Corso di **Tecnologie per il cloud**

4. Membership e failure detection

Introduzione

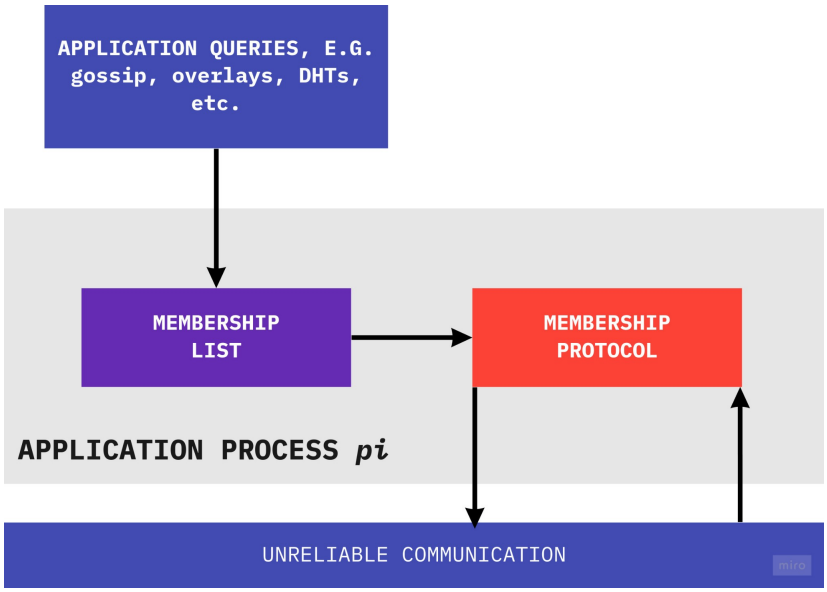
I concetti di **membership** (appartenenza ad un gruppo) sono molto richiesti nell'ambito dei data center e degli ambienti cloud. In tali sistemi, i fallimenti e i guasti sono all'ordine del giorno, molto più di quanto si possa pensare ed è quindi nostro compito studiare un sistema **failure detector** che rilevi eventuali guasti al data center. I guasti sono la norma in un data center, si pensi che in uno con circa 12000 server, il **MTBF (Mean Time Before Failure)** è di circa 7,2 ore. Ciò significa che si presenta un malfunzionamento almeno ogni 7,2 ore. Sarebbe anormale se all'improvviso si avesse una rete di calcolatori senza fallimenti.

In assenza di un buon algoritmo di failure detection, alcuni dati potrebbero andare persi e ciò causerebbe **inconsistenza** nella distribuzione dei dati ai client.

Con **processo** intenderemo un server, con **gruppo di processi** o **membri** intenderemo quindi un gruppo di server in un data center.

Membership list

In un gruppo di processi, ciascuno di essi ha una **membership list**, la quale contiene la lista degli altri processi del gruppo che sono ancora in funzione. Tale lista può essere consultata dalle query di svariate applicazioni, ad esempio quelle gossip-based.

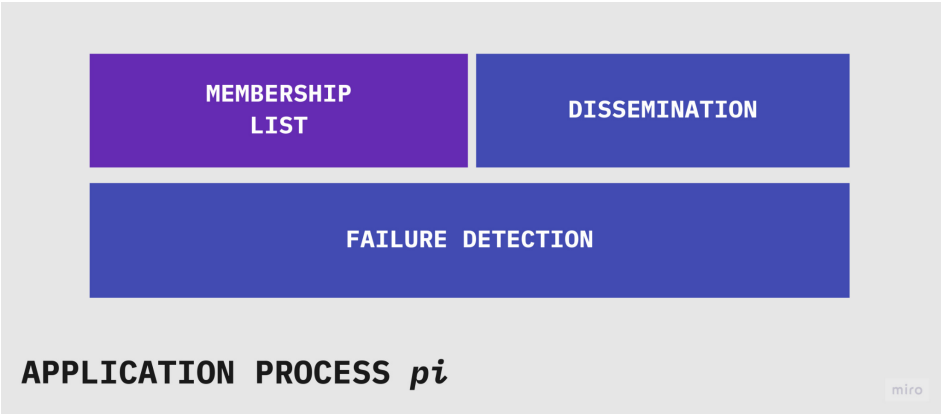


IVAN SOLLAZZO - ANDREA PISCOPO - NICOLAS BRUNO
PROFESSORE PIERLUIGI GALLO - INGEGNERIA CIBERNETICA
UNIVERSITA' DEGLI STUDI DI PALERMO

Un buon algoritmo di membership fa in modo che questa membership list sia sempre aggiornata, sulla base di quali processi sono entrati a far parte del gruppo e quali sono usciti per guasto o per espressa volontà.

