elicsir
Elevating Italian Computer Science Innovation and Research

Tutti i nodi vengono al Server

Il sistema Parameter Server per il training distribuito

Marco Fasoli
Università degli Studi di Genova

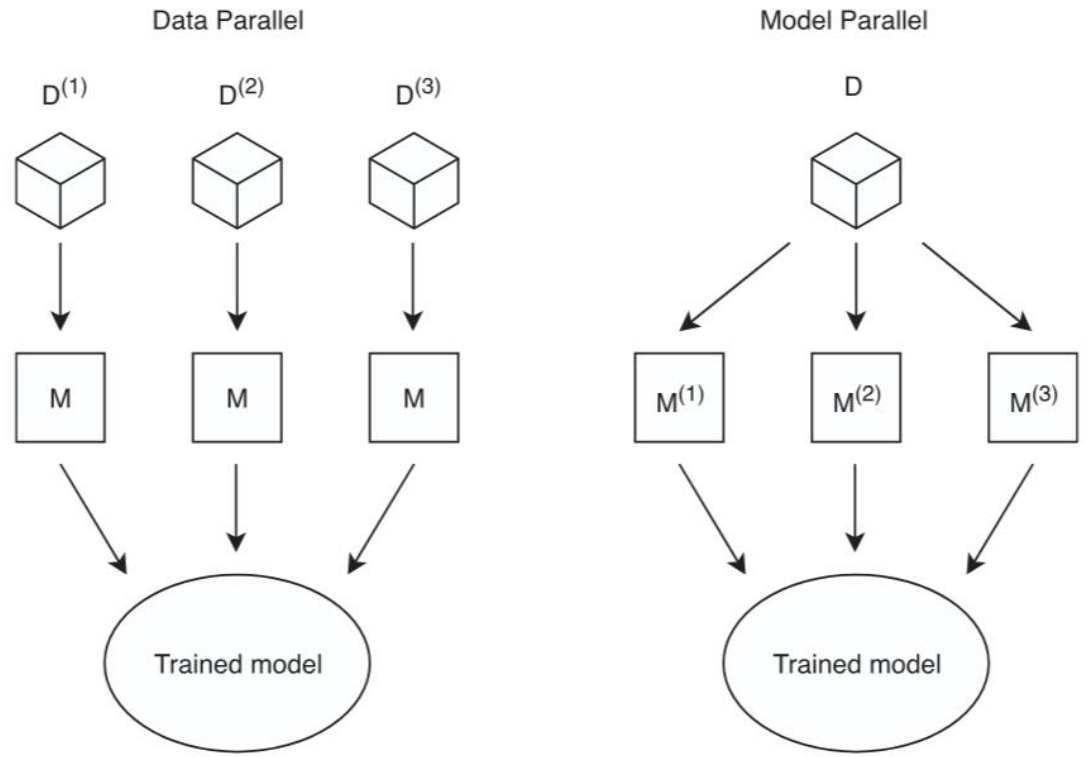
Valeria Cardellini
Università degli Studi di Roma Tor Vergata

Introduzione

- Il **Machine Learning distribuito** consiste nell'allenare i modelli suddividendo il lavoro su più macchine
- Due sfide:
  - **Mole di dati:** dataset difficilmente memorizzabili e processabili per singole macchine. Esempio: dataset nell'ordine dei PetaByte
  - **Complessità dei modelli:** numero di parametri fuori portata per singole GPU. Esempio: modelli attuali nell'ordine di $\sim 10^{12}$ parametri
- Questo richiede uno **scale out** del sistema

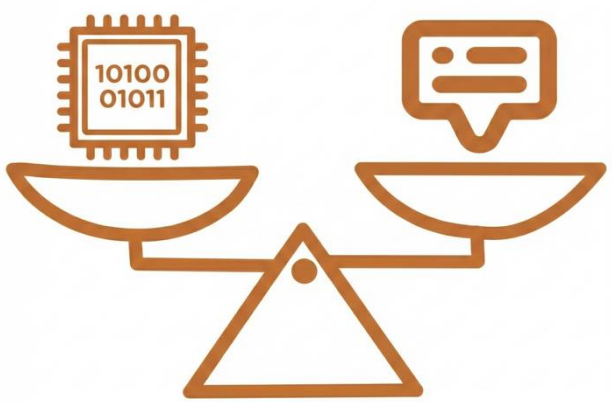
Strategie di Scaling

- **Data Parallelism:** modello replicato su ogni worker e il dataset è suddiviso in porzioni diverse
- **Model Parallelism:** parametri del modello suddivisi tra i vari nodi del sistema



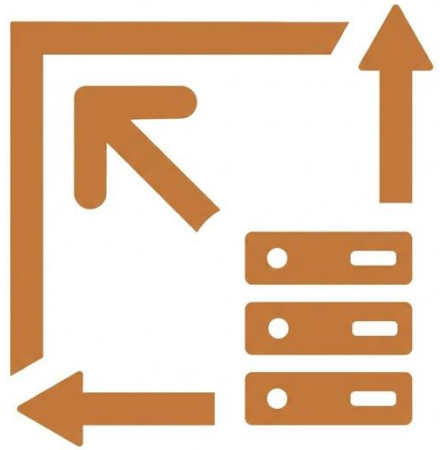
Le Sfide dello Scaling

Efficienza di Comunicazione



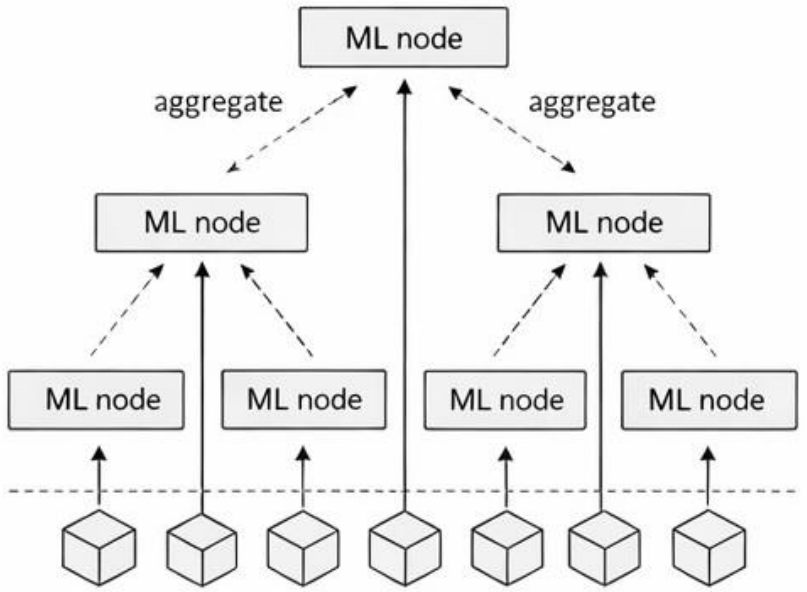
Computazione vs. Comunicazione

Tolleranza ai Guasti

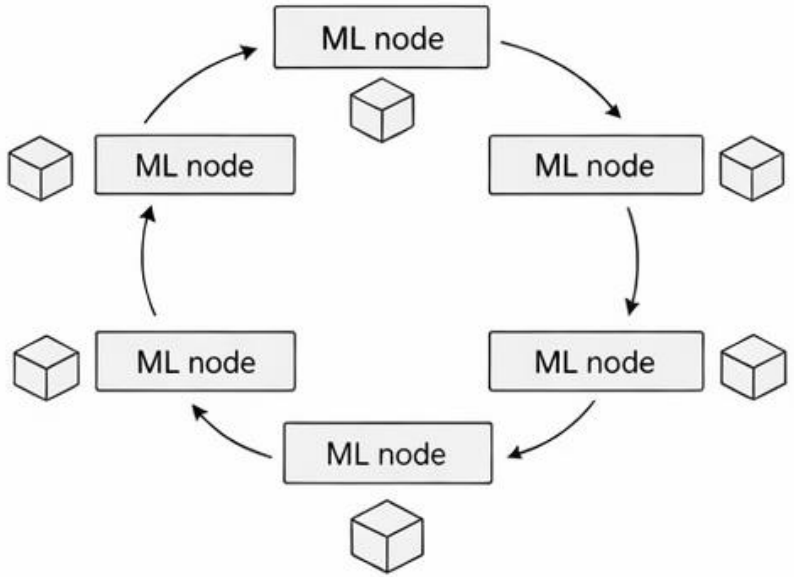


Scalabilità Dinamica

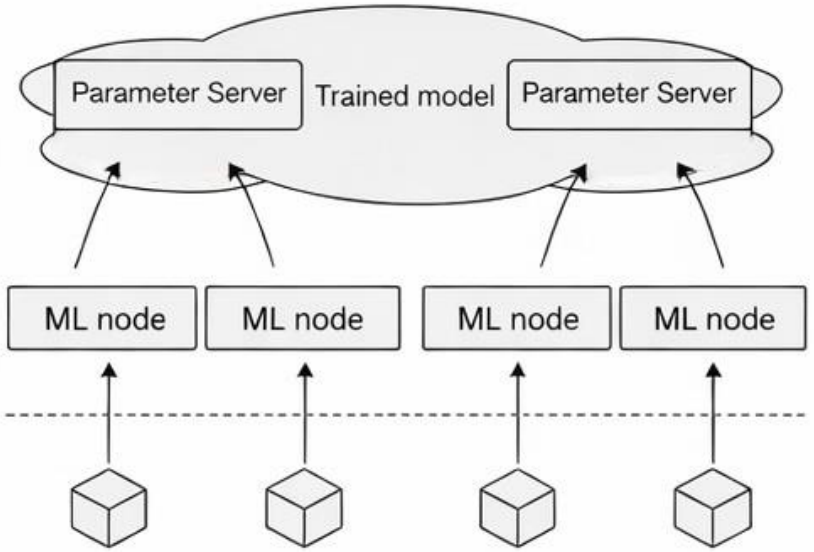
Panoramica delle Architetture



Decentralizzato (Albero)



