali di riferimento per un sistema HPC: HPL, HPCG, IO**

**🧪 1. HPL – High Performance Linpack**

**✅ Cosa misura?**

* La capacità del sistema di **risolvere una grande matrice piena** (densa) di equazioni lineari.

**📌 Caratteristiche:**

* **Carico computazionale molto alto**, poca comunicazione tra nodi.
* Sfrutta al massimo le **unità di calcolo (CPU/GPU)**.

**📊 Efficienza tipica:**

* **~70%** del picco teorico → RMAX/RPEAK ≈ 0,7

**🧠 Applicazioni rappresentate:**

* Simulazioni numeriche intensive (CFD – fluidodinamica computazionale),
* FFT (Fast Fourier Transform),
* Simulazioni geofisiche e in scienza dei materiali.

**🌐 2. HPCG – High Performance Conjugate Gradient**

**✅ Cosa misura?**

* La capacità di **risolvere una matrice sparsa** con molti zeri → più simile a calcoli scientifici reali.

**📌 Caratteristiche:**

* **Molto più realistica** di HPL.
* Richiede **molta comunicazione tra nodi** → valuta la rete e la memoria più della sola CPU.

**📊 Efficienza tipica:**

* **~5%** del picco teorico → RMAX/RPEAK ≈ 0,05

**🧠 Applicazioni rappresentate:**

* Simulazioni meteo e climatiche,
* Simulazioni CFD in geometrie complesse,
* Elementi finiti in meccanica strutturale,
* Sistemi **Multiphysics** (simulazioni che combinano più fenomeni fisici).

**💾 3. IO500 – Input/Output benchmark**

**✅ Cosa misura?**

* Le **prestazioni del sottosistema di I/O**: lettura/scrittura di **dati non strutturati**, anche in modalità random.

**📌 Caratteristiche:**

* Enfatizza l’**efficienza dello storage** (SSD, HDD, file system),
* E valuta la **capacità di comunicazione I/O tra nodi**.

**🧠 Applicazioni rappresentate:**

* Bioinformatica,
* Analisi di knowledge graphs (grafi semantici),
* Carichi AI/ML che accedono a tanti file di piccole dimensioni.

Immagine che contiene testo, schermata, Carattere, numero

Il contenuto generato dall'IA potrebbe non essere corretto.

**🧠 Tipologie di architettura parallela**

**🔹 SIMD – *Single Instruction, Multiple Data***

* **Una singola istruzione** viene applicata **a più dati contemporaneamente**.
* È perfetta per **operazioni ripetitive su grandi quantità di dati**.
* Tipico delle **GPU** e dei **processori vettoriali**.

**Esempio:**  
Sommare due vettori da 100 elementi → una sola istruzione che somma in parallelo tutti gli elementi.

**🔹 MIMD – *Multiple Instruction, Multiple Data***

* **Ogni core** può eseguire **istruzioni differenti su dati differenti**.
* È il modello tipico delle **CPU multicore**.
* Più flessibile, adatto a compiti **non omogenei** e **multi-threading**.

**Esempio:**  
Un core elabora un'immagine, un altro analizza un testo, un altro ancora calcola una simulazione.

**🧩 Componenti fondamentali della CPU**

**1. ALU – Arithmetic Logic Unit**

* Esegue operazioni matematiche (somma, moltiplicazione) e logiche (AND, OR, confronto).
* È il "cuore operativo" del processore.

**2. Registers (Registri)**

* Memorie molto piccole e veloci all’interno della CPU.
* Conservano dati e istruzioni temporanee.
* Comprendono:
  + **Program Counter (PC)**: indica la prossima istruzione da eseguire.
  + **Instruction Register (IR)**: contiene l’istruzione attualmente in esecuzione.

**3. Cache Memory**

* Memoria molto veloce **più vicina della RAM**.
* Conserva dati/istruzioni usati di frequente per **accelerare l’accesso**.

**4. System Clock**

* Il "metronomo" del sistema: regola il ritmo con cui la CPU esegue le istruzioni.
* Più è veloce (clock rate), più istruzioni al secondo → maggiore performance.

**5. Bus**

* Canali fisici (fili, piste) usati per **trasmettere dati e istruzioni** tra le varie parti della CPU e verso l’esterno (RAM, dispositivi, ecc.).

**🔢 Cos'è SAXPY/DAXPY/CAXPY/ZAXPY?**

Sono **operazioni vettoriali elementari** usate nei benchmark, ovvero test di prestazioni, per valutare l'efficienza di CPU e GPU.

* Il nome deriva da:
  + **SAXPY**: *Single-precision A·X Plus Y*
  + **DAXPY**: *Double-precision A·X Plus Y*
  + **CAXPY**: *Complex A·X Plus Y* (numeri complessi)
  + **ZAXPY**: *Double-precision Complex A·X Plus Y*

Immagine che contiene testo, schermata, Carattere, linea

Il contenuto generato dall'IA potrebbe non essere corretto.

**⚙️ Perché è utile per misurare le prestazioni?**

Perché coinvolge **tutte le principali operazioni di una CPU/GPU**:

1. **Load**: preleva i dati da RAM alla CPU (x[i] e y[i])
2. **Multiply and Add**: fa il calcolo a \* x[i] + y[i] usando l’**ALU vettoriale**
3. **Store**: salva il risultato di nuovo in memoria (y[i])

💡 È un'operazione semplice, ma contiene:

* **carico della memoria** (load/store),
* **calcolo floating point** (moltiplicazione + somma),
* **utilizzo della pipeline vettoriale** (SIMD),
* E si può **ripetere milioni di volte**: perfetta per testare il throughput massimo.

**🔄 Overlap (Sovrapposizione delle operazioni)**

I moderni processori sono **pipelined** e **paralleli**, quindi possono eseguire più operazioni **simultaneamente**:

* Mentre caricano un nuovo dato (Load), possono già sommare e salvare il precedente (Mult + Add + Store).
* Questo aumenta il throughput, ovvero quante operazioni riesce a fare al secondo.

Immagine che contiene testo, schermata, Carattere, numero

Il contenuto generato dall'IA potrebbe non essere corretto.

Immagine che contiene testo, schermata, Carattere, numero

Il contenuto generato dall'IA potrebbe non essere corretto.

⚙️ **Metriche tipiche di un sistema HPC moderno**

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

Immagine che contiene testo, schermata, Carattere, algebra

Il contenuto generato dall'IA potrebbe non essere corretto.

Immagine che contiene testo, schermata, Carattere, linea

Il contenuto generato dall'IA potrebbe non essere corretto.

Immagine che contiene testo, schermata, Carattere, ricevuta

Il contenuto generato dall'IA potrebbe non essere corretto.

**🔧 Cos’è il calcolo parallelo?**

Il calcolo parallelo consiste nel **dividere un lavoro in più parti** ed eseguirle **contemporaneamente** su più processori (CPU o core), per **risparmiare tempo**. È come dividere una grande torta tra più persone: tutti mangiano più in fretta.

**⚠️ Ma ci sono dei limiti**

Non tutto può essere diviso. In ogni programma, ci sono:

* Parti che si possono **eseguire in parallelo** (es. calcoli ripetitivi su molti dati).
* Parti che devono essere fatte **in sequenza**, una dopo l’altra (es. leggere i dati da un file, inizializzare una variabile, coordinare le operazioni).

Più è grande la parte **non parallelizzabile**, **meno efficace sarà il parallelismo**.

**📉 Cosa succede se aggiungiamo sempre più processori?**

All’inizio, usare più processori fa **aumentare molto le prestazioni**.

Ma dopo un certo punto:

* **Il guadagno si riduce**, perché c’è sempre una parte che non può essere divisa.
* **La comunicazione tra processori** può diventare un problema: si sprecano tempo e risorse per coordinarsi.

**🧠 Quindi, cosa dobbiamo ricordare?**

* Il calcolo parallelo è potente, ma **non fa miracoli**.
* Serve **progettare bene il software** per sfruttarlo al meglio.
* Oltre un certo numero di processori, **i vantaggi si riducono**.
* L’efficienza **dipende da quanto del lavoro può essere diviso** e da **quanto bene si coordinano i processori**.

**🧠 Cos’è la complessità computazionale?**

È il modo con cui **misuriamo quanto cresce il tempo di calcolo** al crescere della **dimensione del problema**.

Si esprime con formule come:

* **N**: numero di dati o elementi
* **N²**, **N·logN**, ecc.: crescita del lavoro necessario

Più è alta la complessità, più è **difficile e costoso** calcolare.

**LEZIONE 2**

**🧠 1. Prestazioni teoriche del processore**

Ogni CPU ha una **potenza di picco teorica**, che dipende da:

* **Frequenza di clock** (velocità in GHz),
* **Numero di core**,
* **Numero di operazioni che ogni core può fare per ciclo** (FLOPs per core).

👉 Ma questa è solo la **teoria**. In pratica si incontrano **diversi limiti**.

**⚠️ 2. Limiti principali alla scalabilità delle CPU**

**🔸 a. Limite di frequenza (clock)**

* Non possiamo aumentare indefinitamente la velocità (GHz).
* Perché?
  + Maggiore clock = **più consumo energetico**.
  + Più energia = **più calore** da dissipare.
* Le CPU moderne **non superano i 4 GHz**, altrimenti il calore sarebbe ingestibile (TDP → fino a 450W in sistemi HPC).

**🔸 b. Limite sul numero di core**

* Aggiungere core migliora le prestazioni, ma:
  + **Richiede più banda di memoria** (L2/L3 cache + RAM).
  + Troppi core possono **competere tra loro per l’accesso alla memoria**.
  + Si creano **colli di bottiglia** nella comunicazione interna alla CPU.

🧠 **Più core ≠ più veloci**, se la memoria non riesce a "nutrirli" abbastanza in fretta.

**🔸 c. Frequenza variabile con AVX**

* Alcune CPU, quando usano istruzioni **vettoriali AVX** (per il calcolo parallelo su array), **abbassano automaticamente la frequenza**.
* Questo succede per contenere il **calore generato**.
* In alcuni casi, il clock operativo con AVX è solo il **70% del valore normale**.

**🧾 Riepilogo concettuale**

| **Fattore** | **Limite pratico** |
| --- | --- |
| Clock | Max ~4 GHz per evitare surriscaldamento |
| Core | Limitato dalla banda della memoria interna |
| AVX / istruzioni vettoriali | Riduzione automatica del clock (~70%) |

**🚀 Le GPU aumentano drasticamente le prestazioni**

Le GPU moderne come le **NVIDIA H100, GH200 e H200** sono progettate specificamente per:

* **Eseguire calcoli paralleli su vasta scala**
* Supportare applicazioni **AI, deep learning, simulazioni scientifiche**
* **Massimizzare la potenza di calcolo per watt consumato**

Immagine che contiene testo, schermata, Carattere, linea

Il contenuto generato dall'IA potrebbe non essere corretto.

Immagine che contiene testo, schermata, Carattere, algebra

Il contenuto generato dall'IA potrebbe non essere corretto.

**🔥 TDP (Thermal Design Power)**

* Le GPU H100 e simili hanno un **TDP di circa 700W**.
* Questo è alto, ma **giustificato dalla potenza offerta**.
* I data center moderni sono progettati per gestire questo carico, con sistemi di **raffreddamento avanzati** (liquido, immersione, ecc.).

Il **TDP (Thermal Design Power)** è la **quantità massima di calore**, espressa in **Watt**, che un componente elettronico (come una CPU o una GPU) può generare **in condizioni normali di utilizzo** e che il **sistema di raffreddamento deve essere in grado di dissipare**.

l TDP ti dice **quanto deve raffreddare** il tuo sistema per **mantenere il processore alla temperatura corretta**, durante carichi intensi ma realistici.

**LEZIONE 3**

**🔀 1. Bisection Bandwidth – Banda di bisezione**

**📌 Cos’è?**

È la **quantità di banda passante disponibile tra due metà di un sistema** (due gruppi di nodi), **nella situazione peggiore**.

**🧠 In parole semplici:**

Se tagli a metà la rete (in due gruppi uguali), la **bisection bandwidth** misura **quanta informazione può fluire da una parte all’altra** contemporaneamente.

**🔍 Perché è importante?**

* È una **misura realistica del collo di bottiglia** massimo nella rete.
* Aiuta a **prevedere i limiti di scalabilità**: se la rete è troppo stretta al centro, **i nodi lavorano male insieme**.

💡 **Esempio**:  
In un cluster con 1000 nodi, se i primi 500 devono comunicare con gli altri 500, la bisection bandwidth dice **quanta comunicazione simultanea può avvenire senza congestione**.