- Un processo **guasto** uscirà dal gruppo di processi senza annunciare nulla agli altri;
- Un processo che manifesta la **volontà** di uscire, lo annuncerà anche agli altri processi.

Non è detto che un algoritmo sfrutti per intero il concetto di membership list: ce ne sono alcuni, come quelli **gossip-based**, che si appoggiano su una membership list all'incirca completa, altri come **Cyclon**, usano una lista parziale. La complessità della membership list determina anche la consistenza del sistema, senza ombra di dubbio gli algoritmi gossip-based sono molto più consistenti di Cyclon.

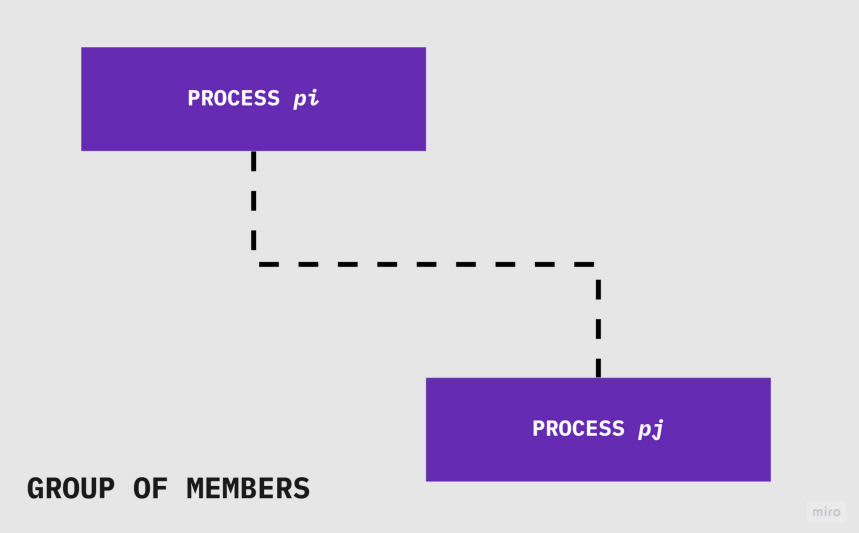


Approfondimento ricevimento

Per quanto riguarda gli algoritmi ad: Heartbeat Centralizzato - Heartbeat ad anello - Heartbeating all.to.all - Hearbeating gossip-style, ciascuno di essi dispone di una membership list, l'unica differenza tra gli algoritmi è la diversa tipologia di comunicazione tra i nodi

Dissemination

La disseminazione di informazioni può essere utilizzata per supportare la rilevazione dei guasti e garantire la coerenza dei dati nella rete. La rilevazione dei guasti, a sua volta, può essere utilizzata per attivare processi di ripristino e di sostituzione dei nodi guasti per garantire la continuità e la robustezza della rete. Sostanzialmente, quando un processo ***pj*** smette di funzionare, deve esservi un altro qualunque processo della rete ***pi*** che si accorga di tale malfunzionamento e che lo diffonda a tutti gli altri processi utilizzando il meccanismo di disseminazione.



Failure detection - Proprietà

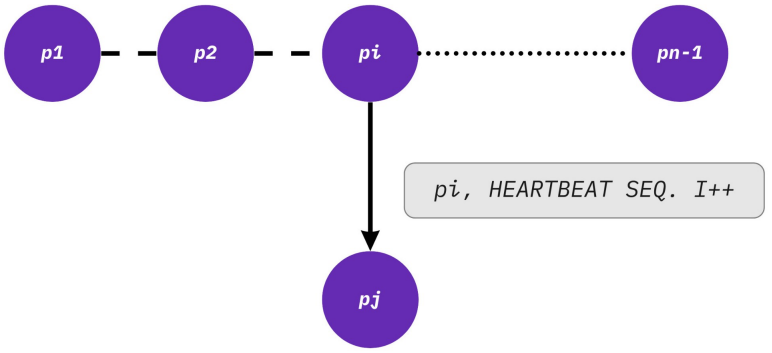
Nei meccanismi di failure detection distribuiti ci sono 4 proprietà principali alle quali si mira:

- **Scalabilità:** il carico deve essere equo per ciascun membro del gruppo.
- **Completezza:** viene rilevato qualsiasi fallimento;
- **Accuratezza:** non ci siano falsi positivi.
- **Velocità:** il tempo richiesto per rilevare il primo fallimento del processo ***pj***;


S.C.A.V.