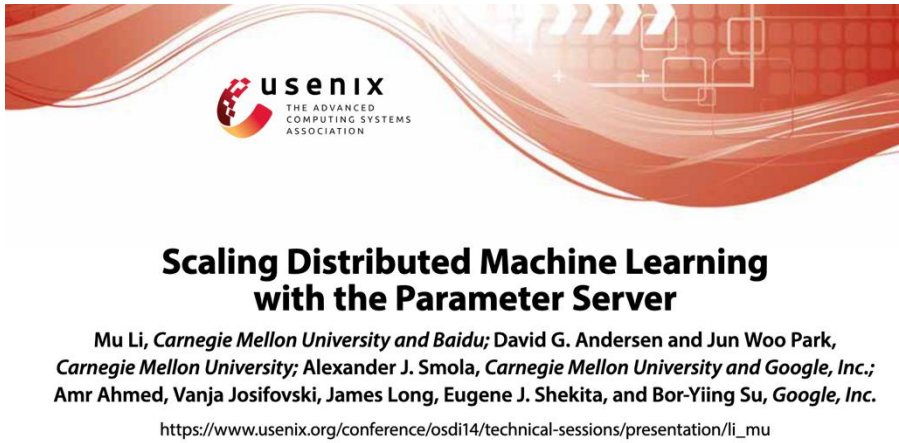
Decentralizzato (Anello)



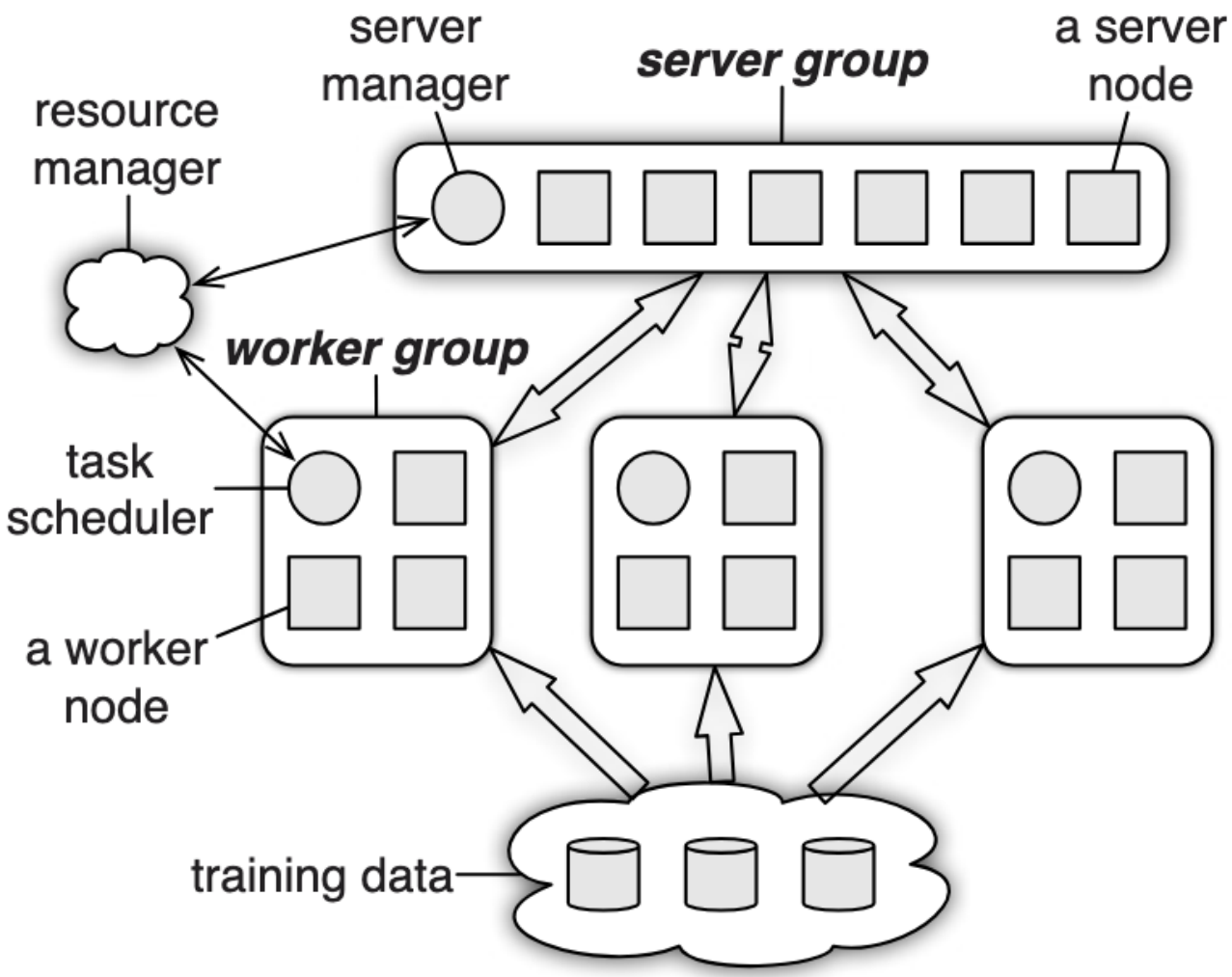
Centralizzato (Parameter Server)

Focus e Introduzione al Parameter Server

- Terza generazione del Parameter Server, introdotta nel 2014
- **Obiettivo:** gestire miliardi di parametri su cluster industriali
- Compromesso tra **efficienza di calcolo, convergenza dell'algoritmo e resilienza ai guasti**
- Casi d'uso:
  - Sparse Logistic Regression (Ad Click Prediction)
  - LDA (Topic modeling)



Architettura del Parameter Server



Discesa Distribuita del Subgradiente

Algorithm 1 Distributed Subgradient Descent

Task Scheduler:

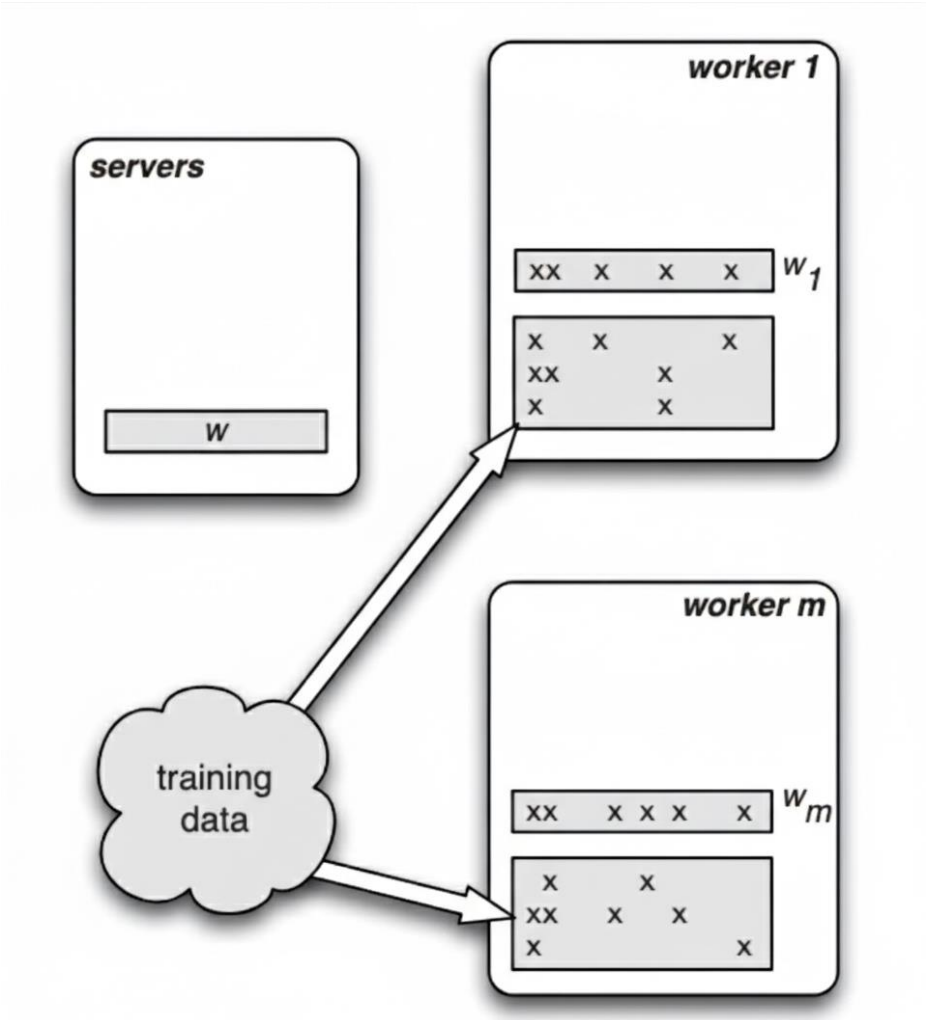
issue LoadData() to all workers
for iteration $t = 0, \ldots, T$ **do**
    issue WORKERITERATE(t) to all workers.
end for