**🔁 2. Oversubscription – Fattore di sovrascrittura**

**📌 Cos’è?**

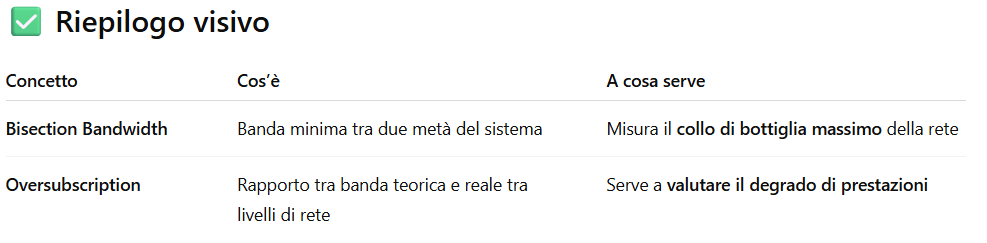
È il **rapporto tra la banda totale disponibile nei nodi di un livello e quella effettivamente disponibile quando comunicano con il livello superiore** (switch o rete backbone).

**📊 Tipici valori:**

* **1:1** → **nessun decadimento**: ogni nodo ha piena banda verso l’alto.
* **2:1** → la banda si **dimezza** tra un livello e l’altro.
* **3:1 o peggio** → **più colli di bottiglia**, peggiori prestazioni in caso di traffico intenso.

🔧 Nelle reti HPC grandi:

* Si progettano **zone locali a 1:1** (dove i nodi comunicano rapidamente tra loro).
* Ma si **accetta un po’ di oversubscription** tra zone diverse (2:1 o 3:1), per risparmiare risorse hardware (switch, cavi, porte).



**📜 Evoluzione della tecnologia InfiniBand**

InfiniBand è emersa come risposta alle limitazioni del bus PCI negli anni '90, quando la crescente domanda di dispositivi esterni e l'espansione di Internet hanno evidenziato la necessità di una comunicazione più efficiente tra i componenti del computer.

**⚙️ Caratteristiche tecniche distintive**

InfiniBand si distingue per le sue prestazioni elevate e la bassa latenza, rendendola ideale per ambienti HPC e data center. Le sue principali caratteristiche includono:

* **Elevata larghezza di banda**: supporta velocità di trasmissione dati molto elevate, essenziali per applicazioni che richiedono un ampio throughput.
* **Bassa latenza**: grazie al supporto per RDMA (Remote Direct Memory Access), InfiniBand consente trasferimenti di dati diretti tra memorie di sistema, riducendo significativamente la latenza.
* **Scalabilità**: progettata per supportare reti di grandi dimensioni, InfiniBand può essere implementata in topologie complesse senza compromettere le prestazioni.
* **Affidabilità**: offre meccanismi avanzati di gestione degli errori e controllo del flusso, garantendo una comunicazione dati robusta.

**🔄 Confronto con Ethernet**

Sebbene Ethernet sia ampiamente utilizzata, InfiniBand offre vantaggi significativi in termini di latenza e larghezza di banda. Tuttavia, Ethernet ha cercato di colmare il divario introducendo tecnologie come RoCE (RDMA over Converged Ethernet), che portano i benefici del RDMA su reti Ethernet. Nonostante questi sforzi, InfiniBand continua a essere la scelta preferita per applicazioni che richiedono le massime prestazioni.

**🔍 Conclusione**

InfiniBand rappresenta una soluzione avanzata per le esigenze di comunicazione ad alte prestazioni nei moderni ambienti di calcolo. La sua evoluzione e adozione testimoniano la sua importanza nel panorama tecnologico attuale, soprattutto in un'epoca in cui l'elaborazione di grandi volumi di dati e l'intelligenza artificiale richiedono infrastrutture di rete altamente efficienti.

**🧩 Componenti di una rete Infiniband**

L’**architettura Infiniband** è pensata per reti ad **altissime prestazioni**, con **bassa latenza** e **alta banda passante**. È molto usata nei **supercomputer e nei data center HPC**.

Immagine che contiene testo, schermata, diagramma, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

**Ecco i quattro componenti chiave:**

**🔹 1. HCA – Host Channel Adapter**

* Si trova nei **server o nodi di calcolo** (dove ci sono le CPU).
* Funziona come **interfaccia di rete** che gestisce le **richieste di trasmissione dati**.
* È collegato direttamente alla memoria e ai processori.

🔧 Nel diagramma: a sinistra, collegato alla **CPU, memoria e sistema**.

**🔹 2. TCA – Target Channel Adapter**

* È il “gemello” dell’HCA, ma si trova **sui dispositivi di destinazione**, come:
  + Sistemi di **storage**
  + Altri server target
* Riceve e gestisce i dati inviati dall’HCA.

🔧 Nel diagramma: a destra, accanto alla scritta **Target**.

**🔹 3. Switch**

* Gestisce il **traffico di rete locale** tra più HCA e TCA.
* Come in una rete Ethernet, **smista i dati** ai nodi corretti **all’interno della stessa sottorete**.

🔧 Nel diagramma: al centro, collega più **IB Link** (Infiniband Link).

**🔹 4. Router**

* Serve a **collegare sottoreti diverse** tra loro.
* È utile quando si ha un’infrastruttura molto grande, divisa in più reti separate.

🔧 Nel diagramma: in basso a destra, etichettato come **Router**, connesso via **IB o rete esterna**.

**🎯 Flusso semplificato della comunicazione**

1. Un’applicazione su CPU vuole inviare dati → passa dall’**HCA**
2. L’HCA invia i dati tramite uno **switch Infiniband**
3. Lo **switch** li inoltra al **TCA** del dispositivo destinatario
4. Il **TCA** riceve e li consegna al sistema di destinazione (es. storage)
5. Se i nodi sono su reti diverse, il traffico passa da un **Router**

**🔄 Evoluzione e sviluppo di Infiniband**

1. **Nata per sostituire PCI, Ethernet e Fibre Channel** nelle interconnessioni ad alte prestazioni.
2. **Sviluppata da un consorzio industriale** guidato da aziende come Intel, IBM, Mellanox (ora NVIDIA).
3. Supporta **Remote Direct Memory Access (RDMA)**, che consente:
   * Trasferimento diretto dei dati da una memoria all’altra
   * Minimo carico per la CPU
   * Massima efficienza energetica
4. Adottata in:
   * Supercomputer (es. Leonardo @CINECA, Frontera, etc.)
   * Cluster AI con GPU (es. NVIDIA DGX)
   * Sistemi storage ad alte prestazioni (es. parallel file system)
5. La **latenza è inferiore a quella degli switch Ethernet**: ad es. uno switch Ethernet ha ~230 ns, uno Infiniband ~100 ns.

**🧠 Cos’è RDMA?**

**RDMA (Remote Direct Memory Access)** è una tecnologia che permette a un computer di:

* **Leggere o scrivere direttamente nella memoria di un altro computer**
* Senza passare per il **sistema operativo (kernel)** né usare troppa **CPU**

In pratica, **elimina passaggi intermedi** e rende la trasmissione dei dati **molto più veloce ed efficiente**.

**❌ Come funziona la trasmissione dati con TCP/IP (il metodo tradizionale)**

1. L’applicazione vuole inviare dati.
2. I dati vengono copiati:
   * Dallo **spazio applicativo** → alla **memoria principale (RAM)**
   * Poi dalla **RAM** → alla **scheda di rete (NIC)**
3. Tutto questo passa attraverso il **sistema operativo** (kernel), che gestisce ogni passaggio.
4. Ogni fase **consuma tempo e risorse CPU**.

📉 **Problemi**:

* **Latenza** (ritardo) più alta
* **Uso intensivo della CPU**
* **Copia doppia dei dati** = inefficienza

**✅ Come funziona RDMA**

Con RDMA:

* I dati **vanno direttamente dalla memoria di un’app all’altra**, attraverso la rete.
* Non serve fare copie intermedie né passare per la CPU.
* Il sistema operativo **non viene coinvolto attivamente** nella trasmissione.

🎯 Risultati:

* **Latenza bassissima** (quasi 1 microsecondo)
* **Zero-copy**: nessuna copia intermedia → più veloce
* **Carico della CPU ridotto** → la CPU può fare altro

Immagine che contiene testo, schermata, diagramma, design

Il contenuto generato dall'IA potrebbe non essere corretto.

**🌳 1. Fat Tree**

* È una **variante ottimizzata** dell’albero classico.
* I livelli superiori (root) hanno **più banda passante** rispetto ai livelli inferiori (da qui "fat", cioè "grasso").
* Ogni collegamento verso l’alto ha banda maggiore per evitare colli di bottiglia.

📌 **Vantaggi**:

* Molto **scalabile** e **bilanciata**.
* Buona per cluster medio-grandi.

📌 **Svantaggi**:

* **Costo** più alto per via della ridondanza di collegamenti e switch.

**🔁 2. Torus**

* I nodi sono disposti in una **griglia** con connessioni ad anello (colonne e righe connesse ciclicamente).
* Può essere **2D o 3D**.

📌 **Vantaggi**:

* **Latenza uniforme**, ideale per carichi regolari.
* **Risparmio di costi** rispetto a topologie completamente connesse.

📌 **Svantaggi**:

* **Prestazioni degradano** se la comunicazione tra nodi è irregolare.
* Più difficile da scalare in grandi dimensioni.

**🐉 3. Dragonfly**

* Composto da **gruppi di nodi fortemente connessi localmente**, e da **connessioni sparse tra i gruppi**.
* Progettato per **scalabilità estrema**.

📌 **Vantaggi**:

* **Alta bisection bandwidth**.
* Ottimo per supercomputer su larga scala.

📌 **Svantaggi**:

* Architettura **più complessa da implementare**.
* Richiede **algoritmi avanzati** per il bilanciamento del traffico.

**🔷 4. Hypercube**

* Ogni nodo è connesso a un altro attraverso **dimensioni binarie**.
* Per esempio: un **4D hypercube** ha 16 nodi, ognuno connesso a 4 altri nodi.

📌 **Vantaggi**:

* **Elevata connettività** e **bassa latenza**.
* Ben studiato nella teoria dei grafi.

📌 **Svantaggi**:

* Difficile da scalare oltre un certo punto.
* Più adatto a cluster medi.

**🧮 5. HyperX**

* Variante più avanzata e **più simmetrica** del Fat Tree.
* Ogni nodo ha **connessioni multiple a ogni livello**, ottimizzando la ridondanza e riducendo la latenza.

📌 **Vantaggi**:

* Ottimo **bilanciamento del carico**.
* **Scalabilità lineare**.

📌 **Svantaggi**:

* Rete **complessa**.
* Richiede **hardware di rete avanzato**.

**LEZIONE 4**

**📈 Crescita dei dati e necessità di storage efficiente**

Il volume di dati digitali sta crescendo esponenzialmente, con una prevalenza di dati non strutturati (come video, immagini, audio) che rappresentano oltre il 90% del totale. Questa crescita impone la scelta di protocolli di storage che garantiscano:

* **Elevata larghezza di banda**: per trasferire grandi quantità di dati rapidamente.
* **Bassa latenza**: per ridurre i tempi di risposta delle applicazioni.
* **Alti IOPS (Input/Output Operations Per Second)**: per gestire numerose operazioni di lettura/scrittura simultanee.

Immagine che contiene testo, schermata, diagramma, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

L'immagine descrive **il percorso che seguono i dati** in un sistema HPC/AI **dal nodo di calcolo al sottosistema di storage e viceversa**, con un focus sui **collegamenti interni e le relative velocità di trasferimento**.

**📤 Writing data – Flusso dati in uscita dal nodo verso lo storage**

**🔷 1. RAM → PCIe (~400 GB/s)**

* I dati partono dalla **memoria RAM** del nodo.
* Vengono trasferiti attraverso il **bus PCI Express**, che collega la RAM alle periferiche (come la scheda di rete o gli acceleratori).
* **Velocità teorica**: fino a **400 GB/s** (Gigabyte al secondo).

🔍 *Questa è una stima molto alta, tipica di sistemi avanzati con memoria DDR5 e PCIe 5.0/6.0.*

**🔷 2. PCIe → HCA (~128 GB/s)**

* I dati passano dal bus PCIe all’**HCA (Host Channel Adapter)**.
* L’HCA è l’interfaccia di rete ad alte prestazioni che gestisce la comunicazione esterna (es. Infiniband).
* **Velocità stimata**: ~**128 GB/s**, sempre teorica.

📌 Questo passaggio è gestito dal **sistema operativo** del nodo.