Ottenere le caratteristiche di completezza e accuratezza è letteralmente **impossibile** in quelle reti dove si verificano perdite di pacchetti. Nel caso reale, si adoperano quindi dei meccanismi di **garanzia** per quanto concerne la completezza e **probabilistici** per quanto riguarda l'accuratezza. Inoltre, può capitare che più processi smettano di funzionare nello stesso momento, quindi è necessario che l'algoritmo li rilevi **tutti** contemporaneamente.

Heartbeat centralizzato

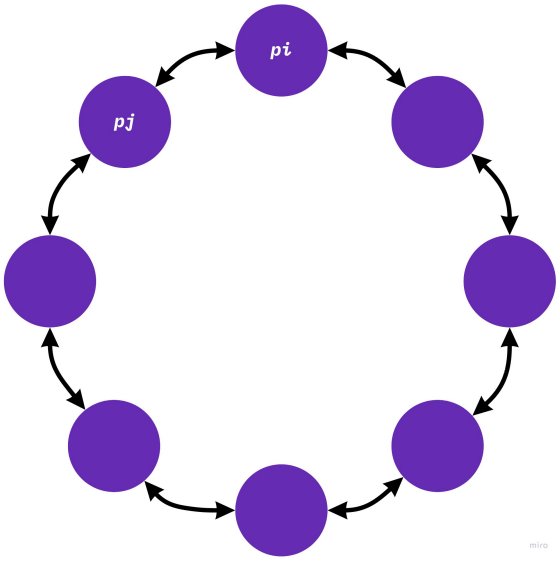


Si tratta di un primo approccio molto semplice che vede una serie di ***pn - 1*** processi che inviano dei messaggi, cosiddetti **heartbeat**, ad un processo ***pj***. Ciascun heartbeat contiene un numero sequenziale che viene incrementato localmente ogni volta che il processo ***pi*** lo invia a ***pj***.

Se ad un certo punto ***pj*** si accorge che ***pi*** non invia più heartbeat entro un certo **timeout**, allora significa che il processo è crashato, quindi ***pj*** lo marcherà come tale.

Il problema fondamentale di questo approccio è che, se crasha ***pj***, il meccanismo di failure detection smette di funzionare, per cui l'ideale sarebbe decentralizzarlo.

Heartbeating ad anello

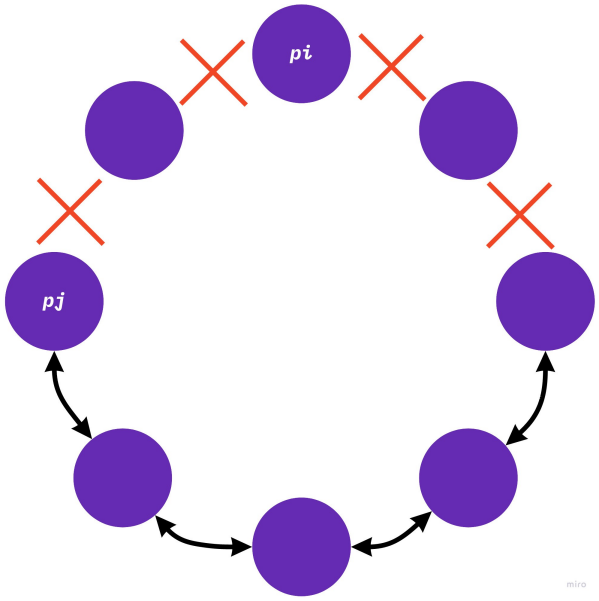


Anziché centralizzare il rilevamento di heartbeat su un unico processo, si potrebbe pensare di:

1. Organizzare il gruppo di processi in una sorta di **anello**;
2. Far girare il meccanismo di rilevamento degli heartbeat su ciascuno di essi.