Worker $r = 1, \ldots, m$:

function LOADDATA()
    load a part of training data $\{y_{i_k}, x_{i_k}\}_{k=1}^{n_r}$
    pull the working set $w_r^{(0)}$ from servers
end function
function WORKERITERATE(t)
    gradient $g_r^{(t)} \leftarrow \sum_{k=1}^{n_r} \partial \ell(x_{i_k}, y_{i_k}, w_r^{(t)})$
    push $g_r^{(t)}$ to servers
    pull $w_r^{(t+1)}$ from servers
end function

Servers:

function SERVERITERATE(t)
    aggregate $g^{(t)} \leftarrow \sum_{r=1}^{m} g_r^{(t)}$
    $w^{(t+1)} \leftarrow w^{(t)} - \eta \left(g^{(t)} + \partial \Omega(w^{(t)})\right)$
end function



Discesa Distribuita del Subgradiente

Algorithm 1 Distributed Subgradient Descent

Task Scheduler:

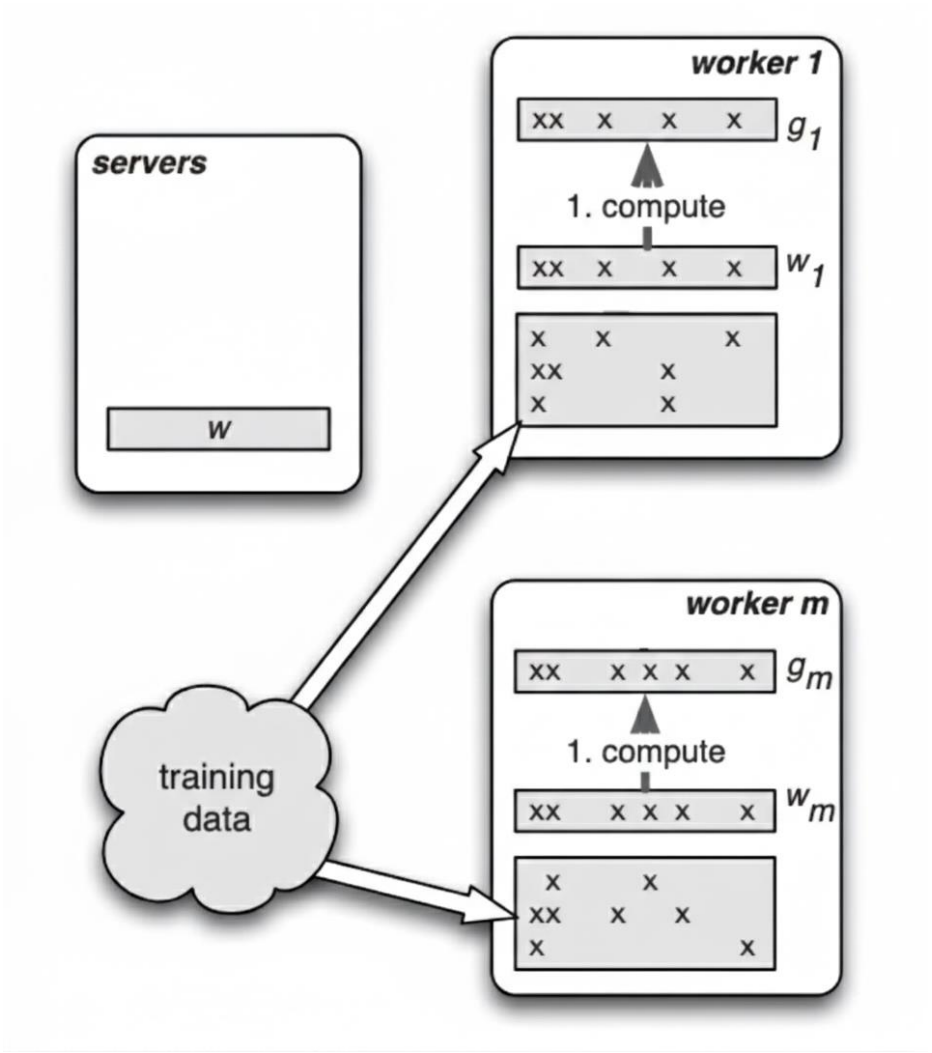
1: issue LoadData() to all workers
2: **for** iteration $t = 0, \ldots, T$ **do**
3: issue WORKERITERATE(t) to all workers.
4: **end for**

Worker $r = 1, \ldots, m$:

1: **function** LOADDATA()
2: load a part of training data $\{y_{i_k}, x_{i_k}\}_{k=1}^{n_r}$
3: pull the working set $w_r^{(0)}$ from servers
4: **end function**
5: **function** WORKERITERATE(t)
6: gradient $g_r^{(t)} \leftarrow \sum_{k=1}^{n_r} \partial \ell(x_{i_k}, y_{i_k}, w_r^{(t)})$
7: push $g_r^{(t)}$ to servers
8: pull $w_r^{(t+1)}$ from servers
9: **end function**

Servers:

1: **function** SERVERITERATE(t)
2: aggregate $g^{(t)} \leftarrow \sum_{r=1}^{m} g_r^{(t)}$
3: $w^{(t+1)} \leftarrow w^{(t)} - \eta \left(g^{(t)} + \partial \Omega(w^{(t)})\right)$
4: **end function**



Discesa Distribuita del Subgradiente

Algorithm 1 Distributed Subgradient Descent

Task Scheduler:

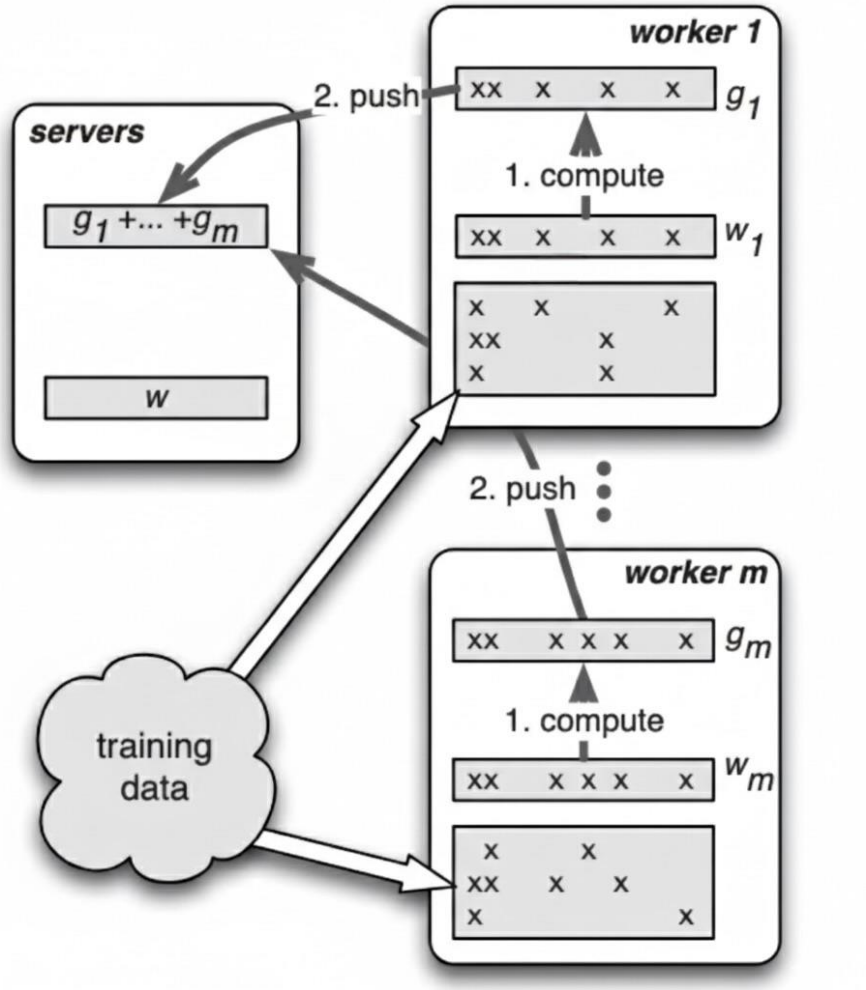
1: issue LoadData() to all workers
2: **for** iteration $t = 0, \ldots, T$ **do**
3: issue WORKERITERATE(t) to all workers.
4: **end for**