**🔷 3. HCA → High Speed Network (~50 GB/s)**

* I dati escono dal nodo tramite l’HCA e vengono trasmessi sulla **rete ad alta velocità** (es. Infiniband, Ethernet 400G).
* Qui il **collo di bottiglia inizia a farsi sentire**: la velocità cala a circa **50 GB/s**.

🔧 *Questa parte è gestita dalla rete e dai protocolli di comunicazione (es. RDMA).*

**🔷 4. High Speed Network → Multi Tier Storage (~50 GB/s)**

* I dati arrivano al sottosistema di **storage multi-livello**, che può includere SSD, HDD, tape.
* Il file system (es. Lustre, GPFS) gestisce dove scrivere effettivamente i dati.

**📥 Reading data – Il flusso inverso**

* Il percorso è esattamente **al contrario**: i dati vengono letti dallo storage, passano attraverso la rete, l’HCA, il PCIe e arrivano alla RAM.

**📉 Nota importante: valori teorici vs reali**

“I dati di transfer rate sono teorici, nella realtà i valori misurati sono circa **80% dei valori teorici** o anche inferiori.”

Immagine che contiene testo, schermata

Il contenuto generato dall'IA potrebbe non essere corretto.

**🔷 1. Dischi a stato solido – SSD e NVMe**

**📌 Cosa sono:**

* **SSD (Solid State Drive)**: memorie a stato solido, prive di parti mobili.
* **NVMe (Non-Volatile Memory Express)**: è un **protocollo veloce** che permette agli SSD di comunicare direttamente con il sistema tramite il bus **PCI Express (PCIe)**.

**⚙️ Caratteristiche principali:**

* **Velocissimi**: fino a decine di GB/s in lettura/scrittura.
* **Bassa latenza**: tempi di accesso molto rapidi.
* **Affidabilità** elevata.
* **Zero rumore** e basso consumo.

**✅ Uso ideale:**

* Dati “caldi” e accessi frequenti.
* Database, AI training, boot rapidi, cache.

**🟦 2. Dischi magnetici – Hard Disk Drive (HDD)**

**📌 Cosa sono:**

* Dischi rotanti con testine magnetiche per leggere/scrivere dati.
* Tecnologia storica ancora molto usata per il buon rapporto **prezzo/capacità**.

**⚙️ Caratteristiche principali:**

* **Capienti**: fino a 20 TB per singolo disco.
* **Più lenti** di SSD (velocità sequenziali intorno a 100–200 MB/s).
* **Maggior latenza** e **componenti meccanici**.

**✅ Uso ideale:**

* Dati “freddi” (consultati meno frequentemente).
* Archiviazione di grandi dataset, backup temporanei.

**🟫 3. Nastri magnetici – Tape**

**📌 Cosa sono:**

* Supporti a **nastro magnetico** usati per **archiviazione a lungo termine**.
* Non accessibili in modo casuale (si leggono in sequenza).

**⚙️ Caratteristiche principali:**

* **Capacità molto alta** (oltre 50 TB per cartuccia).
* **Costi bassissimi per GB**.
* **Tempi di accesso molto lunghi**, ma ideali per conservazione di massa.

**✅ Uso ideale:**

* Backup, disaster recovery, archiviazione legale o scientifica a lungo termine.

**LEZIONE 5**

**🧩 Cos'è il RAID?**

Il RAID (Redundant Array of Independent Disks) è una tecnica che distribuisce i dati su più dischi, permettendo operazioni di input/output (I/O) sovrapposte e bilanciate, migliorando così le prestazioni complessive. Le configurazioni RAID differiscono per come distribuiscono i dati, la ridondanza e la tolleranza ai guasti. I sistemi RAID appaiono al sistema operativo come un'unica unità logica.

RAID serve a **combinare più dischi fisici in un’unica unità logica** per migliorare:

* **Affidabilità** (tolleranza ai guasti)
* **Prestazioni** (lettura/scrittura parallela)
* **Continuità operativa**

Il RAID **non sostituisce il backup**, ma riduce il rischio di **perdita dati per guasto hardware**.

**🔧 Tipi di RAID**

Esistono diversi livelli di RAID, ciascuno con caratteristiche specifiche:

**RAID 0**

Il livello più semplice è il **RAID 0**, che punta tutto sulle prestazioni. In questo schema i dati vengono "strisciati", cioè, suddivisi in blocchi e distribuiti su più dischi contemporaneamente (si parla infatti di *striping*). Questo fa sì che la lettura e scrittura dei dati sia molto veloce perché avviene in parallelo. Tuttavia, RAID 0 non offre alcuna protezione: se anche solo un disco si rompe, tutti i dati diventano inaccessibili. È una scelta adatta quando la velocità è più importante della sicurezza, come in alcune attività di elaborazione temporanea o in ambito gaming.

**RAID 1**

Al contrario, **RAID 1** mette al primo posto la sicurezza dei dati. Funziona tramite *mirroring*: ogni disco ha un duplicato perfetto su un secondo disco. In pratica, ogni volta che scrivi un dato, questo viene copiato su entrambi. Se uno dei dischi fallisce, l’altro continua a funzionare senza perdita di dati. Questo tipo di RAID è semplice e molto efficace per la protezione, ma dimezza lo spazio disponibile e non migliora la velocità di scrittura, anche se la lettura può essere un po' più rapida grazie alla ridondanza.

**RAID 2**

RAID 2 è una configurazione piuttosto rara e mai davvero adottata in sistemi commerciali. Funziona usando lo **striping a livello di bit** (non a blocchi) su più dischi e affianca ad essi dischi dedicati al controllo degli errori usando un **codice di correzione degli errori ECC**, in particolare il **codice di Hamming**. Questo schema è teoricamente molto sicuro e affidabile perché permette non solo di rilevare errori, ma anche di correggerli. Tuttavia, richiede molti dischi (alcuni solo per l’ECC) e un’architettura complessa. L’arrivo di dischi con meccanismi di correzione integrati ha reso RAID 2 superfluo. Per questo motivo non è mai stato implementato in ambito commerciale.

**RAID 3**

RAID 3 è un’altra configurazione ormai superata, ma interessante da capire. Utilizza **striping a livello di byte**, cioè i dati vengono suddivisi in byte consecutivi e distribuiti su più dischi. Un disco separato è dedicato a contenere le **informazioni di parità**, necessarie per ricostruire i dati in caso di guasto. La lettura è veloce, perché può avvenire in parallelo su più dischi. Tuttavia, tutte le scritture devono coinvolgere il disco di parità, che diventa presto un collo di bottiglia. Inoltre, RAID 3 **non supporta accessi simultanei a più file**, rendendolo poco adatto a carichi di lavoro multitasking. Era usato in ambiti specifici, come l’elaborazione video, ma è stato rimpiazzato da configurazioni più flessibili.

**RAID 4**

RAID 4 è molto simile al RAID 5, ma con una differenza importante: lo **striping avviene a livello di blocchi**, mentre la **parità è memorizzata su un disco dedicato**, come in RAID 3. Rispetto a RAID 3 migliora la gestione di più richieste contemporanee, ma mantiene lo stesso problema: il disco di parità è un punto critico, perché viene coinvolto in ogni operazione di scrittura. Anche RAID 4 è quindi stato superato da RAID 5, che distribuisce la parità su tutti i dischi e bilancia meglio il carico.

**RAID 5**

**RAID 5** è uno dei livelli più diffusi perché cerca un equilibrio tra sicurezza, prestazioni e utilizzo dello spazio. In questo caso, i dati e le informazioni di *parità* (utili per ricostruire i dati in caso di guasto) sono distribuiti tra tutti i dischi. Se un disco si guasta, i dati possono essere recuperati usando la parità salvata sugli altri. Rispetto al RAID 1, RAID 5 offre una maggiore efficienza dello spazio, perché non serve duplicare tutto. Tuttavia, le operazioni di scrittura sono un po’ più lente a causa del calcolo e della scrittura della parità. È adatto a scenari dove le letture sono più frequenti delle scritture, come nei sistemi di archiviazione dati.

**RAID 6**

**RAID 6** è una variante del RAID 5 che aumenta la tolleranza ai guasti: introduce una seconda parità, permettendo al sistema di sopravvivere anche al guasto di due dischi contemporaneamente. Questo livello è usato in ambienti dove l’affidabilità deve essere elevata, ad esempio nei data center o nei sistemi di backup di grande scala. L’aspetto negativo è che le scritture sono ancora più lente rispetto al RAID 5, e servono almeno quattro dischi per implementarlo.

**RAID 10**

Infine, c'è **RAID 10**, noto anche come RAID 1+0, che combina le strategie di mirroring e striping. In questo schema, prima si creano coppie di dischi in mirroring (come in RAID 1) e poi si applica lo striping su queste coppie (come in RAID 0). Il risultato è una configurazione molto performante e sicura, che tollera guasti multipli e garantisce ottime prestazioni. Tuttavia, richiede almeno quattro dischi e, come il RAID 1, solo metà della capacità totale è effettivamente utilizzabile. RAID 10 è una scelta eccellente per sistemi mission-critical come database o ambienti virtualizzati.

* **RAID 0 (Striping)**: Distribuisce i dati su più dischi senza ridondanza. Offre alte prestazioni ma nessuna tolleranza ai guasti.
* **RAID 1 (Mirroring)**: Duplica i dati su due o più dischi, garantendo ridondanza. Offre alta disponibilità ma utilizza più spazio di archiviazione.
* **RAID 5**: Combina striping con parità distribuita. Richiede almeno tre dischi e offre un buon equilibrio tra prestazioni, capacità e tolleranza ai guasti.
* **RAID 6**: Simile al RAID 5 ma con doppia parità, permettendo la tolleranza a due guasti simultanei.
* **RAID 10 (1+0)**: Combina mirroring e striping. Richiede almeno quattro dischi e offre alte prestazioni e tolleranza ai guasti.

Immagine che contiene testo, Carattere, linea, numero

Il contenuto generato dall'IA potrebbe non essere corretto.

Immagine che contiene testo, schermata, Carattere, numero

Il contenuto generato dall'IA potrebbe non essere corretto.

N.B: **Lo striping** è una tecnica di gestione dei dati usata nei sistemi RAID (e in altre architetture di archiviazione) che consiste nel **dividere i dati in blocchi** e distribuirli **in parallelo su più dischi**.

**🛠️ Controller RAID**

Il controller RAID gestisce l'array di dischi e può essere:

* **Hardware**: Un dispositivo fisico dedicato che gestisce l'array, offrendo migliori prestazioni e funzionalità avanzate.
* **Software**: Gestito dal sistema operativo, utilizza le risorse del sistema e può avere prestazioni inferiori rispetto a un controller hardware.
* **Firmware (o "fake RAID")**: Integrato nella scheda madre, offre una soluzione intermedia tra hardware e software, ma può gravare sulla CPU del sistema.

**📌 Conclusione**

* I RAID **0, 1, 5, 6 e 10** sono i più usati oggi.
* RAID 2, 3 e 4 sono considerati **superati o specializzati**.
* RAID è utile, ma in cluster HPC si usano sempre più **Erasure Coding** (es. Ceph, HDFS) per scalabilità e flessibilità.

**📦 Cos'è Ceph?**

**Ceph** è un sistema di **storage distribuito open-source**, progettato per essere:

* **Scalabile** (da pochi nodi a migliaia)
* **Fault-tolerant** (resistente a guasti hardware e software)
* **Autogestito** (self-healing e bilanciamento automatico)

🔧 L’unità di base di Ceph è l’**OSD (Object Storage Daemon)**:

* Ogni **Ceph OSD** gestisce un disco
* Archivia dati, replica, bilancia e comunica con altri nodi
* Un cluster Ceph può contenere **centinaia o migliaia di OSD**

**🧠 Ceph: due modelli di protezione dati**

**🔁 1. Replica (3 copie)**

📌 **Caratteristiche**:

* I dati vengono **copiati integralmente 3 volte**
* Ogni copia è conservata su un OSD diverso

✅ **Vantaggi**:

* **Alta durabilità** (puoi perdere fino a 2 dischi)
* **Recupero molto veloce**
* **Prestazioni elevate**, perché le copie possono essere lette in parallelo

❌ **Svantaggi**:

* **Spreco di spazio**: 3 copie = solo 33% efficienza reale (per ogni 3 TB usati, 1 TB di dati effettivi)

**🔍 Cos'è Erasure Coding?**

L’**Erasure Coding (EC)** è un algoritmo di **codifica ridondante** che suddivide i dati originali in più frammenti, e poi genera frammenti aggiuntivi (di parità) tali che anche se alcuni frammenti si perdono o si danneggiano, i dati originali possano comunque essere ricostruiti **senza perdita di informazione**.