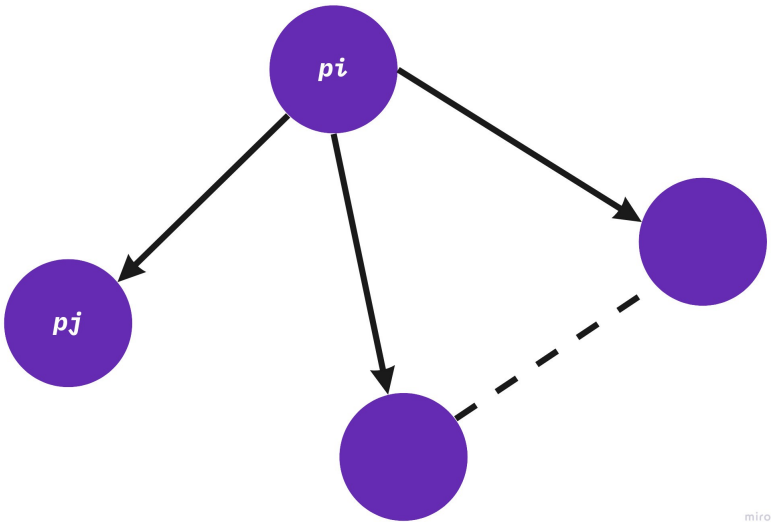
Ciascun processo invierà gli heartbeat ad almeno **uno** dei suoi vicini, senza influire sulla qualità del sistema di heartbeating (sempre sequenziale). Quando ***pj*** non riceverà più heartbeat da ***pi***, allora significa che quest'ultimo ha smesso di funzionare e sarà marcato così da ***pj***.

Questo meccanismo è superiore a quello centralizzato, ma nel caso in cui dovessero sussistere più crash in contemporanea, c'è il rischio che alcuni di questi non vengano rilevati correttamente.



Immaginiamo una situazione in cui alcuni processi adiacenti a pi falliscono, causando la chiusura delle comunicazioni bidirezionali tra i processi e pi. Ciò avrebbe come conseguenza l'isolamento di pi, che non sarebbe più in grado di comunicare con il resto del ring.

Heartbeating all-to-all

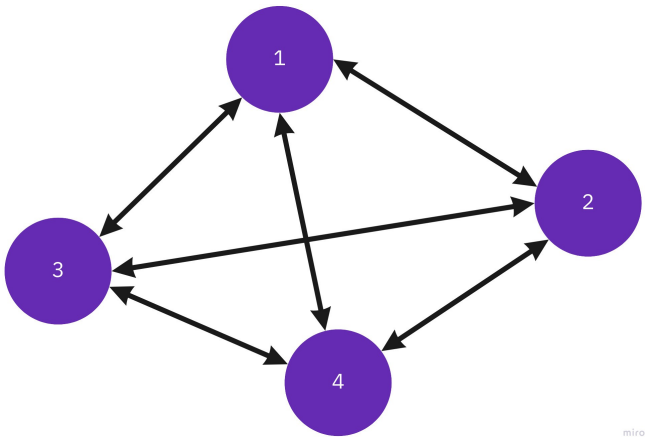


Ciascun processo ***pi*** invia gli heartbeat, non a un singolo, ma a **tutti** i processi che fanno parte del gruppo. Analogamente a prima, il primo processo che si accorge che ***pi*** non invia più heartbeat, lo marca come fallito. Anche qui avviene un bilanciamento del carico.

Il problema principale in questo tipo di heartbeating è che in presenza di un processo ***pj*** lento, il quale riceve pacchetti con un tempo maggiore degli altri, potrebbero presentarsi falsi positivi di processi crashati che in realtà non lo sono.

Heartbeating gossip-style

L'heartbeating gossip-style è nettamente più robusto dell'all-to-all, e analogamente a prima, ciascun processo manda degli heartbeat agli altri che fanno parte del gruppo, incrementando ogni volta un numero di sequenza locale. I processi che non ricevono gli heartbeat entro un certo timeout, marcano al sorgente come **failed.**



Nel dettaglio, si hanno ***n*** processi, ciascuno dotato di una **membership list** costituita da tre campi: **indirizzo di ciascun processo del gruppo** (sostanzialmente l'ID), l'**hearbeat counter** e il **tempo locale dall'ultimo aggiornamento dell'heartbeat counter** (varia per ciascun processo e non fa testo, dato che i nodi potrebbero non essere sincronizzati).