Worker $r = 1, \ldots, m$:

1: **function** LOADDATA()
2: load a part of training data $\{y_{i_k}, x_{i_k}\}_{k=1}^{n_r}$
3: pull the working set $w_r^{(0)}$ from servers
4: **end function**
5: **function** WORKERITERATE(t)
6: gradient $g_r^{(t)} \leftarrow \sum_{k=1}^{n_r} \partial \ell(x_{i_k}, y_{i_k}, w_r^{(t)})$
7: push $g_r^{(t)}$ to servers
8: pull $w_r^{(t+1)}$ from servers
9: **end function**

Servers:

1: **function** SERVERITERATE(t)
2: aggregate $g^{(t)} \leftarrow \sum_{r=1}^{m} g_r^{(t)}$
3: $w^{(t+1)} \leftarrow w^{(t)} - \eta \left(g^{(t)} + \partial \Omega(w^{(t)})\right)$
4: **end function**



Discesa Distribuita del Subgradiente

Algorithm 1 Distributed Subgradient Descent

Task Scheduler:

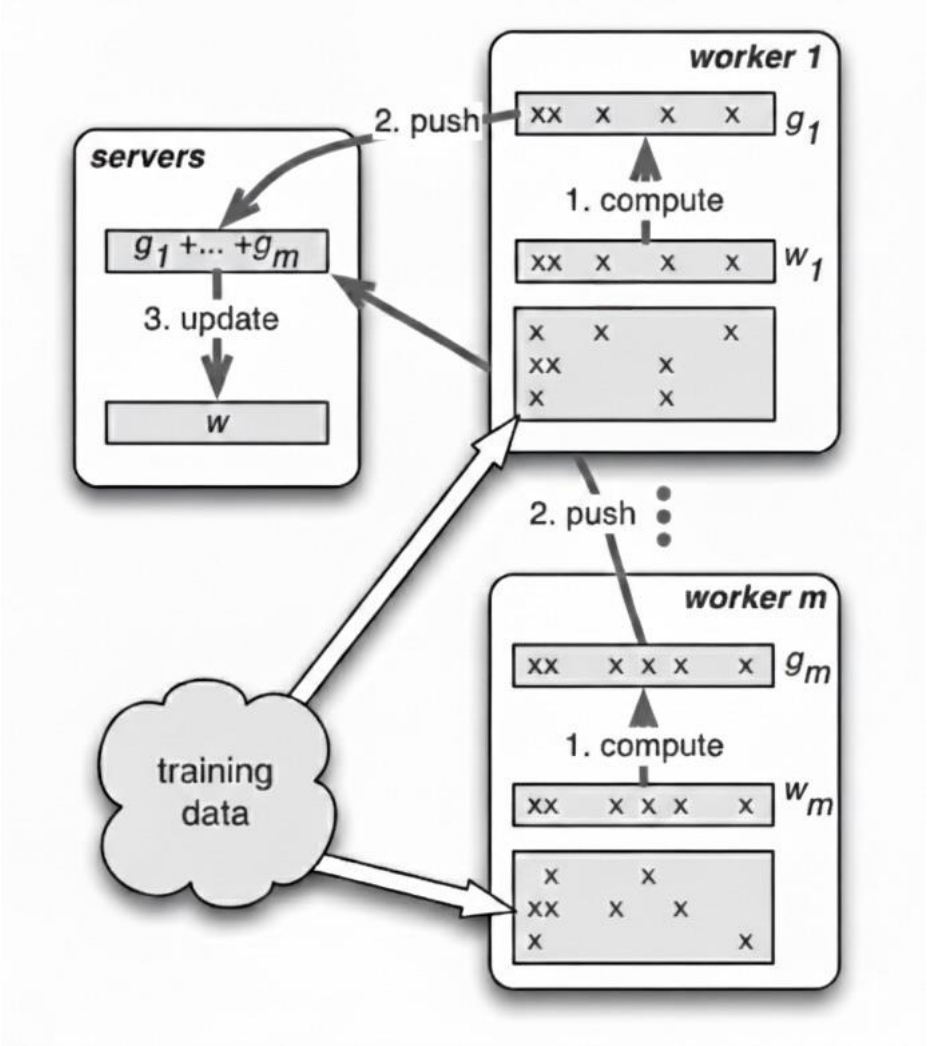
1: issue LoadData() to all workers
2: **for** iteration $t = 0, \ldots, T$ **do**
3: issue WORKERITERATE(t) to all workers.
4: **end for**

Worker $r = 1, \ldots, m$:

1: **function** LOADDATA()
2: load a part of training data $\{y_{i_k}, x_{i_k}\}_{k=1}^{n_r}$
3: pull the working set $w_r^{(0)}$ from servers
4: **end function**
5: **function** WORKERITERATE(t)
6: gradient $g_r^{(t)} \leftarrow \sum_{k=1}^{n_r} \partial \ell(x_{i_k}, y_{i_k}, w_r^{(t)})$
7: push $g_r^{(t)}$ to servers
8: pull $w_r^{(t+1)}$ from servers
9: **end function**

Servers:

1: **function** SERVERITERATE(t)
2: aggregate $g^{(t)} \leftarrow \sum_{r=1}^{m} g_r^{(t)}$
3: $w^{(t+1)} \leftarrow w^{(t)} - \eta \left(g^{(t)} + \partial \Omega(w^{(t)})\right)$
4: **end function**



Discesa Distribuita del Subgradiente

Algorithm 1 Distributed Subgradient Descent

Task Scheduler:

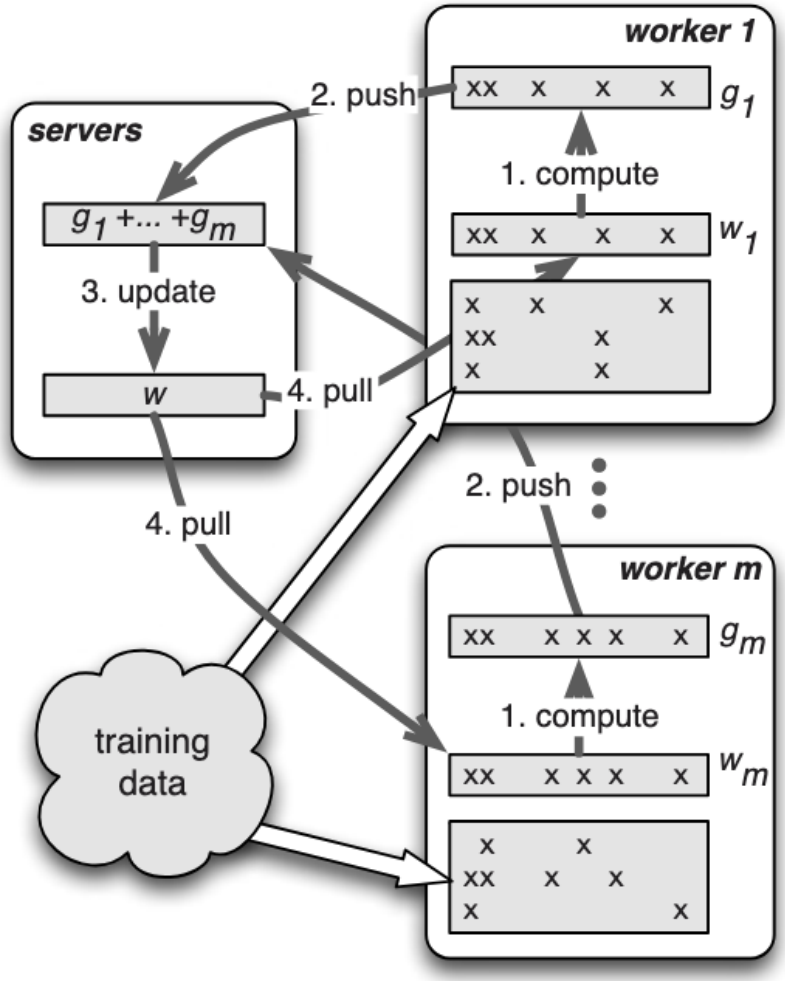
issue LoadData() to all workers
for iteration $t = 0, \ldots, T$ **do**
    issue WORKERITERATE(t) to all workers.
end for

Worker $r = 1, \ldots, m$:

function LOADDATA()
    load a part of training data $\{y_{i_k}, x_{i_k}\}_{k=1}^{n_r}$
    pull the working set $w_r^{(0)}$ from servers
end function
function WORKERITERATE(t)
    gradient $g_r^{(t)} \leftarrow \sum_{k=1}^{n_r} \partial \ell(x_{i_k}, y_{i_k}, w_r^{(t)})$
    push $g_r^{(t)}$ to servers
    pull $w_r^{(t+1)}$ from servers
end function