* 🧠 **Evita di replicare l’intero dato** (come fa la replica classica)
* ➕ Invece crea **blocchi di parità matematici**
* ♻️ Permette di **ricostruire i dati mancanti** anche se alcuni dischi falliscono

**📦 Come funziona?**

Supponiamo di avere un file che vogliamo proteggere. L’Erasure Coding:

1. **Divide** il file in *k* pezzi di dati.
2. **Genera** *m* pezzi di parità.
3. Distribuisce i *k + m* pezzi su diversi dischi o nodi.

Finché ne hai **almeno k qualsiasi**, puoi **ricostruire tutto il file originale**, anche se fino a *m* pezzi sono andati persi.

📌 **Esempio concreto**:

* Hai un file diviso in **4 pezzi di dati** (k = 4).
* Generi **2 pezzi di parità** (m = 2).
* Hai 6 pezzi totali distribuiti su 6 dischi.  
  👉 Anche se **2 dischi si guastano**, puoi ricostruire tutto con i 4 rimanenti.

Immagine che contiene testo, Carattere, linea, numero

Il contenuto generato dall'IA potrebbe non essere corretto.

**🔧 Classificazione fisica delle architetture di storage scalabili**

Questa classificazione si basa su **come i dischi e le risorse sono distribuiti e condivisi** tra i nodi di un cluster.

**🔹 1. Share Nothing Storage Architecture**

📌 **Definizione**:  
Ogni nodo ha **dischi locali e indipendenti**. Non esiste condivisione diretta del disco con altri nodi.

📦 **Caratteristiche**:

* Ogni nodo gestisce i **propri dati**, il suo **OS**, la **home directory**, ecc.
* **Nessuna interdipendenza fisica** tra nodi: se un nodo cade, **non blocca gli altri**
* Elevata **scalabilità e resilienza**.
* Si può estendere con software che **simula la condivisione via rete** (→ share anything)

✅ **Pro**:

* Alta affidabilità e isolamento dei guasti
* Ottimo per workload paralleli indipendenti (es. cluster scientifici, Ceph)

❌ **Contro**:

* Più complesso da gestire globalmente (es. no file system condiviso nativamente)

**🔹 2. Scale-Out Storage Architecture**

📌 **Definizione**:  
È un’architettura **a blocchi modulari**, in cui ogni unità (es. nodo o server) può contenere storage **interno** ma condivide risorse attraverso la rete.

🧱 Ogni “blocco” può:

* Essere un server con dischi
* Aggiungersi al cluster senza dover cambiare gli altri componenti

📦 **Caratteristiche**:

* **Condivisione logica** (non fisica) → "share anything"
* Prestazioni e capacità **crescono linearmente** aggiungendo nodi
* I nodi collaborano per gestire uno spazio dati unico

✅ **Pro**:

* Scalabilità lineare → aggiungi nodi per più spazio o potenza
* Bilanciamento del carico automatico
* Tipico in sistemi come **Ceph, GlusterFS, Lustre**

❌ **Contro**:

* Complessità software e di gestione (orchestrazione, replica, fault handling)

**🔹 3. Cluster di Sottosistemi Storage**

📌 **Definizione**:  
Un’architettura che **raggruppa più sottosistemi storage** (es. appliance RAID, NAS, SAN) in un ambiente **software centralizzato**.

📦 **Caratteristiche**:

* Sottosistemi simili/identici (es. NAS IBM, DDN, NetApp)
* Visti come un'unica entità dal punto di vista logico
* **Condivisione tramite rete** (es. NFS, iSCSI, S3)

✅ **Pro**:

* Architettura consolidata in ambienti enterprise
* Elevata capacità e affidabilità
* Facile da integrare in ambienti già esistenti

❌ **Contro**:

* Costi elevati per hardware specializzato
* Scalabilità limitata rispetto allo scale-out puro



**✅ In sintesi:**

* **Share-nothing** è semplice e robusto → ottimo per cluster autonomi.
* **Scale-out** è il modello moderno per **scalabilità orizzontale** e gestione distribuita.
* **Cluster di sottosistemi** è più **tradizionale**, adatto a chi ha infrastrutture già in uso.

**🔄 Scale-Up Storage: Espansione Verticale**

Il modello tradizionale di **scale-up** prevede l'aggiunta di risorse (come dischi) a un sistema esistente per aumentare la capacità di storage. Tipicamente, un sistema scale-up utilizza una coppia di controller centrali che gestiscono più shelf di dischi.

**✅ Vantaggi:**

* **Semplicità iniziale**: Facile da implementare e gestire in ambienti con esigenze di crescita modeste.
* **Utilizzo consolidato**: Approccio familiare e ampiamente adottato nelle infrastrutture IT tradizionali.

**❌ Svantaggi:**

* **Scalabilità limitata**: Una volta raggiunta la capacità massima dei controller, è necessario sostituire l'intero sistema.
* **Silos di storage**: L'aggiunta di nuovi sistemi può portare a isolamenti di dati e complessità nella gestione.
* **Protezione dei dati**: Basata su RAID, che non si estende oltre i controller, limitando la resilienza.

**🔁 Scale-Out Storage: Espansione Orizzontale**

Il modello **scale-out** consente di aumentare la capacità aggiungendo nuovi nodi al sistema, ciascuno con le proprie risorse di storage e computazionali. Questi nodi lavorano insieme come un cluster unificato.

**✅ Vantaggi:**

* **Scalabilità illimitata**: Facile aggiunta di nodi per aumentare capacità e prestazioni.
* **Resilienza**: Utilizzo di RAIN (Redundant Array of Independent Nodes) per la protezione dei dati, offrendo maggiore tolleranza ai guasti rispetto al RAID tradizionale.
* **Gestione semplificata**: Unificazione dello storage in un unico namespace, facilitando l'accesso e la gestione dei dati.

**❌ Svantaggi:**

* **Complessità iniziale**: Richiede una pianificazione più attenta e una comprensione approfondita dell'architettura distribuita.
* **Costi iniziali**: Potrebbero essere più elevati a causa dell'infrastruttura necessaria per supportare l'espansione orizzontale.

**🧭 Conclusione**

La scelta tra scale-up e scale-out dipende dalle esigenze specifiche dell'organizzazione:

* **Scale-Up**: Adatto per ambienti con crescita prevedibile e limitata.
* **Scale-Out**: Ideale per organizzazioni che prevedono una crescita significativa dei dati e necessitano di alta disponibilità e resilienza.

**🧩 Classificazione logica dello storage scalabile: i tre principali modelli**

Questa classificazione si basa sul **modo in cui i dati vengono gestiti e accessi**, indipendentemente da come sono fisicamente distribuiti. I tre modelli principali sono:

**1. 📦 Block Storage**

**Cos'è:**  
Divide i dati in **blocchi di dimensioni fisse** (es. da 512 KB a diversi MB), ciascuno con un identificativo. Non gestisce direttamente file o cartelle: questo viene lasciato al sistema operativo o al file system.

**Caratteristiche:**

* Usato in **SAN (Storage Area Network)**.
* Adatto a **workload ad alte prestazioni**, come database, email server, macchine virtuali.
* **Non contiene metadati**, perciò è efficiente ma “cieco” al contenuto.
* I blocchi sono **accessibili e modificabili direttamente** a basso livello.

**Vantaggi:**

* **Prestazioni elevate** (lettura/scrittura a bassa latenza).
* Supporta **accessi simultanei concorrenti** a sotto-porzioni di dati.
* Adatto per **dati strutturati** (es. database relazionali).

**Svantaggi:**

* **Maggior complessità nella gestione** (necessita di file system esterni).
* **Non facilmente scalabile** rispetto all’object storage.

**2. 🗂️ File Storage (NAS)**

**Cos'è:**  
Gestisce i dati tramite un **file system** gerarchico (cartelle, file), accessibile via protocolli di rete come **NFS**, **SMB/CIFS**. Tipico dei dispositivi NAS.

**Caratteristiche:**

* Ogni file ha il suo **percorso, nome e metadati** (dimensione, data, permessi).
* Accesso via rete, **trasparente per l’utente** come se fossero cartelle locali.
* I protocolli **NFS** (usato su Linux/Unix) e **SMB** (su Windows) sono i più comuni.

**Vantaggi:**

* Facile da usare e implementare.
* Ottimo per ambienti condivisi: uffici, scuole, piccole imprese.
* Supporta **file locking**, utile per la **collaborazione simultanea**.

**Svantaggi:**

* **Meno performante** in ambienti di grandi dimensioni.
* Gerarchia rigida e meno adatta a **big data** o **cloud-native**.

**3. ☁️ Object Storage**

**Cos'è:**  
Memorizza i dati come **oggetti indipendenti**, ciascuno con un identificatore unico e metadati associati. Non ha gerarchie (no cartelle).

**Caratteristiche:**

* Utilizza API (es. **Amazon S3**, **Azure Blob**, **Ceph RADOS**).
* Ottimo per **dati non strutturati** (immagini, video, log, backup).
* Supporta metadati **ricchi e personalizzabili**.
* È "eventualmente consistente" (non sempre in tempo reale, ma scalabile).

**Vantaggi:**

* **Altamente scalabile** (petabyte ed oltre).
* Perfetto per il **cloud e l’intelligenza artificiale**.
* Ottimo per backup, archiviazione e content delivery.

**Svantaggi:**

* Non compatibile con applicazioni legacy (es. che richiedono un file system).
* **Assenza di file locking nativo**, richiede gateway software.

**📊 Benchmark per architetture storage ad alte prestazioni**

Questi strumenti servono a **misurare le performance reali** del sistema storage, in termini di **velocità di lettura/scrittura, accesso concorrente, latenza**, ecc. Sono fondamentali per:

* Validare nuovi sistemi
* Confrontare configurazioni
* Ottimizzare applicazioni I/O intensive

**🔹 IOR (Interleaved or Random)**

📌 **Cos’è**:  
IOR è uno dei benchmark **standard più diffusi** per testare **file system paralleli** su cluster HPC.

🔧 **Caratteristiche principali**:

* Misura **throughput aggregato** in lettura/scrittura sequenziale e casuale
* Supporta **MPI** → può simulare migliaia di processi concorrenti
* Permette test con variabili personalizzabili: dimensione blocchi, numero di thread, modalità di accesso

💡 **Tipico per**:

* Lustre, IBM Spectrum Scale (GPFS), BeeGFS, CephFS
* Analisi scientifica, simulazioni, AI training distribuito

**🔹 IOZone**

📌 **Cos’è**:  
Altro benchmark file-level, simile a IOR ma **oggi meno usato** nelle architetture più moderne.

🔧 **Funzioni**:

* Misura vari tipi di operazioni: read, write, re-read, random read/write, mmap
* Output dettagliato su prestazioni in funzione della dimensione dei file

💡 **Usato per**:

* Testare prestazioni su singoli nodi o NAS
* Valutare file system locali, USB, NAS legacy

❗ **Meno adatto a sistemi paralleli distribuiti**

**🔹 IOPS (Input/Output Operations Per Second)**

📌 **Cos’è**:  
Una **metrica**, non un benchmark in sé. Misura **quante operazioni di I/O** il sistema può eseguire **in un secondo**.

🔧 **Particolarmente importante per**:

* Sistemi **flash/SSD/NVMe** ad alta velocità
* Accessi a **piccoli blocchi** (es. 4 KB)
* Workload con molte operazioni brevi e frequenti

💡 **Tipico per**:

* Database, server web, ambienti virtualizzati (VM)
* Sistemi enterprise ad alta reattività

📐 **Attenzione**:  
Alte IOPS ≠ sempre alte prestazioni complessive! Va interpretata con latenza e dimensione blocco.

**LEZIONE 6**

**⚡️ Efficienza energetica nei sistemi HPC a larga scala (Panoramica)**

Con l’aumento esponenziale della **potenza computazionale** (supercomputer con milioni di core, GPU, acceleratori), l’**energia consumata** è diventata una **variabile critica** sia per ragioni economiche sia ambientali. L'efficienza energetica è ormai un criterio centrale nella progettazione dei sistemi HPC.

**🔋 1. Cos'è il PUE – Power Usage Effectiveness**

**PUE = Total Power / IT Power**

* **IT Power**: energia usata da CPU, GPU, RAM, storage.
* **Total Power**: energia totale assorbita, inclusi raffreddamento, UPS, luci, ecc.

📌 **Valori tipici**:

* PUE ideale: **1.0** (tutta l’energia va al calcolo)
* Data center moderni: **1.1 – 1.4**
* Sistemi inefficienti: >2.0

🔧 Obiettivo: **ridurre il PUE**, ovvero aumentare la quota di energia realmente usata per il calcolo.