Esempio: tabella processo 1

ID	Hearbeat counter	Tempo
1	10120	66
2	**10103**	**62**
3	10098	63
4	10111	65

Ad esempio, il processo 2 ha inviato al processo 1 l'ultimo heartbeat quando il tempo locale di quest'ultimo era a 62.

Periodicamente, ciascun processo invia la sua tabella ai suoi vicini. Quando essi la ricevono, effettuano un confronto tra la tabella ricevuta e quella locale:

- Se l'heartbeat **count** associato a ciascun processo nella tabella ricevuta è **inferiore** a quello locale, non viene aggiornato localmente;

- Se l'heartbeat **count** associato a ciascun processo nella tabella ricevuta è **superiore** a quello locale, viene aggiornato anche localmente e aggiornato il tempo locale **corrente**.

ATTENZIONE!!! NON SI PARLA DI TEMPI MA DI COUNT DATO CHE GLI ORARI DEI PROCESSI SONO ASINCRONI

Il timeout funziona analogamente all'all-to-all heartbeating, ed è basato sul tempo locale del processo. Inoltre:

- Se il contatore di heartbeat di un processo non viene incrementato per un tempo superiore a T_{fail} secondi, il processo è considerato **failed**;
- Dopo un tempo superiore a $T_{cleanup}$, il processo viene eliminato dalla membership list.

Perché la presenza di due tempi? Si consideri il seguente scenario, supponendo che $T_{cleanup}$ non ci sia e che il processo venga quindi eliminato dalla lista subito dopo T_{fail}.

1. Il processo ***p2*** si rende conto che ***p3*** è crashato e lo rimuove dalla lista;
2. Per il principio del gossip, ***p1***, che ancora non ha raggiunto T_{fail}, sceglie ***p2*** nel suo fan out e gli invia la sua tabella, tuttavia contenente ancora ***p3*** non aggiornato (ancora in vita);
3. In tale caso ***p2*** prenderebbe tale informazione come nuova, penserebbe che ***p3*** sia rientrato nel gruppo e lo riaggiungerebbe alla lista, causando una situazione in cui ***p3*** non verrebbe mai rimosso.

La presenza di $T_{cleanup}$ da il tempo a tutti i processi di raggiungere T_{fail} nel loro orologio locale e di eventualmente **ignorare** eventuali membership tables vecchie. Ricollegandoci allo scenario di prima, con $T_{cleanup}$ presente, ***p2*** ignorerà qualunque elemento delle tabelle ricevute relativo al processo considerato crashato infatti dopo il T_{fail} le informazioni del nodo in down non verranno più inoltrate ad altri. Tipicamente, i tempi di fail e cleanup coincidono (considerando $T_{cleanup}$ "avviato" al termine di T_{fail}).

$$T_{fail} = T_{cleanup}$$

Approfondimento sul periodo di gossip

Per il principio del gossip, un singolo heartbeat impiega un tempo T_{gossip} pari a $O(log_2(N))$ per propagarsi nel gruppo di processi. Un numero ***n*** di heartbeat:

- Impiega un tempo pari a $O(log_2(N))$ se la banda di ciascun nodo è nell'ordine di $O(N)$;
- Impiega un tempo pari a $O(Nlog_2(N))$ se la banda di ciascun nodo è nell'ordine di $O(1)$.

Maggiore è la banda a disposizione, più il tempo di gossip diminuisce. Ma cosa succede in tal caso? Diminuisce chiaramente anche il tempo T_{fail}. Viceversa, aumenta se diminuisce la banda a disposizione..

False positive rate

Si definisce $P_{mistake}$ il cosiddetto **false positive rate**, che influisce sui tempi T_{fail} e $T_{cleanup}$.

- Se T_{fail} e $T_{cleanup}$ **diminuiscono**,$P_{mistake}$ aumenta, in quanto c'è minore accuratezza e maggiore probabilità di incorrere in falsi positivi;
- Se T_{fail} e $T_{cleanup}$ **aumentano**,$P_{mistake}$ diminuisce, in quanto c'è maggiore accuratezza e minore probabilità di incorrere in falsi positivi.