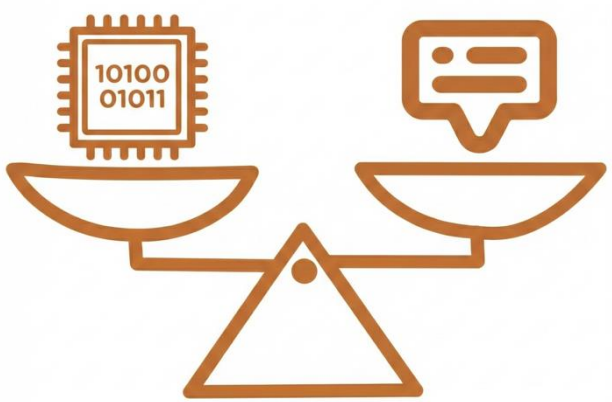
Servers:

function SERVERITERATE(t)
    aggregate $g^{(t)} \leftarrow \sum_{r=1}^{m} g_r^{(t)}$
    $w^{(t+1)} \leftarrow w^{(t)} - \eta \left(g^{(t)} + \partial \Omega(w^{(t)})\right)$
end function



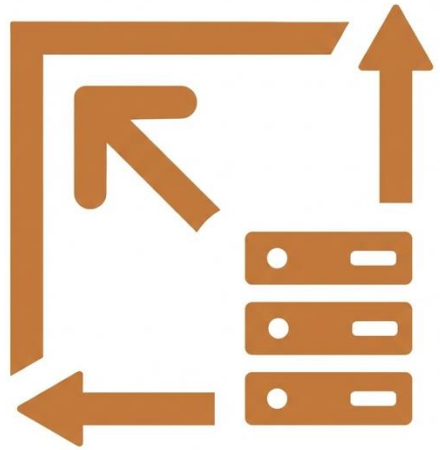
Le Sfide dello Scaling

Efficienza di Comunicazione



Computazione vs. Comunicazione

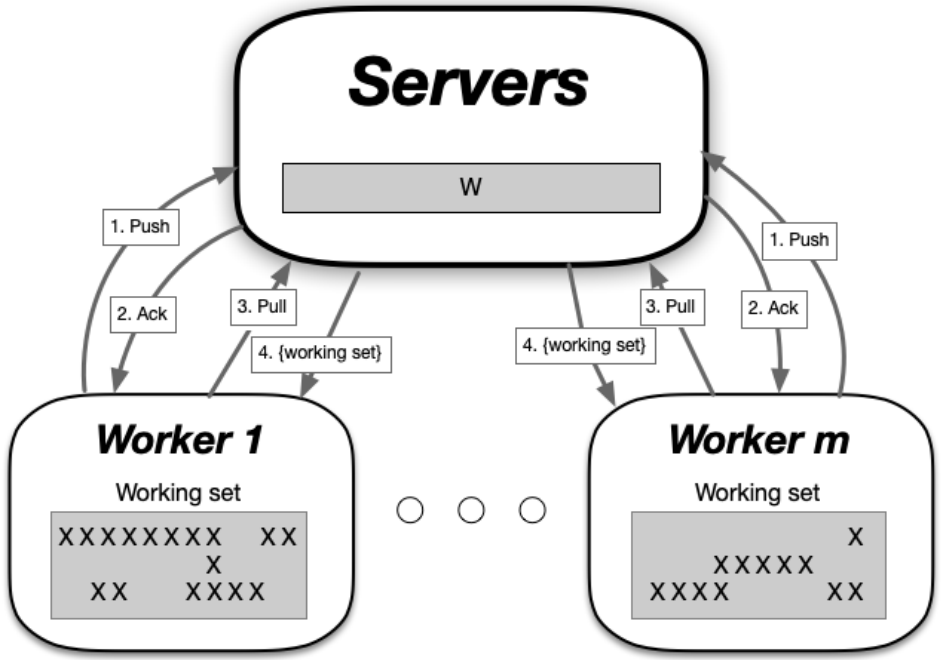
Tolleranza ai Guasti



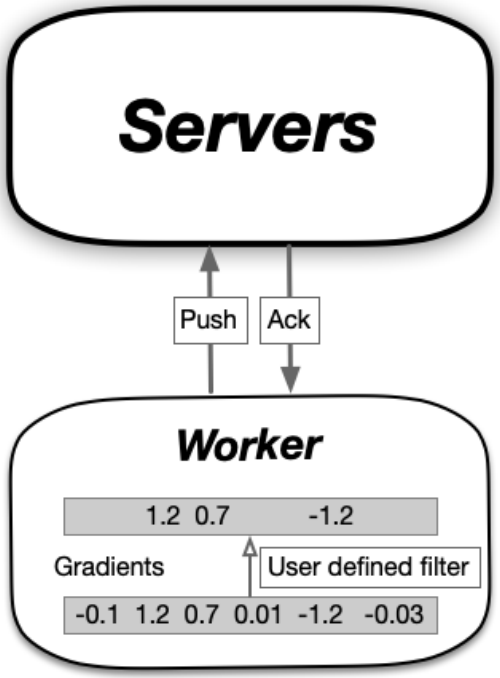
Scalabilità Dinamica

Comunicazione Efficiente

- Parametri del modello memorizzati come coppie **chiave – valore**
- Tra le tecniche implementate per ottimizzare la banda di rete troviamo:
  - **Range Push & Pull**