**🧠 2. Potenza di Calcolo per Watt**

Le architetture vengono valutate per la loro **efficienza energetica**, misurata in:

* **GigaFLOPs/W**
* **TeraFLOPs/W**
* ⚙️ TOP500 Green500: classifica dei supercomputer più efficienti al mondo

📌 Esempi:

* CPU classiche: ~10-20 GFLOPs/W
* GPU recenti (es. NVIDIA H100): ~40-50 GFLOPs/W
* Architetture custom (Fugaku, Aurora, Frontier): superano i **60-80 GFLOPs/W**

**📊 3. Tecniche per aumentare l’efficienza energetica**

**🟢 a. Scheduling energetico**

* Job scheduler consapevoli dei consumi (es. SLURM con power capping)
* **Dynamic Voltage and Frequency Scaling (DVFS)**: abbassa dinamicamente la frequenza di CPU/GPU per risparmiare
* **Power-aware workload placement**: distribuisce i carichi tra nodi meno stressati termicamente

**🟡 b. Gestione del carico**

* Esegui carichi “a basso wattaggio” in fasce orarie più costose (load shifting)
* Evita over-provisioning (eseguire troppi job inutili)

**🔵 c. Monitoring energetico**

* Sensori integrati nei nodi
* Report dettagliati su consumo e temperatura

**❄️ 4. Raffreddamento: aria vs acqua**

**🔹 Raffreddamento ad aria**

* Ventole, scambiatori, condizionamento
* Limite: sopra i **250-300W per nodo**, diventa poco efficace
* Più economico da installare, meno efficiente

**🔸 Raffreddamento a liquido / acqua diretta (DLC, direct-to-chip)**

* Porta via il calore direttamente da CPU/GPU
* Supporta nodi **oltre 1000W**
* Consente alta densità nei rack (più nodi nello stesso spazio)
* Più efficiente ma più costoso e complesso

✅ Sistemi exascale come **Frontier (ORNL)**, **LUMI (Europe)** e **Leonardo (CINECA)** usano **raffreddamento a liquido**

**🏗️ 5. Progettazione di Data Center Efficienti**

Punti chiave:

* **Layout intelligente**: flussi d’aria separati, corridoi caldo/freddo
* **Energy reuse**: uso del calore residuo per riscaldamento ambienti
* **Modularità**: sezioni attivabili/scalabili a seconda dei carichi
* **Fonti rinnovabili**: alimentazione solare, eolica, geotermica

**🔥 Cos'è il TDP?**

**TDP = Thermal Design Power**  
È la **quantità massima di calore (in Watt)** che un componente elettronico, come una **CPU o GPU**, può generare sotto carico **tipico o massimo**, e che il **sistema di raffreddamento** deve essere in grado di dissipare in modo sicuro.

🧊 **Non è il consumo elettrico massimo**, ma è **strettamente correlato**.

**🧮 Esempio pratico:**

* Una **CPU con TDP di 125W** significa che il dissipatore o il raffreddamento (ad aria o liquido) deve essere **progettato per smaltire almeno 125 Watt di calore.**
* Se il TDP è **sottodimensionato**, la CPU può:
  + Rallentare (thermal throttling)
  + Surriscaldarsi
  + Danneggiarsi a lungo termine

**⚙️ Dove viene usato il TDP?**

* **CPU e GPU**: Intel, AMD, NVIDIA indicano il TDP nei datasheet
* **Supercomputer**: per dimensionare il raffreddamento (es. Leonardo ha nodi > 450W, Frontier usa GPU > 700W)
* **Data center**: per stimare consumo, raffreddamento e densità rack

**⚖️ Token per second vs Megawatt (MW)**

Nel contesto delle **AI Factory** (grandi centri computazionali per LLM come GPT, BERT, ecc.), si misura:

* **Token/s** = velocità con cui un modello genera output (efficienza di calcolo)
* **MW** = potenza elettrica assorbita (costo energetico dell’infrastruttura)

🔍 Quando l’efficienza migliora (più token/s per ogni MW), ci si aspetterebbe **meno energia consumata**… ma non è sempre così. Entra in gioco:

**📈 Il Paradosso di Jevons**

🧠 **Definizione**:

Quando l’efficienza di utilizzo di una risorsa aumenta, il **costo unitario cala**, ma ciò può causare un **aumento complessivo del consumo** della risorsa, perché la domanda totale cresce.

**🔄 Applicazione all’IA:**

* Se una AI factory raddoppia l’efficienza (es. 2× token/s per MW), il **costo per token scende**.
* Questo incentiva **nuovi utilizzi**: chatbot ovunque, LLM embedded, inference su edge, ecc.
* Risultato: la **domanda cresce così tanto** da consumare **più energia** di prima.

💬 È lo stesso effetto visto nella storia:

* Auto più efficienti → le persone guidano di più
* Luci a LED → illuminazione usata in più contesti
* CPU/GPU efficienti → training più frequente e con modelli più grandi

**🔍 Esempio pratico:**

* 2023: 1M token/s → 10 MW
* 2025: 10M token/s → 15 MW  
  (→ efficienza ×10, ma consumo ×1.5)

La **domanda ha superato il risparmio** ottenuto con l’efficienza.

**🧩 Conclusione**

Il **paradosso di Jevons** ci insegna che:

* L’efficienza è necessaria, ma **non sufficiente** a garantire sostenibilità.
* Serve una **governance dell’uso dell’IA**, bilanciando accesso, potenza, e utilità reale.

**🧠 Scalabilità delle prestazioni nei processori: limiti fisici e architetturali**

**🔹 Formula base:**

**Immagine che contiene testo, Carattere, ricevuta, bianco

Il contenuto generato dall'IA potrebbe non essere corretto.**

* **Clock** = Frequenza del processore (es. 3.5 GHz)
* **fops** = Operazioni in virgola mobile per ciclo (es. 16–32 per architetture a 64 bit)
* **cores** = Numero totale di core nella CPU

**🔩 1. Limiti sul clock**

* I clock delle CPU **non possono salire all’infinito**, perché:
  + Aumentano **i consumi** energetici (in modo quadratico)
  + Aumenta il **calore generato**
  + Superati i **4–5 GHz**, diventa **insostenibile il TDP**

📌 Oggi le CPU arrivano **fino a 4 GHz** nei core ad alte prestazioni, ma con **TDP anche > 400 W**.

**🔧 2. fops e iops: precisione e parallelismo**

* **fops (floating point operations)** = operazioni in virgola mobile (FP64, FP32)
  + 64 bit (precisione doppia): **16–32 fops/ciclo**
* **iops (integer operations)** = operazioni intere (32 bit): **32–64 iops/ciclo**

🔍 Le architetture moderne (Intel, AMD, ARM) includono **unità vettoriali (SIMD)** come AVX, che permettono di eseguire **più operazioni per ciclo** → parallelismo intrinseco.

**🔻 3. Limiti nel numero di core**

Per scalare oltre i limiti del clock, si aumentano i **core** per CPU (es. 64, 96, 128).

Ma qui nascono nuovi limiti:

* **Bandwidth della memoria condivisa (L2, L3 cache)** diventa col collo di bottiglia
* Accessi concorrenti ai dati possono creare **latenze**
* Serve **NUMA balancing** (distribuzione intelligente del carico sui socket)

**❗️ 4. Il clock non è “unico”**

* Alcune istruzioni (es. **AVX-512** su Intel) causano un abbassamento **dinamico** del clock della CPU.
* Questo succede perché sono **molto energivore**, quindi per rientrare nel TDP la CPU **si autoregola**.

📌 Es. su AVX512:  
Clock normale = 3.6 GHz → con AVX512 = 2.5 GHz  
👉 Questo può ridurre le prestazioni **se non gestito correttamente**.

**📊 Esempio concreto**

Una CPU a 3 GHz con:

* 64 core
* 16 fops per ciclo

📌 **Perf teorica = 3 × 16 × 64 = 3072 GFLOPs (3.07 TFLOPs)**  
Ma nel mondo reale:

* Clock effettivo può essere ridotto
* Accessi alla memoria rallentano l’esecuzione
* La **Peak Performance reale è sempre inferiore alla teorica**

**📜 Moore’s Law – La legge di Moore**

**🔍 Osservazione empirica**

“Il numero di transistor su un chip raddoppia approssimativamente ogni 24 mesi.”

Questa legge è stata formulata da **Gordon Moore** (co-fondatore di Intel) nel 1965 ed è stata valida per decenni, anche se oggi comincia a mostrare dei limiti.

🔹 **Effetti diretti:**

* **I transistor diventano più piccoli** (“features shrink”)
* **Le dimensioni fisiche del chip possono aumentare** (“semiconductor dies grow”) → più spazio per altri transistor

**⚡ Impatto sulle prestazioni**

Grazie a questo raddoppio continuo:

* Le **prestazioni dei chip** sono aumentate di circa **1000 volte in 20 anni**.

💡 Questo è stato possibile perché:

* I chip sono stati **ottimizzati a livello di microarchitettura**
* I **transistor più veloci** hanno permesso **clock più alti**
* Più transistor = più **unità logiche parallele**, più cache, più core

**🧠 In parole semplici:**

* Immagina un processore nel 2000 con 10 milioni di transistor.
* Nel 2020, secondo Moore, avrebbe avuto **oltre 10 miliardi** (e infatti… molti moderni ne hanno più di 100 miliardi).

📉 Tuttavia, oggi **la legge di Moore sta rallentando**:

* I limiti fisici del silicio si avvicinano
* I costi di miniaturizzazione aumentano
* La densità non è più sufficiente da sola: serve innovazione architetturale (es. chiplet, 3D stacking, acceleratori AI)

**❄️ 1. Capacità estrattiva del calore**

**Formula approssimativa:**  
**Q = C × portata × ΔT**

* **Q** = calore estratto
* **C** = capacità termica del fluido (aria o acqua)
* **portata** = quanto fluido passa (es. m³/s o l/s)
* **ΔT** = differenza di temperatura tra ingresso e uscita

📌 Valori tipici:

* **Aria (Cₐ):** ~0,24 cal/g·°C
* **Acqua (Cₕ₂ₒ):** ~1 cal/g·°C
* Quindi **l'acqua è ~4 volte più efficace dell'aria** nel trasferire calore a parità di massa.

👉 **Conclusione**: a parità di condizioni, il raffreddamento a liquido è **molto più efficiente** di quello ad aria.

**💨 2. Raffreddamento ad aria**

* 💡 Il metodo più comune nei server standard.
* Aria ambiente viene spinta attraverso ventole per raffreddare CPU, RAM, GPU, ecc.
* **Range temperatura in ingresso**: **25°C – 32°C**

📉 **Limiti**:

* Non adatto a server ad alta densità (> 300W per nodo)
* Più rumoroso e meno efficiente
* Rischio di "hot spot"

Un **hot spot** è un **punto localizzato** all’interno del sistema (come un data center, un rack o anche una singola scheda madre) in cui la **temperatura è significativamente più alta** rispetto all’ambiente circostante.

**💧 3. Raffreddamento ad acqua nei nodi (D2C – Direct to Chip)**

* L’acqua scorre in **piastre metalliche** o tubi a contatto diretto con CPU/GPU.
* È **molto più efficace** nel trasferire calore rispetto all’aria.
* **Range tipico**: **30°C – 40°C**

✅ **Vantaggi**:

* Supporta nodi > 1000W
* Riduce la necessità di raffreddamento dell’aria ambientale

**💧 4. Raffreddamento ad acqua nei rack**

* L'acqua viene fatta scorrere **in serpentine o scambiatori** montati all'interno dei rack stessi.
* Utilizzato per raffreddare l'intero armadio/server rack.
* **Range tipico**: **18°C – 22°C**

✅ Più efficiente rispetto all’aria, ma meno diretta del D2C.

**🧪 5. Raffreddamento a liquido a due fasi**

* Utilizza un **fluido speciale** che **evapora** assorbendo calore (fase 1) e poi si **condensa** in un’altra area (fase 2), trasferendo calore molto velocemente.
* Esempi: 3M Novec, Fluorinert

**Range tipico**: **30°C – 40°C**

✅ Estremamente efficiente  
❗ Più costoso e complesso da implementare

**🛁 6. Raffreddamento immersivo (Immersion Cooling)**

* I server vengono **totalmente immersi** in un liquido dielettrico (non conduttivo).
* Calore trasferito direttamente dal componente al liquido.