Approfondimento ricevimento

Possiamo paragonare l'heartbeating a due persone **(A e B)** che parlano e si chiedono notizie a vicenda di una terza persona **(C)**; quando ci si scambia informazioni non si può fare affidamento ai propri orologi perchè **non sono sincronizzati** e non segnano esattamente lo stesso orario, quindi si fa riferimento ad un **heartbeat counter** ovvero un contatore che viene incrementato ogni volta che **qualcuno** invia un heartbeat.

Il nodo **A** invia l'ultimo heartbeat locale del nodo **C** al nodo **B** e il nodo **B** invia l'ultimo heartbeat locale di **C** ad **A**.

Se il nodo **A** si rende conto di avere un heartbeat counter di **C** con valore inferiore a quello di **B** aggiorna il counter locale che fa riferimento a **C** con il valore che ha mandato **B.**

Supponiamo che il nodo **C** fallisca, osserviamo le dinamiche del nodo **B**:

- dato che il nodo C non è più online non invia più heartbeat,
- il timer $\mathbf{T_{fail}}$ del nodo B non viene azzerato
- nell'intervallo di tempo che intercorre dall'ultimo heartbeat ricevuto fino allo scadere del $\mathbf{T_{fail}}$ se si riceve un aggiornamento con valore di heartbeat superiore al valore locale da parte di un altro nodo, oppure C manda un nuovo heartbeat, il nodo B aggiornerà la sua memebrship list e il $\mathbf{T_{fail}}$ verrà azzerato
- Se il timer di $\mathbf{T_{fail}}$ non viene azzerato (e quindi non si riceve un'heartbeat più recente di quello locale), viene avviato il $\mathbf{T_{cleanup}}$ in questa lasso di tempo il nodo B **non diffonde heartbeat riguardanti quel nodo e neanche ne accetta da altri**, dopo il $\mathbf{T_{cleanup}}$ se il nodo **C** non ha inviato heartbeat, B elimina il nodo dalla membership list.

Se dopo aver eliminato il nodo dalla membership list al nodo **B** arriva l'heartbeat da parte di altri nodi che il nodo **C** è online (anche se in realtà ancora è offline) **B** aggiunge nuovamente il nodo salvando però un'informazione errata. Per evitare ciò si impostano le soglie di $\mathbf{T_{fail}}$ e $\mathbf{T_{cleanup}}$ in modo tale che il fallimento del nodo **C** si diffonda su tutto il diametro della rete e scongiurare falsi positivi.

Condizioni ottimali per un failure detector

Il failure detector **ottimale** è quello che abbia quindi il miglior compromesso tra:

- **Scalabilità:** associamo il prodotto $N * L$ (numero nodi per carico, cioè numero nodi per numero di messaggi inviati).
- **Completezza:** ci si basa sulle garanzie;
- **Accuratezza:** alla quale associamo la probabilità $PM(T)$;
- **Velocità:** associamo le unità di tempo di invio di ciascun heartbeat T;

Nell'heartbeating **all-to-all**, per ogni unità di tempo, si ha che:

$$L = \frac{N}{T}$$

Nell'heartbeating **gossip-based**, si ha che:

$$T = \log_2(N) \times T_{gossip} \quad ; \quad L = \frac{N}{T_{gossip}} = \frac{N \times \log_2(N)}{T}$$

Per ogni T_{gossip} , viene mandato un messaggio di gossip nell'ordine di $O(N)$, poiché si assume che ci siano N target a cui inviarlo.

Il caso peggiore

Dati T, $PM(T)$ e N, definiamo la probabilità di **perdere** un messaggio come p_{ml}, la quale è applicata a ciascun messaggio, indipendentemente dagli altri. Nel caso **peggiore**, da ciascun processo devono essere inviati un numero di messaggi pari a:

$$L^{\star} = \frac{\log\left(PM(T)\right)}{\log\left(p_{ml}\right)} \times \frac{1}{T}$$

Questa formula garantisce, anche in tale contesto che venga soddisfatta in pieno l'accuratezza. **Si noti** inoltre che il valore del carico peggiore NON è dipendente dal numero di nodi, e così è detto **scale free** (o **scale independent**).

In generale, il valore ottimale di L deve essere **indipendente** dal numero di nodi, e i sistemi di heartbeating **all-to-all** e **gossip-based** non soddisfano questo requisito, poiché $L = O(N/T)$.

Mappa capitolo