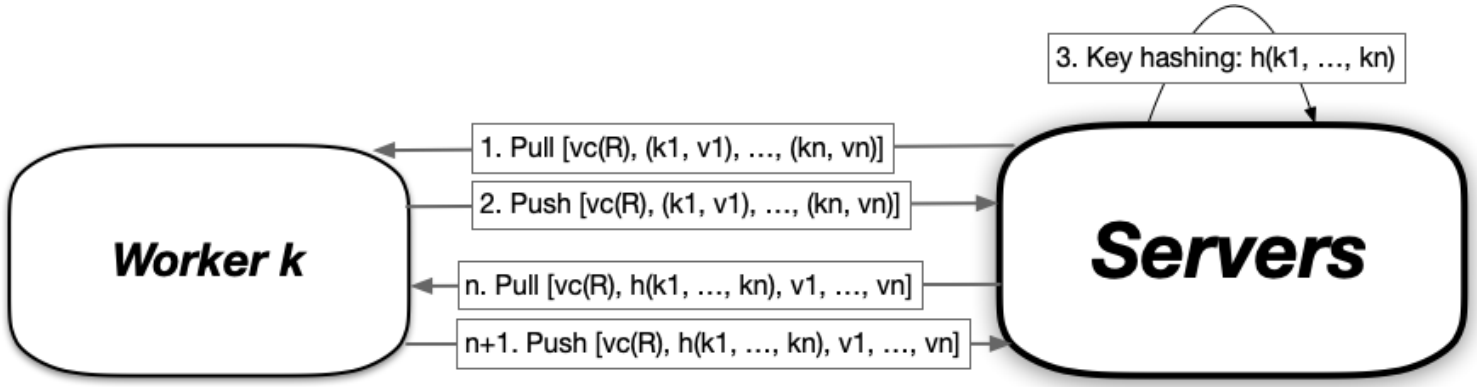


  - **Filtri definiti dall'utente**

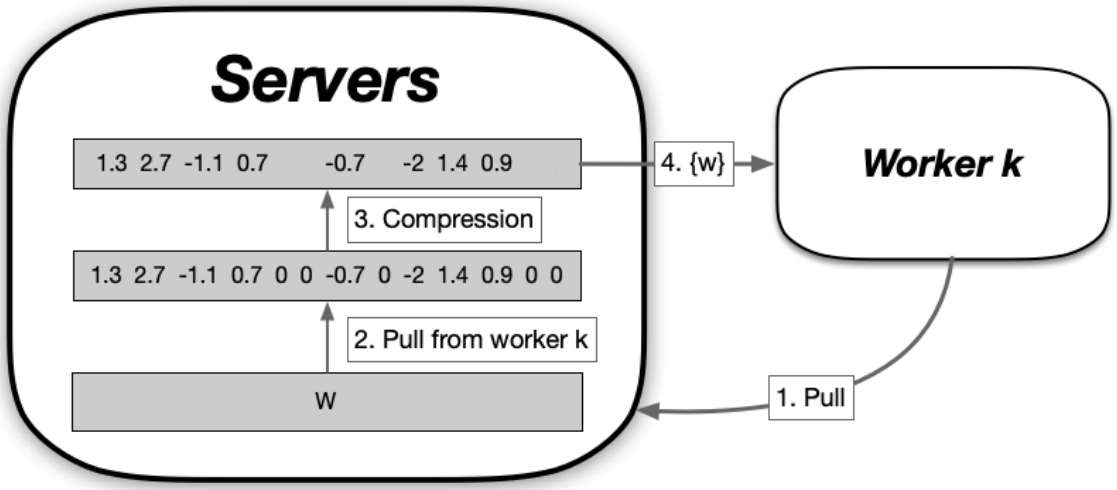


Comunicazione Efficiente

- **Cache delle chiavi:**

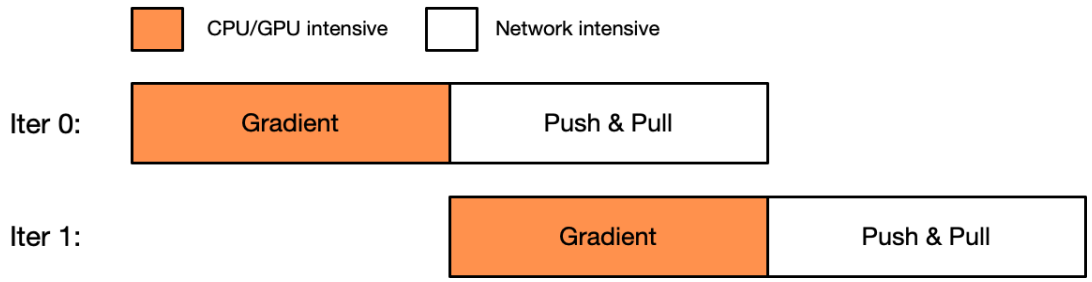


- **Compressione:**

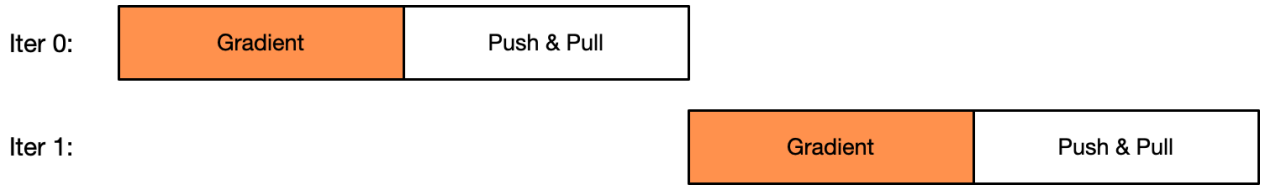


Computazione vs. Comunicazione

- **Task:** operazioni definite dall'utente / calcolo del gradiente / push / pull
- I task vengono eseguiti in modo asincrono

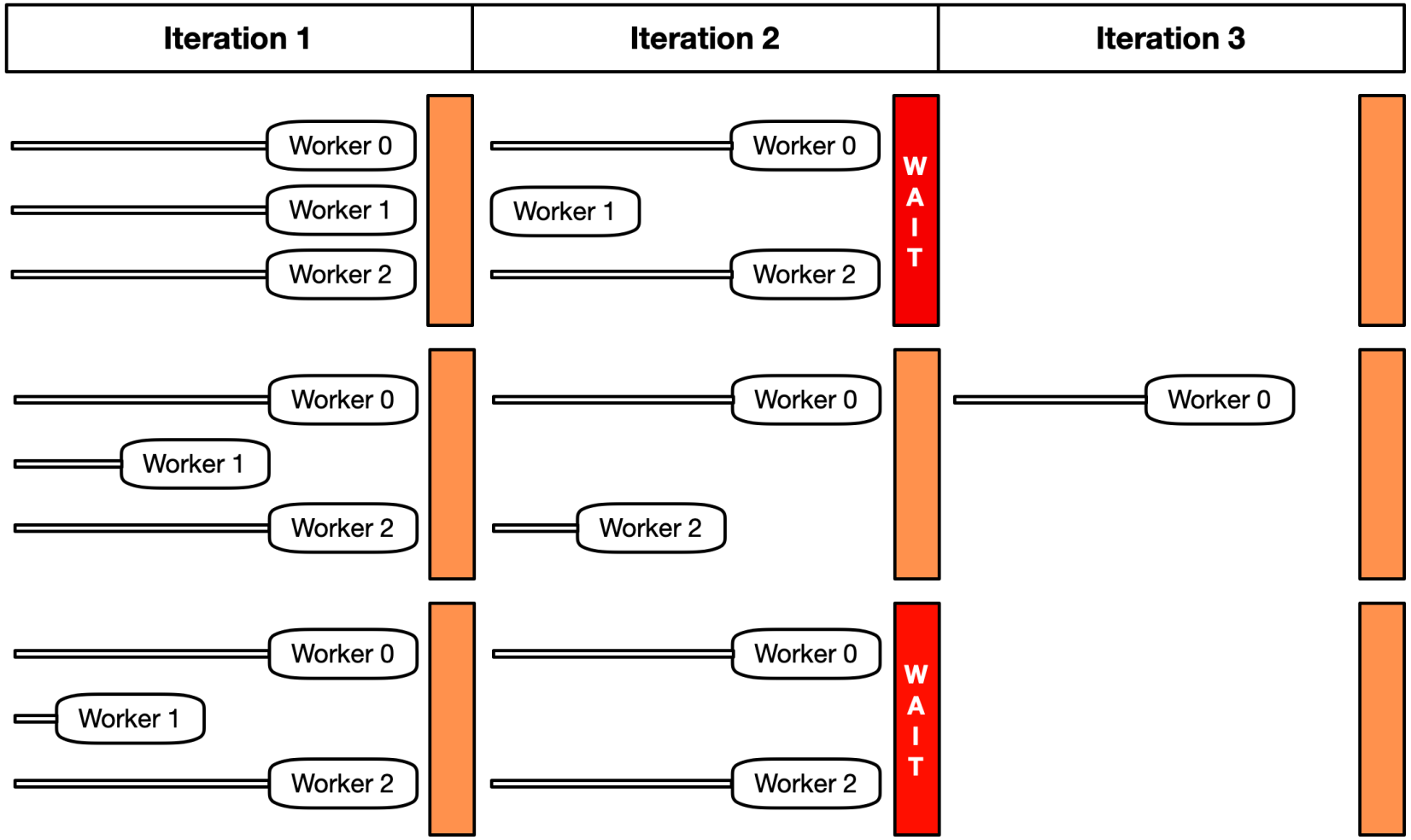


- È comunque possibile definire delle **dipendenze** tra task



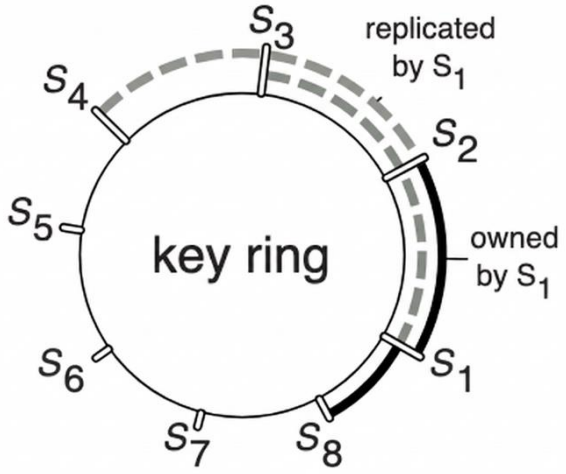
Computazione vs. Comunicazione

- **Sincrono:**
- **Asincrono:**
- **Ritardo limitato:**



Tolleranza ai Guasti

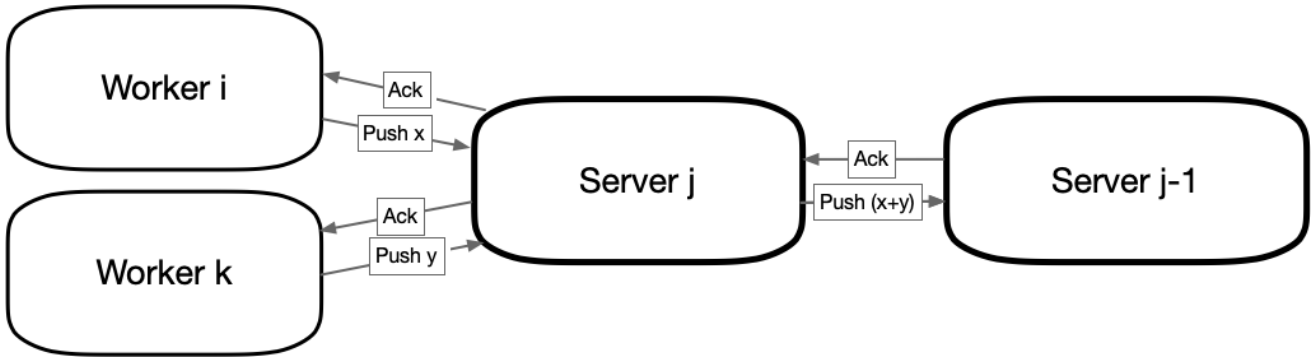
- **Consistent Hashing:** parametri e Server sono mappati su un anello



- **Chain Replication:** parametri replicati su k Server

Tolleranza ai Guasti

- **Pre-Replication Aggregation:** il Server replica solo il risultato finale dell'aggiornamento



- Utilizzati i **Range Vector Clock** per tracciare la "storia" di un dato
- **Risultati:** il recupero dai guasti < 1s senza interrompere il training