**Range tipico**: **≤ 30°C**

✅ Massima efficienza  
✅ Ideale per HPC e AI (GPU fino a 700–1000W)  
❗ Richiede hardware compatibile e progettazione su misura

**Questi sono i tre indicatori energetici fondamentali usati nei data center e ambienti HPC:**

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

Immagine che contiene testo, schermata, Carattere, numero

Il contenuto generato dall'IA potrebbe non essere corretto.

**🧠 In sintesi:**

* **PUE** → efficienza del data center **nel suo insieme**
* **ITUE** → efficienza **interna ai server**
* **ERE** → capacità del data center di **recuperare energia** anziché sprecarla

**DEF RACK**: Un **rack è un armadio metallico standardizzato** usato nei data center per **alloggiare in modo ordinato e modulare** i componenti IT come server, switch e storage, ottimizzando spazio, raffreddamento e gestione.

**🚫 1. Limite fisico del raffreddamento ad aria (Rack ≥ 35 kW IT)**

* Anche con aria **fredda (≤ 25°C)**, non si riesce a rimuovere efficacemente tutto il calore generato da rack ad alta densità.
* **Motivo fisico:** l'aria ha una **bassa capacità termica** (cp ≈ 1 kJ/kg·K), quindi serve una grande quantità di aria per assorbire calore → inefficienza.

🔴 **Conclusione**: l’aria **non basta oltre i 35–40 kW/rack**

**⚡ 2. Costo energetico del raffreddamento ad aria**

* Nei data center tradizionali, **oltre il 30%** dell’energia totale viene usata solo per raffreddare con aria.
* Questo **aumenta il PUE** (Power Usage Effectiveness), rendendo il sistema meno efficiente dal punto di vista energetico ed economico.

**♻️ 3. Soluzioni ibride aria-acqua**

* Una **combinazione** di:
  + Aria fredda frontale
  + **Pannelli posteriori raffreddati a liquido**
* **Fino a 40 kW/rack**, questo sistema può essere efficace.
* Ma **oltre**, si raggiunge un limite termico → non abbastanza efficiente.

**💧 4. DLC parziale (Direct Liquid Cooling)**

* Si raffreddano **direttamente CPU e GPU**, le componenti più calde.
* Rimuove fino al **75% del calore** con acqua → molto più efficace dell’aria.
* Richiede ancora **aria fredda per RAM, VRM e storage**, ma in misura minore.

**💡 5. DLC + pannelli acqua = raffreddamento quasi totale**

* Combinando:
  + **Raffreddamento diretto a liquido (DLC)** su CPU/GPU
  + **Pannelli posteriori ad acqua** per dissipare il resto
* Si arriva a **quasi il 100% del calore estratto via liquido**
* Ma **serve comunque aria fredda frontale** (≤ 32°C) per alcuni componenti non liquido-raffreddati.

**🚿 6. DLC completo (full-liquid) per rack ≥ 70 kW**

* Quando un rack raggiunge o supera i **70 kW**, **solo il raffreddamento liquido completo** può gestire il carico termico.
* Esempi: sistemi per AI (con GPU tipo H100), cluster HPC, acceleratori AI.

**LEZIONE 7**

Analizziamo l’evoluzione **dell’hardware nei sistemi HPC e AI di nuova generazione**, suddiviso per tre aree principali: **CPU**, **GPU** e **rete (Network)**.

**🔹 CPU (Central Processing Unit)**

* 📈 **Numero di core in aumento**: da 64 a 128 fino a 256 core per singola CPU.  
  → Questo consente di elaborare molte più istruzioni in parallelo, rendendo la CPU molto più potente per compiti intensivi.
* 💾 **HBM – High Bandwidth Memory** integrata:  
  → Memoria ad alta velocità posizionata molto vicino al processore (spesso sullo stesso chip), per ridurre la latenza e aumentare la velocità di accesso ai dati.
* ⚡ **Consumo energetico in crescita**:  
  → Le CPU di ultima generazione possono arrivare a **500 Watt** di potenza assorbita. Serve un raffreddamento adeguato e un'infrastruttura energetica robusta.

**🔹 GPU (Graphics Processing Unit)**

* 🔄 **CPU e GPU nello stesso socket**:  
  → In alcune architetture avanzate, si integrano CPU e GPU in un singolo componente hardware, riducendo la latenza di comunicazione e migliorando le prestazioni.
* 💾 **HBM anche sulle GPU**:  
  → Le GPU possono avere **fino a 3 Terabyte al secondo (TB/s)** di banda passante con la memoria, essenziale per applicazioni AI e Deep Learning.
* ⚡ **Consumi molto elevati**:  
  → Le GPU ad alte prestazioni possono richiedere **fino a 1000 Watt**. Questo le rende estremamente potenti ma anche energivore.

**🔹 Network (Rete di interconnessione)**

* 📡 **Aumento della banda per link di rete**:  
  → I collegamenti di rete tra nodi o dispositivi arrivano a **400 Gbps**, permettendo trasferimenti ultra-rapidi tra server, GPU e storage.
* 🔁 **RoCE (RDMA over Converged Ethernet)**:  
  → Protocollo che consente trasferimenti diretti in memoria via Ethernet, riducendo l’uso della CPU e la latenza.
* 🔗 **Interconnessione interna tra GPU nel nodo**:  
  → La connessione diretta tra GPU all’interno dello stesso server può arrivare a **1 TB/s**, fondamentale per il calcolo parallelo distribuito nei grandi modelli AI.

**✅ In sintesi**

Questa sezione mostra come la **scalabilità delle architetture AI e HPC** si basi su:

* Più core per CPU,
* Memorie HBM ad altissima velocità,
* Connessioni interne ed esterne ultra-performanti,
* Un consumo energetico sempre più critico, da gestire con attenzione.

C’è stato un **aumento esponenziale** delle risorse necessarie per **addestrare i modelli di intelligenza artificiale (AI)**, in particolare quelli di tipo **neural network transformer**, come GPT-3.

**▶️ 1. GPT-3 (175 miliardi di parametri)**

* Ha richiesto **più di 1000 volte** la potenza di calcolo rispetto a modelli come BERT (2018).
* Occupa **centinaia di GB di memoria** solo per l'inferenza.

**▶️ 2. Trend crescente (2018 → 2020+)**

* In appena **2 anni**, la complessità sia cresciuta in modo **esponenziale**.
* Dal modello BERT base (110M parametri) a GPT-3, e ora verso modelli **multi-trillion** (es. MT-NLG da 530B e MSFT-1T da 1 trilione di parametri).

**🔮 Futuro prossimo:**

* Si prevede lo sviluppo di **modelli AI con trilioni di parametri**.
* Questi richiederanno **petabyte di memoria** e **exaflop-days** per l'addestramento.
* Implicheranno **enormi sfide di scalabilità**, alimentazione, raffreddamento e parallelizzazione.

**✅ Cosa deduciamo da tutto ciò?**

* L’**AI non scala linearmente**: la crescita dei parametri porta a una **crescita esponenziale dei requisiti di calcolo e memoria**.
* Serve **hardware estremo** (GPU, interconnessioni ultraveloci, HBM, sistemi distribuiti).
* L’evoluzione di modelli AI sta trainando **lo sviluppo delle architetture HPC**.

**🔷 COS'È NVLINK?**

**NVLink** è un’**interconnessione ad alta velocità** sviluppata da **NVIDIA** per collegare direttamente tra loro **GPU**, **CPU**, e altri componenti come **memorie HBM** o **switch NVSwitch**, **bypassando** le tradizionali interfacce come **PCIe**.

È pensato per superare i limiti di banda, latenza e scalabilità del PCI Express, soprattutto per applicazioni che richiedono **enorme scambio di dati tra GPU** come:

* AI/Deep Learning
* HPC (High Performance Computing)
* Simulazioni scientifiche
* Analytics avanzati

**⚙️ COME FUNZIONA**

**🔗 Architettura base**

* NVLink **collega direttamente** GPU-GPU, GPU-CPU o GPU-switch.
* Ogni **collegamento NVLink** è composto da più **linee seriali** bidirezionali.
* Più collegamenti = più banda totale disponibile.

Immagine che contiene testo, schermata, Carattere, numero

Il contenuto generato dall'IA potrebbe non essere corretto.

**🧠 VANTAGGI PRINCIPALI**

**1. Altissima Banda Passante**

* Migliaia di gigabyte al secondo tra GPU → fondamentale per il **training distribuito**.

**2. Bassa Latenza**

* NVLink è progettato per essere **molto più veloce** di PCIe nella comunicazione tra nodi interni.

**3. Memory Pooling**

* Le GPU **possono condividere memoria** tra loro tramite NVLink → creano un **grande spazio di memoria virtuale condivisa**.

**4. Scalabilità lineare**

* È possibile **collegare decine di GPU tra loro** nello stesso rack (es. 72 GPU nei SuperPOD NVIDIA).

Immagine che contiene testo, schermata, diagramma

Il contenuto generato dall'IA potrebbe non essere corretto.

Immagine che contiene testo, schermata, Carattere, numero

Il contenuto generato dall'IA potrebbe non essere corretto.

**Immagine che contiene testo, schermata, Carattere, diagramma

Il contenuto generato dall'IA potrebbe non essere corretto.**

**🔷 Parte alta: Intra-node data transfer to High Perf Network**

**🔁 Flusso dati verso la rete ad alte prestazioni (es. Infiniband con HCA)**

* **CPU/GPU → HBM3**: ~3 TB/s  
  (accesso interno ai dati in memoria ad alta velocità)
* **HBM3 → HCA (Host Channel Adapter)**: ~128 GB/s
* **HCA → Network (es. RoCE o Infiniband)**: ~50 GB/s

🔽 **Collo di bottiglia**:

* La rete può trasferire solo 50 GB/s rispetto ai 3000 GB/s iniziali → **efficienza teorica ≈ 0,02 (2%)**

📌 **Conclusione**: anche se la CPU/GPU e la memoria sono molto veloci, il vero limite arriva nella **velocità della rete esterna**.

**🔷 Parte bassa: Intra-node data transfer to NVLINK protocol**

**🔁 Flusso dati interno tra GPU tramite NVLINK**

* **CPU/GPU → HBM3**: ~3 TB/s
* **HBM3 → NVLINK**: ~1 TB/s
* **NVLINK → Altra GPU**: ~1 TB/s

🔼 **Efficienza molto maggiore**:

* NVLINK può mantenere un throughput elevato → **efficienza teorica ≈ 0,3 (30%)**

📌 **Conclusione**: la comunicazione **intra-nodo** tra GPU con NVLINK è molto più efficiente rispetto al trasferimento verso l’esterno via rete.

Immagine che contiene testo, Carattere, linea, schermata

Il contenuto generato dall'IA potrebbe non essere corretto.

Vediamo ora l'evoluzione del Data Center (DC) e come si stanno trasformando le architetture di calcolo e connessione per renderle più efficienti, modulari e scalabili.

Immagine che contiene testo, diagramma, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

**1. TMO vs UMO**

**TMO – Traditional Mode of Operation**

* Architettura classica dei data center.
* **Ogni server è dotato di CPU e GPU** integrate e connesse in locale tramite interconnessioni proprietarie (es. NVLink).
* La rete è il backbone di clustering, ma le risorse (GPU, CPU, ecc.) sono fisse e locali.
* Questo porta a **bassa flessibilità e riuso**, e a un impiego non ottimale delle risorse.

**UMO – Ultimate Mode of Operation**

* Architettura **disaggregata e modulare**.
* Le GPU non sono più integrate nei server ma raggruppate in un **“pool di risorse”** condivisibili da più nodi.
* Collegate tramite **tessuti ottici (optical fabrics)** che agiscono come switch/patch panel PCIe.
* Vantaggi:
  + **Maggiore riuso delle risorse**.
  + Riduzione del **CapEx** (capitale investito).
  + Scalabilità e flessibilità superiore.

Immagine che contiene testo, diagramma, schermata, Piano

Il contenuto generato dall'IA potrebbe non essere corretto.

**2. RoCE vs Disaggregated Design**

**RoCE – RDMA over Converged Ethernet**

* Architettura con risorse fisse per ogni nodo.
* Le GPU sono montate localmente nei server.
* Comunicazione tra nodi avviene tramite rete Ethernet ad alta velocità con RDMA.
* Buone prestazioni, ma **le risorse non sono riutilizzabili dinamicamente**.

**Disaggregated Design**

* Separazione completa delle risorse (CPU, GPU, storage).
* Le GPU sono **centralizzate in rack dedicati** e collegate otticamente ai server.
* Permette **allocazione dinamica** delle risorse in base al carico di lavoro.
* Ideale per scenari AI/HPC in cui i carichi cambiano spesso.
* Migliora **efficienza operativa** e riduce i costi.