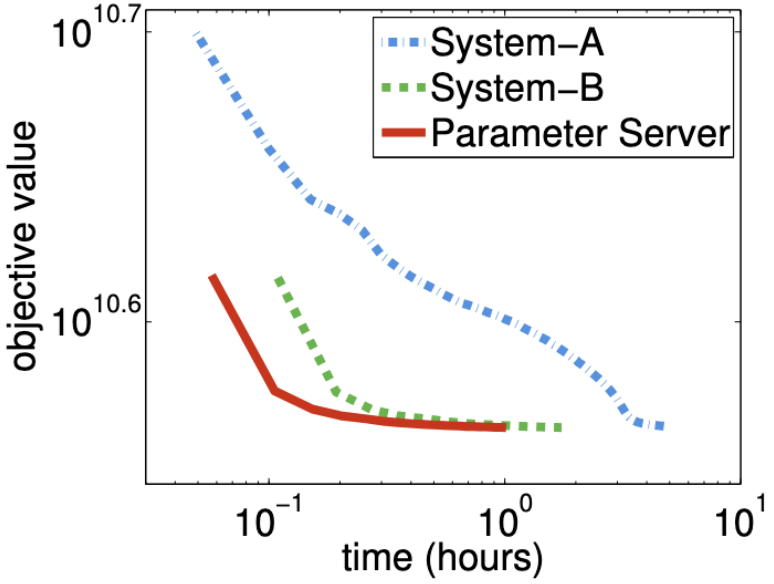
Scalabilità Dinamica

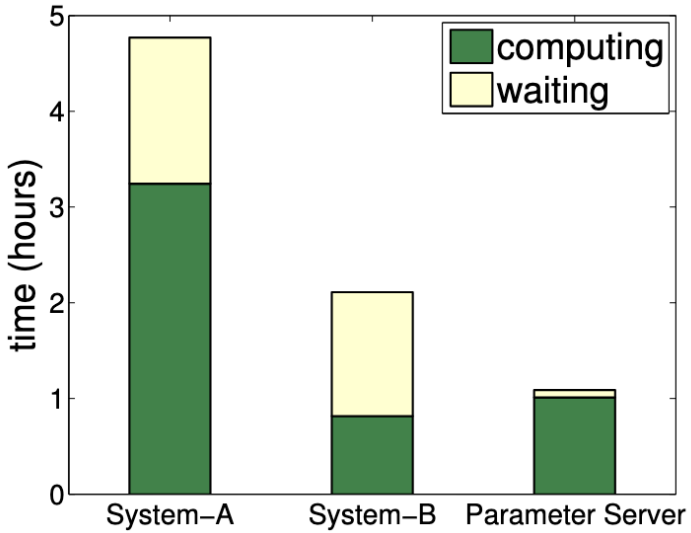
- **Nodo Server:** operazione critica perché i Server gestiscono lo stato. Il trasferimento avviene in due fasi:
  - Pre-copia dei parametri
  - Sincronizzazione finale degli aggiornamenti avvenuti durante la copia
- **Nodo Worker:** più semplici da gestire
  - Aggiunta: il Worker deve solo ricevere i dati dal Task Scheduler
  - Recupero: opzionale

Esperimenti: Sparse Logistic Regression

- **Problema:** Ad Click Prediction (170 mld di esempi, 636 TB)
- **Algoritmo:** Block PG con consistenza limit

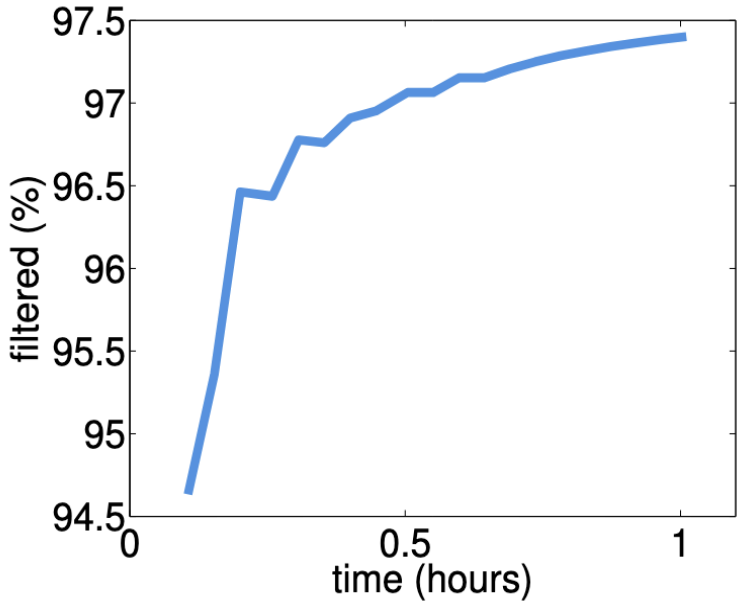
	Method	Consistency	LOC
System A	L-BFGS	Sequential	10,000
System B	Block PG	Sequential	30,000
Parameter Server	Block PG	Bounded Delay KKT Filter	300

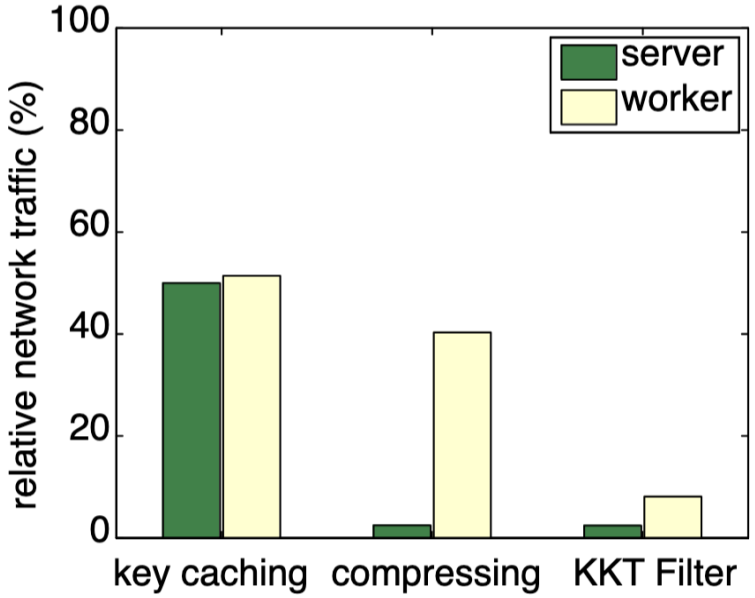


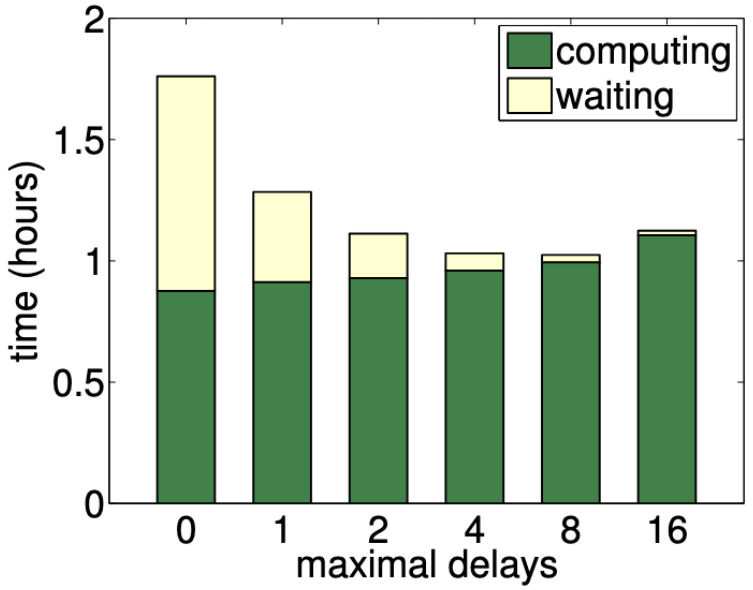


Esperimenti: Sparse Logistic Regression

- Altri risultati ottenuti implementando le tecniche elencate:

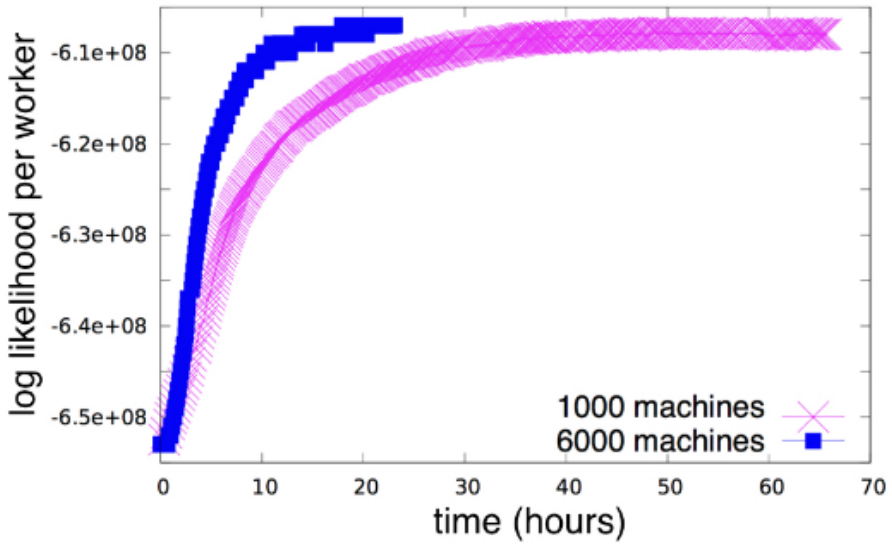


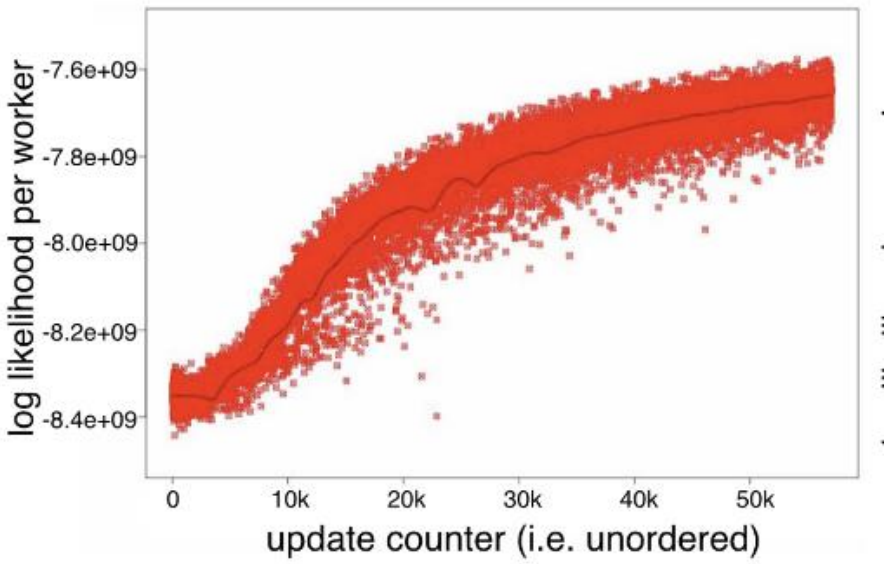




Esperimenti: Latent Dirichlet Allocation

- **Problema:** Topic Modeling di 5 miliardi di utenti analizzando le URL che hanno cliccato
- **Algoritmo:** LDA asincrono



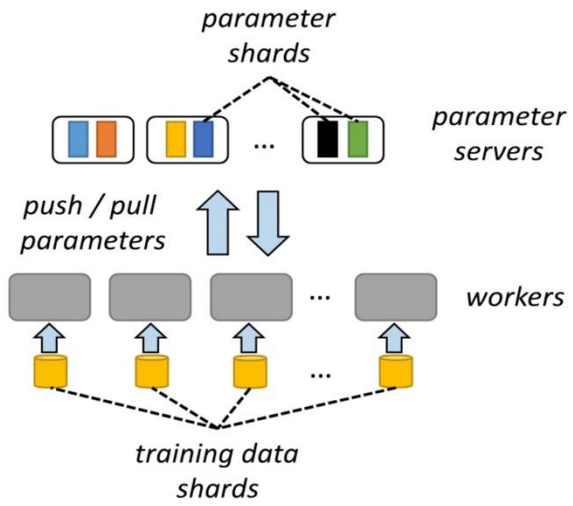


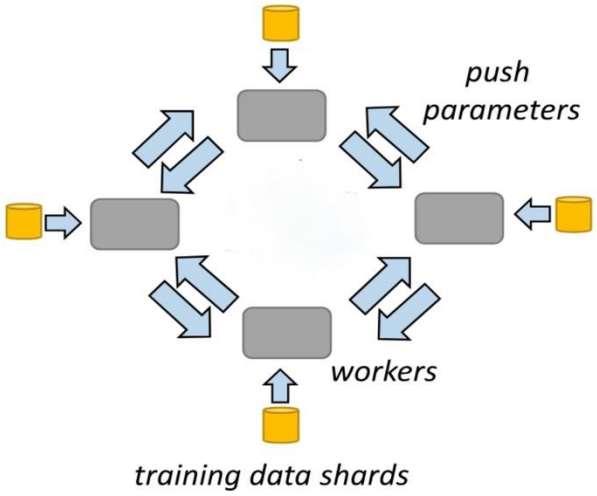
Parameter Server in Pratica

- **TensorFlow:** evoluzione flessibile basata su dataflow graph. Permette la scelta tra approccio sincrono e asincrono
- **PyTorch:** framework versatile che abilita il training distribuito e comunicazioni RPC flessibili per architetture Parameter Server
- **Apache SINGA:** supporta modelli di addestramento sincroni, asincroni e ibridi. Offre flessibilità nel parallelismo di dati o del modello
- **Monolith:** sistema di ByteDance ottimizzato per recommendation systems su larga scala; utilizza il Parameter Server per gestire embedding table con miliardi di parametri

Parameter Server vs. AllReduce

Caratteristica	Parameter Server	AllReduce
Architettura	Centralizzata (sharded)	Decentralizzata (peer to peer)
Punti di forza	Modelli massivi, aggiornamenti sparsi, tolleranza ai guasti, sincronia flessibile, sistemi eterogenei	Efficienza estrema per modelli densi su reti ultra-veloci, sistemi omogenei
Punti deboli	Possibile collo di bottiglia dovuto ai server	Problema dello straggler (sincrono)





Conclusioni

- Il Parameter Server è la scelta ideale per modelli massivi con miliardi di parametri e aggiornamenti sparsi
- Garantisce che il training non si fermi mai, gestendo **hardware eterogeneo** e **guasti continui**
- Ottimizza il trade-off tra **efficienza di calcolo** e **robustezza** per gestire carichi su scala industriale

Bibliografia

- Li, Andersen, Park, Smola, Ahmed, Josifovski, Long, Shekita, Su. Scaling Distributed Machine Learning with the Parameter Server. *Proceedings of OSDI*. 2014
- Verbraeken, Wolting, Katzy, Kloppenburg, Verbelen, Rellermeyer. A Survey on Distributed Machine Learning. *ACM Computing Surveys*. 2020
- Li, Andersen, Smola, Yu. Communication Efficient Distributed Machine Learning with the Parameter Server. *Proceedings of NIPS*. 2014
- Mayer, Jacobsen. Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques, and Tools. *ACM Computing Surveys*. 2020

Bibliografia

- Dean, Corrado, Monga, Chen, Devin, Le, Mao, Ranzato, Senior, Tucker, Yang, Ng. Large Scale Distributed Deep Networks. *Proceedings of NIPS*. 2012
- Gu, Niu, Wu, Chen, Hu, Lyu, Wu. From Server-Based to Client-Based Machine Learning: A Comprehensive Survey. *ACM Computing Surveys*. 2020
- Course CS4787/5777: Principles of Large-Scale Machine Learning — Fall 2025

elicsir
Elevating Italian Computer Science Innovation and Research

Grazie per l’attenzione

Appendice: Sketch di Dimostrazione Block PG

Algorithm 2 Delayed Block Proximal Gradient Method Solving (1)

Scheduler:

1: Partition parameters into k blocks $b_1, \ldots, b_k$
2: **for** $t = 1$ **to** T**:** Pick a block b_{i_t} and issue the task to workers

Worker r at iteration t

1: Wait until all iterations before $t - \tau$ are finished
2: Compute first-order gradient $g_r^{(t)}$ and coordinate-specific learning rates $u_r^{(t)}$ on block b_{i_t}
3: Push $g_r^{(t)}$ and $u_r^{(t)}$ to servers with user-defined filters, e.g., the random skip or the KKT filter
4: Pull $w_r^{(t+1)}$ from servers with user-defined filters, e.g., the significantly modified filter

Servers at iteration t

1: Aggregate $g^{(t)}$ and $u^{(t)}$
2: Solve the generalized proximal operator (2) $w^{(t+1)} \leftarrow \text{Prox}_{\gamma_t}^{U}(w^{(t)})$ with $U = \text{diag}(u^{(t)})$.

Appendice: Sketch di Dimostrazione Block PG

- **Assunzione:** la funzione è Lipschitz Continua a blocchi, ovvero la variazione del gradiente è limitata per ogni blocco di parametri. In particolare i limiti sono i seguenti:
  - $\mathbf{L_{var}}$: variazione interna al blocco aggiornato
  - $\mathbf{L_{cov}}$: interferenza tra blocchi diversi
- **Teorema:** l'algoritmo converge a un **punto stazionario** anche con ritardi nella comunicazione (τ) a condizione che il learning rate sia:

$$\gamma_t \leq \frac{M_t}{L_{var} + \tau L_{cov} + \epsilon}$$