**In sintesi**

Il passaggio da architetture tradizionali (TMO/RoCE) a quelle disaggregate (UMO/Disaggregated Design) rappresenta un'evoluzione importante nei data center moderni. Questo approccio:

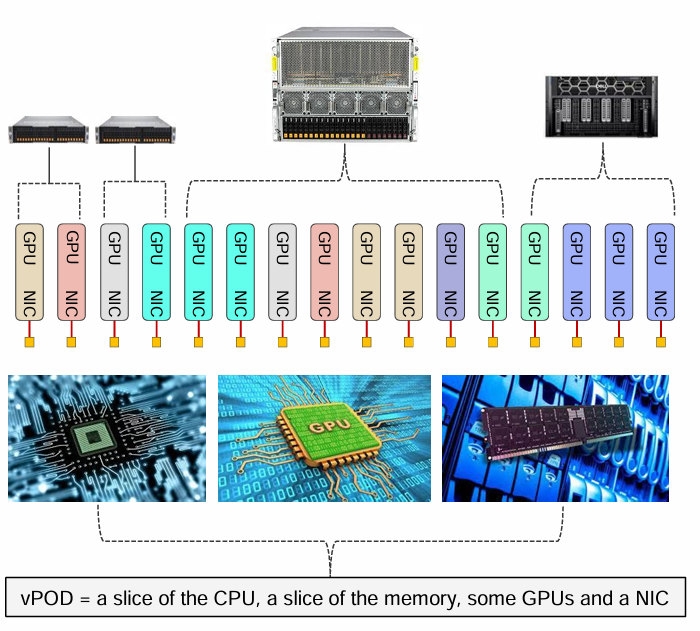
* Riduce sprechi.
* Aumenta flessibilità.
* Ottimizza i consumi e l’utilizzo delle risorse.
* È particolarmente efficace in ambiti come AI e HPC, dove i requisiti cambiano rapidamente e la scalabilità è fondamentale.

**🔍 Cos'è un vPOD?**

Un **vPOD** è un'unità logica che raggruppa risorse hardware condivise (CPU, GPU, memoria, NIC – schede di rete) per formare un'entità isolata e sicura per un **utente o workload specifico**.

**📌 Caratteristiche principali dei vPODs**

* **🔄 Riconfigurabili dinamicamente**:  
  Consentono una **configurazione flessibile** delle risorse, superando le limitazioni degli ambienti hardware statici tradizionali. È possibile riassegnare risorse a seconda delle esigenze del carico di lavoro.
* **🌐 Possono attraversare macchine fisiche differenti**:  
  Un vPOD può essere costituito da CPU, GPU e memoria **provenienti da più server fisici**, rendendo possibile un’allocazione distribuita ma coerente.
* **🧠 Network back-end basato su GPU (RoCE)**:  
  Utilizzano una rete ad alta velocità (100/200/400 Gbps) per collegare le GPU tra loro sfruttando **NIC RoCE (Remote Direct Memory Access over Converged Ethernet)**. Questo garantisce latenza molto bassa nelle comunicazioni tra GPU, come richiesto nei carichi AI.
* **🔐 Sicurezza e isolamento**:  
  Ogni vPOD è isolato dagli altri e rappresenta una **unità di elaborazione sicura** e dedicata per l'utente o l'applicazione, migliorando la gestione multi-tenant nei data center.
* **🏢 Ideali per aziende e cloud provider**:  
  In particolare per chi gestisce molte GPU e vuole evitare sprechi, permettendo **un’allocazione granulare e ottimizzata** delle risorse.



"vPOD = una fetta di CPU, una fetta di memoria, alcune GPU e una NIC"

Questo significa che ogni vPOD è una "porzione virtuale" di un'infrastruttura fisica completa, assemblata su richiesta.

**✅ Vantaggi**

* Miglior **utilizzo delle risorse**.
* Possibilità di **scalare** facilmente senza riprogettare l'hardware.
* Massima **flessibilità per workload dinamici**, come quelli in AI/ML o in ambienti cloud multi-tenant.
* **Riduzione del CapEx e OpEx**, dato che le risorse sono condivise e non sottoutilizzate.

Collegare i **vPODs** con le **architetture disaggregate** significa capire come queste due tecnologie si completano per massimizzare flessibilità, efficienza e scalabilità nei data center moderni, specialmente in ambito **AI** e **cloud**.

**🔧 Cos’è un’architettura disaggregata?**

Nelle **architetture tradizionali (TMO)**, CPU, GPU, RAM e storage sono **fisicamente fissi** in un server. Questo limita l’utilizzo efficiente delle risorse: una GPU può restare inutilizzata se il suo server è spento o satura in CPU.

Con le **architetture disaggregate (UMO)**:

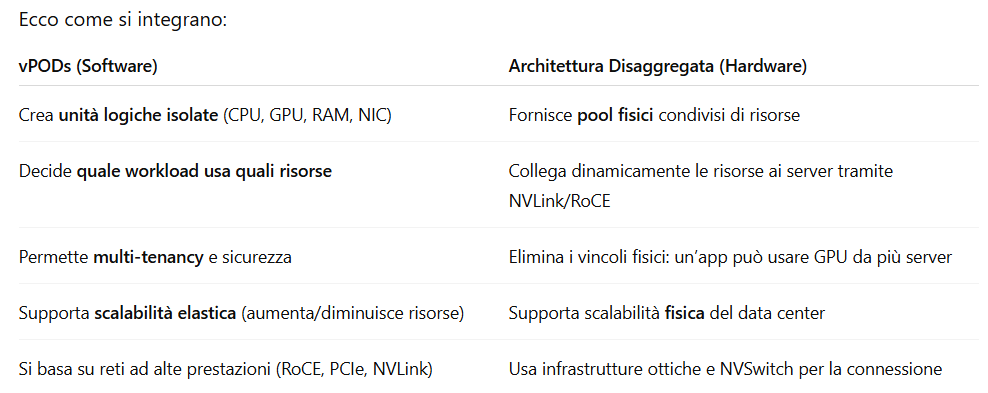
* Le **risorse sono separate fisicamente** (disaggregate) e raccolte in **pool** (es. gruppi di GPU, RAM, storage).
* I server (front-end) si connettono dinamicamente a queste risorse tramite **reti ottiche** o **interconnessioni ad alte prestazioni** (come NVLink o RoCE).
* Le risorse vengono assegnate **on demand**, in base alle esigenze del carico di lavoro.

**🧩 Come si inseriscono i vPODs?**

I **vPODs** sono la **componente software/logica** che gestisce e **orchestra** le risorse fornite dall’architettura disaggregata.

Le **vPODs** sono **unità virtuali** che combinano dinamicamente risorse disaggregate in gruppi isolati e sicuri:

* Una vPOD può includere **una o più CPU**, **una porzione di memoria**, **una o più GPU**, e **una scheda di rete**.
* È **ri-configurabile** al volo, permettendo ad esempio di passare da 8 server con 8 GPU ciascuno → a 64 server con 1 GPU ciascuno, o a qualsiasi altra combinazione.



**🎯 In sintesi**

* Le **architetture disaggregate** sono l’infrastruttura fisica moderna del data center.
* I **vPODs** sono lo **strato di astrazione** che consente agli utenti o ai carichi di lavoro di **utilizzare solo le risorse necessarie**, provenienti da più dispositivi fisici.
* Insieme, permettono di:
  + Ridurre i costi (meno risorse sprecate)
  + Aumentare l'efficienza (più workload gestiti con le stesse risorse)
  + Garantire flessibilità e isolamento (multi-tenant e sicuro)

Adesso vediamo **le sfide della scalabilità dei cluster GPU per l'addestramento di grandi modelli di deep learning** e introduciamo un’alternativa proposta da **Cerebras WSE2**, un’architettura su singolo chip pensata per superare questi limiti.

**🧩 Le 3 forme di parallelismo tradizionale nei cluster GPU**

**1. Data Parallel**

* **Ogni GPU elabora copie del modello su input diversi** (es. batch di dati differenti).
* ✅ Vantaggi: semplice da implementare.
* ❌ Limiti:
  + **Ogni GPU deve contenere una copia completa del modello** → Problemi con modelli molto grandi (es. GPT-3 da 175B parametri).
  + **Sincronizzazione dei pesi** tra GPU crea overhead.

**2. Pipelined Model Parallel**

* Il modello è diviso in **segmenti/livelli**, e ogni GPU elabora una parte della rete.
* I dati fluiscono **a pipeline** tra GPU.
* ✅ Vantaggi: consente di gestire modelli troppo grandi per una singola GPU.
* ❌ Limiti:
  + **Overhead comunicativo tra GPU** (trasferimento attivazioni).
  + **Memoria delle attivazioni cresce quadraticamente** con la profondità della rete (N²).

**3. Tensor Model Parallel**

* **Divisione interna dei tensori** (matrici dei pesi e delle attivazioni) tra GPU.
* Ogni GPU calcola solo una **parte delle operazioni di ogni layer**.
* ✅ Vantaggi: estrema granularità.
* ❌ Limiti:
  + Partizionamento e sincronizzazione **molto complessi**.
  + **Gravissimo overhead comunicativo**.

📉 **Morale:** tutte queste tecniche presentano **complessità e colli di bottiglia comunicativi** che crescono con il numero di GPU → “**Distribution complexity scales dramatically with cluster size**”.

**🚀 Cerebras WSE2: un’alternativa su chip singolo**

Il **Cerebras CS-2** utilizza un **singolo chip (WSE2)** con:

* Migliaia di core,
* Memoria on-chip vicina,
* Nessuna comunicazione esterna tra nodi (niente cluster!).

🧠 Questo approccio:

* Elimina **comunicazioni inter-GPU**,
* Rende superflua la gestione del parallelismo distribuito,
* Permette l’addestramento di modelli fino a **20 miliardi di parametri** su un solo sistema,
* Semplifica la vita a chi sviluppa modelli: **non serve essere esperti di HPC, ma solo di ML**.

**🧪 Considerazioni finali**

🔬 **Problema attuale:**

L’addestramento di modelli giganti (GPT, BERT, T5...) richiede enormi cluster e grande expertise in calcolo distribuito.

🛠️ **Soluzione Cerebras:**

Spostare tutta la computazione in un singolo chip per **ridurre complessità, costi e tempi**.

**Confronto tra l’approccio tradizionale su cluster GPU e l’architettura monolitica di Cerebras (WSE2)**.

**🧠 Problemi nello scaling tradizionale su GPU cluster**

* **Data Parallelism:** ogni dispositivo esegue il modello completo su batch diversi di dati. Problema: **i limiti di memoria per i parametri**.
* **Pipeline Model Parallelism:** il modello è diviso in sezioni (layer), eseguite in sequenza su dispositivi diversi. Problema: **overhead di comunicazione** e uso di **N² memoria per attivazioni**.
* **Tensor Model Parallelism:** ciascun layer è suddiviso in più dispositivi. Problema: **partizionamento complesso** e **elevato overhead**.

❗ **Conclusione:** più aumenta il numero di GPU, più cresce la **complessità distribuita**, **latenza** e **fragilità** (sincronizzazione, errori, tuning...).

Immagine che contiene testo, schermata, diagramma, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

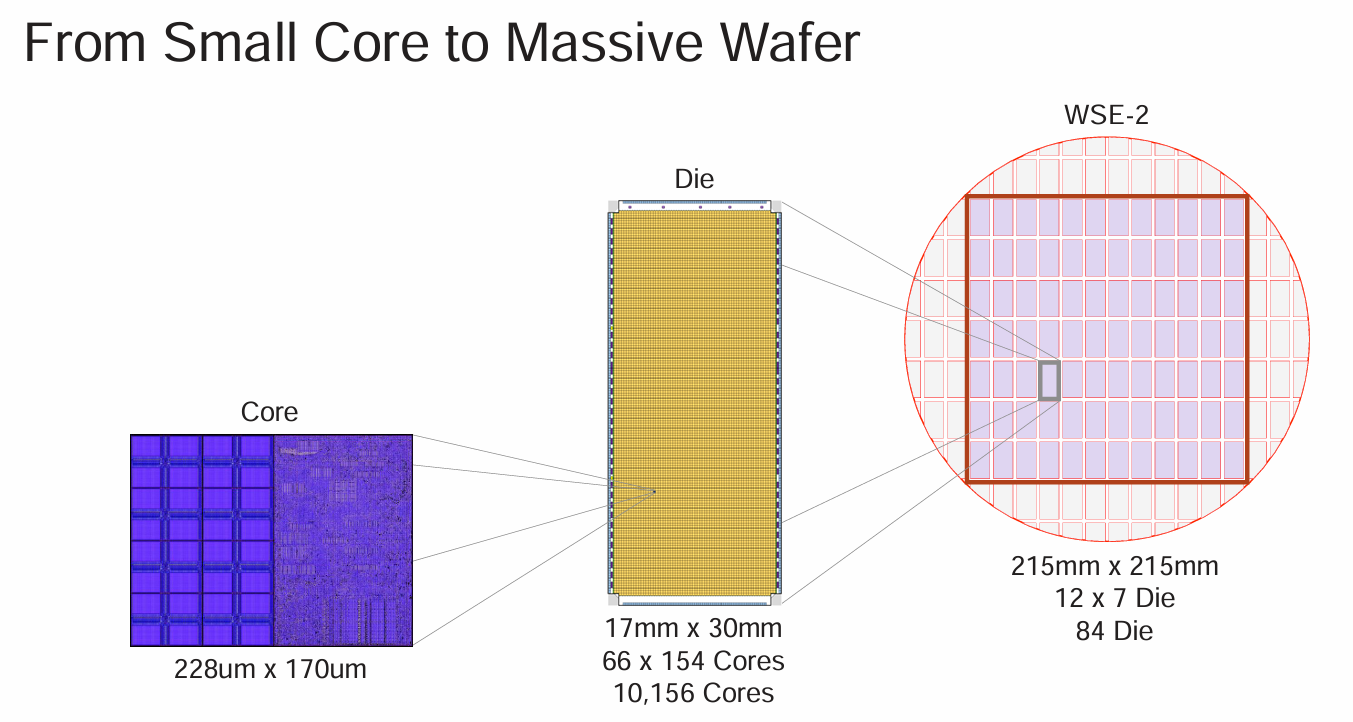
**🧩 Cerebras WSE2: un approccio radicalmente diverso**

L’immagine successiva mostra l’architettura **wafer-scale**:

* Il **WSE2** (Wafer Scale Engine 2) è un singolo chip di **215 x 215 mm**, realizzato con **84 die** collegati.
* Ogni die contiene **10.156 core**, ottimizzati per AI.
* Totale: **850.000 core** circa in un unico “pezzo” di silicio.
* Tutti i core hanno **accesso locale alla memoria** e sono **interconnessi con una rete on-chip ad alta larghezza di banda**.

💡 Quindi:

* Nessuna latenza di rete esterna
* Nessun partizionamento del modello
* Nessuna gestione esplicita del parallelismo



**Architettura:**

* È un singolo **chip grande quanto un wafer** (215 mm × 215 mm).
* Ospita **850.000 core** e **2.6 trilioni di transistor**.
* Tutti i core sono connessi tramite una **rete on-chip mesh bidirezionale**, con **banda interna di 220 Pb/s**.
* Nessuna necessità di partizionare il modello: tutto viene caricato su un unico chip.

**Vantaggi:**

* ✅ **Zero overhead di comunicazione**: non ci sono limiti di rete tra core.
* ✅ **Nessun partizionamento modello**: modelli fino a 20B parametri possono essere gestiti senza slicing.
* ✅ **Latenza minima**: tutti i core comunicano in 1 ciclo di clock.
* ✅ **Efficienza energetica**: consuma **10 volte meno** e offre **7 volte più banda** rispetto a GPU standard (es. A100).
* ✅ **Supporto a sparsità**: ottimizzato per operazioni con matrici sparse → performance fino a 75 PFLOPS sparse.

Mostriamo ora una **visione di trasformazione radicale dell’R&D tecnico** (Ricerca e Sviluppo), spinta dall’integrazione di tecnologie cognitive (come l’AI) nei processi di conoscenza, simulazione ed esperimento.

**🧠 Problemi dell’R&D Tradizionale**

1. **Knowledge**
   * Informazioni altamente non strutturate.
   * Accesso lento e limitato da capacità umane.
   * Approccio umano poco scalabile.
2. **Inference & Simulation**
   * Mancanza di automazione nelle simulazioni.
   * Configurazioni complesse, tempo-intensive.
   * Richiede molto capitale umano.
3. **Evidence & Experiments**
   * Basato su brute force empirico.
   * Spesso isolato dal contesto globale.
   * Non sfrutta il sapere esistente in modo efficiente.

**🚀 Approccio Futuro: R&D Cognitivo**

**Cambiamento chiave: dalla scoperta opportunistica umana a quella sistematica e automatizzata**

* **Knowledge Space:**  
  Le informazioni (documenti, immagini, formule) sono organizzate e indicizzate in spazi semantici, accessibili con ricerca contestuale e non solo keyword.
* **Deep Search:**  
  Motori semantici ed NLP estraggono fatti da dati non strutturati e costruiscono una **Knowledge Graph DB** ad alte prestazioni.
* **Inference:**  
  Algoritmi ML e DL trovano correlazioni, predizioni, gap di conoscenza, e decidono quali simulazioni siano più rilevanti da eseguire.

**LEZIONE 8**

**🔧 Specifiche di progetto e requisiti base**

**1. Progetto basato su criteri prestazionali**

Significa che la progettazione dell’intera infrastruttura è guidata **dalle performance richieste**, non solo dal costo o dallo spazio. L’obiettivo è massimizzare l’efficienza e la potenza di calcolo disponibile, valutando ogni componente con attenzione.

**🧠 Le tre macro-aree prestazionali da considerare:**

**🖥️ 1. Calcolo (CPU e GPU)**

* **General Partition (CPU)**: Serve per elaborazioni tradizionali (simulazioni, analisi, pre/post-processing).
* **Booster Partition (GPU)**: Potenza dedicata ai carichi AI, deep learning, e simulazioni massive.
* La prestazione complessiva si misura spesso in **teraflop (TFs)** o **petaflop (PFs)**.

**💽 2. Storage**

* Deve supportare **grande capacità** e **alte velocità di lettura/scrittura**, specialmente se si usano filesystem paralleli.
* La performance si valuta in **GB/s o IOPS**, e si divide su più livelli (scratch, archive, backup).

**🌐 3. Rete di interconnessione**

* È il “sistema nervoso” del cluster.
* Serve per:
  + Scambio dati tra nodi CPU/GPU.
  + Accesso allo storage.
  + Comunicazione tra software distribuiti (es. MPI).
* Si valuta per **larghezza di banda (Gbs/Tbs)** e **latenza (μs)**.

**⚡ Criteri di efficienza energetica – TCO Optimization**

Il progetto non si limita alle prestazioni, ma include:

* **TCO** = Total Cost of Ownership → misura il **costo complessivo nel tempo** (hardware, elettricità, raffreddamento, manutenzione).
* Ottimizzare l’**efficienza energetica** significa:
  + Scegliere tecnologie a **basso consumo/Watt per TF**.
  + Usare sistemi di **raffreddamento efficienti** (es. aria + acqua o DLC).
  + Bilanciare **prestazioni e costi operativi** nel lungo periodo.

**✅ In sintesi**

* Le prestazioni **non dipendono solo dalla CPU o GPU**,
* L’**efficienza della rete** e dello **storage** sono altrettanto cruciali,
* Tutto va valutato insieme al **consumo energetico** e ai **costi operativi**, soprattutto in architetture di grandi dimensioni.

Vediamo i **componenti “non computazionali”** che completano un’infrastruttura HPC, fondamentali per il suo funzionamento operativo, amministrativo e per l'interazione con gli utenti.

**🔧 Componenti Aggiuntivi – Non solo prestazioni**

Anche se il cuore di un sistema HPC è nella potenza di calcolo (CPU, GPU, rete e storage), ci sono **componenti essenziali** per:

* la **gestione del sistema**,
* l’**interazione degli utenti**,
* il **monitoraggio** e la **manutenzione**.

**🧑‍💻 Nodi di front-end**

Questi nodi **non eseguono i calcoli** del cluster, ma servono agli **utenti** per:

* **Sottomettere i job** al sistema (tramite workload manager come SLURM o PBS);
* **Compilare il codice**;
* **Visualizzare i risultati**, anche con interfacce grafiche (per esempio via VNC o X11 forwarding);
* **Interagire in modalità interattiva** prima di lanciare job più pesanti.

✅ Sono l’interfaccia tra l’utente e il cluster.

**🛠️ Nodi di management (mgmt)**

Sono riservati al **superuser (amministratore di sistema)** e servono per:

* **Accendere/spegnere i nodi**;
* **Diagnosi di errori hardware e software** (con alert e notifiche);
* **Verifica dei consumi energetici** per nodo o per gruppo;
* **Manutenzione predittiva e programmata**: ad esempio, sapere in anticipo se un nodo sta per fallire (grazie a metriche e log di sistema).

✅ Sono essenziali per l’affidabilità e la continuità operativa del sistema HPC.

**🌐 Rete di front-end**

È la rete che collega l’HPC con la **LAN aziendale o universitaria**. Serve per:

* Connessione **SSH remota** degli utenti;
* Accesso ai dati di input/output;
* Aggiornamenti software;
* Accesso web a interfacce di monitoraggio.

🧩 È distinta dalla **rete ad alta velocità** usata tra i nodi di calcolo (es. Infiniband o Ethernet 100/400 Gbps).

**💻 Ambienti Software di gestione e accesso**

Sistemi che permettono:

* La gestione del carico di lavoro (scheduler);
* La creazione e monitoraggio dei job;
* L’accesso sicuro (es. LDAP, Kerberos);
* L’installazione di **moduli software** (programmi, librerie, ambienti conda, container Singularity);
* Il **monitoraggio dello stato di sistema** (con software come Ganglia, Nagios, Prometheus, ecc.).

**✅ In sintesi**

Questi componenti non aumentano direttamente le **prestazioni computazionali**, ma **abilitano, semplificano e rendono affidabile** l’intero ecosistema HPC. Senza di essi, il cluster non sarebbe utilizzabile in modo efficiente o sostenibile.

**✅ Punti chiave delle conclusioni**

**1. Non solo prestazioni**

Le **prestazioni massime** non sono l’unico parametro da considerare. Una buona progettazione considera anche:

* **Budget disponibile**
* **Tempi di realizzazione**
* **Efficienza energetica**
* **Spazi fisici**
* **Affidabilità a lungo termine**
* **Flessibilità futura**

**2. Ottimizzazione costo/prestazione (HW e SW)**

* Bisogna valutare il **rapporto tra costo e prestazioni** non solo per i processori e le GPU, ma anche per:
  + rete di interconnessione
  + storage
  + software di gestione
  + licenze
  + costi operativi e di manutenzione

**3. Uso di tecnologie di prossima generazione**

* Se l’infrastruttura sarà realizzata **nei prossimi mesi/anni**, si può **considerare hardware in fase di rilascio** (es. nuove CPU, GPU, reti).
* Questo può portare **vantaggi competitivi** se pianificato con attenzione (rischi vs. benefici).

**4. Flessibilità e affidabilità dell’architettura**

* Un sistema ben progettato deve essere:
  + **Scalabile** (facile da espandere)
  + **Manutenibile**
  + **Ridondante** dove necessario (per evitare interruzioni)
  + **Compatibile** con più carichi di lavoro (generale, AI, simulazioni, visualizzazione, ecc.)

**5. Efficienza energetica**

* Fondamentale per tenere bassi i costi operativi (OPEX).
* L’efficienza energetica non riguarda solo i nodi di calcolo, ma anche:
  + sistemi di raffreddamento
  + alimentazione elettrica (UPS, PDU)
  + gestione carichi di lavoro
  + metriche come **PUE**, **ITUE**, **ERE**

**6. Vincoli fisici e temporali**

* Ogni infrastruttura ha limiti fisici:
  + spazio disponibile nei rack
  + capacità dell’impianto elettrico e di raffreddamento
* I **tempi di consegna e installazione** sono vincoli pratici da rispettare, specie per bandi o progetti pubblici.

**7. Proposte innovative**

* Anche se **non richieste esplicitamente**, è consigliato:
  + **proporre soluzioni migliorative**
  + suggerire approcci alternativi più vantaggiosi (es. architetture disaggregate, raffreddamento a liquido, storage ibrido, CPU-GPU integrate)
* Ma **solo se fattibili** in termini di costi, tempi e compatibilità con il progetto.

**🎯 In sintesi**

La progettazione di un’infrastruttura HPC o AI moderna è un **compromesso intelligente**: non si punta solo alla potenza, ma si bilancia tutto per costruire un sistema **sostenibile, scalabile e affidabile**, capace di rispondere a esigenze attuali e future.