3. Hadoop e Protocolli di gossip

Cosa è un sistema distribuito

Per sistemi **distribuiti** si intende una serie di entità, ognuna delle quali è **programmabile**, **autonoma**, **asincrona** e pronta a reagire ai **guasti**, e che comunicano mediante dei mezzi **inaffidabili**. Per asincrona si intende che l'orologio delle macchine non è sincronizzato e quindi gli istanti di tempo sono differenti.

Il nostro interesse per sistemi distribuiti si focalizza in particolare su:

- **Algoritmi:** funzionamento, manutenzione e studio;
- **Entità:** processi in un dispositivo (PC, smartphone, ecc.);
- **Comunicazione:** reti senza fili o cablate.

Hadoop

Hadoop è una **piattaforma** open-source per il processing distribuito di grandi quantità di dati su cluster di computer, che dunque si sposa alla perfezione con le grid applications (che tratteremo a breve). È stato creato per gestire grandi dataset che non possono essere elaborati da un singolo computer, suddividendo il lavoro su più nodi all'interno del cluster. Hadoop viene utilizzato in diverse applicazioni, come ad esempio nell'analisi dei dati, nell'elaborazione di immagini e nella ricerca di informazioni su grandi dataset. Grazie alla sua scalabilità e alla **capacità di elaborare grandi quantità di dati**, Hadoop è diventato uno degli strumenti più popolari per la gestione e l'elaborazione dei big data.

MapReduce

Uno dei moduli più importanti di Hadoop è **MapReduce,** sviluppato da Google nel 2004. Si tratta di un framework usato per elaborare dati su cluster **Hadoop**.

MapReduce prevede due fasi principali: **map** e **reduce**.

1. **Map:** si occupa di applicare una generica funzione di mappatura a tutti gli elementi in input;
2. **Shuffle and sort:** si occupa di distribuire gli elementi mappati ai rispettivi nodi;
3. **Reduce:** si occupa di ridurre tutti gli elementi di una lista ad un unico risultato.

Fase di map

La fase di map consiste nell'associare ai vari dati una coppia **chiave-valore**. I dati di input vengono suddivisi e assegnati a dei **mapper**, e vengono etichettati sulla base dell'algoritmo specificato nella funzione map. Potrebbe anche essere scartata parte dei dati che è irrilevante ai fini dello scopo finale, nonché essere filtrati dati errati o poco attendibili. Infine, i dati vengono raggruppati in base a una chiave e al valore scelto. La stessa chiave può avere valori diversi.

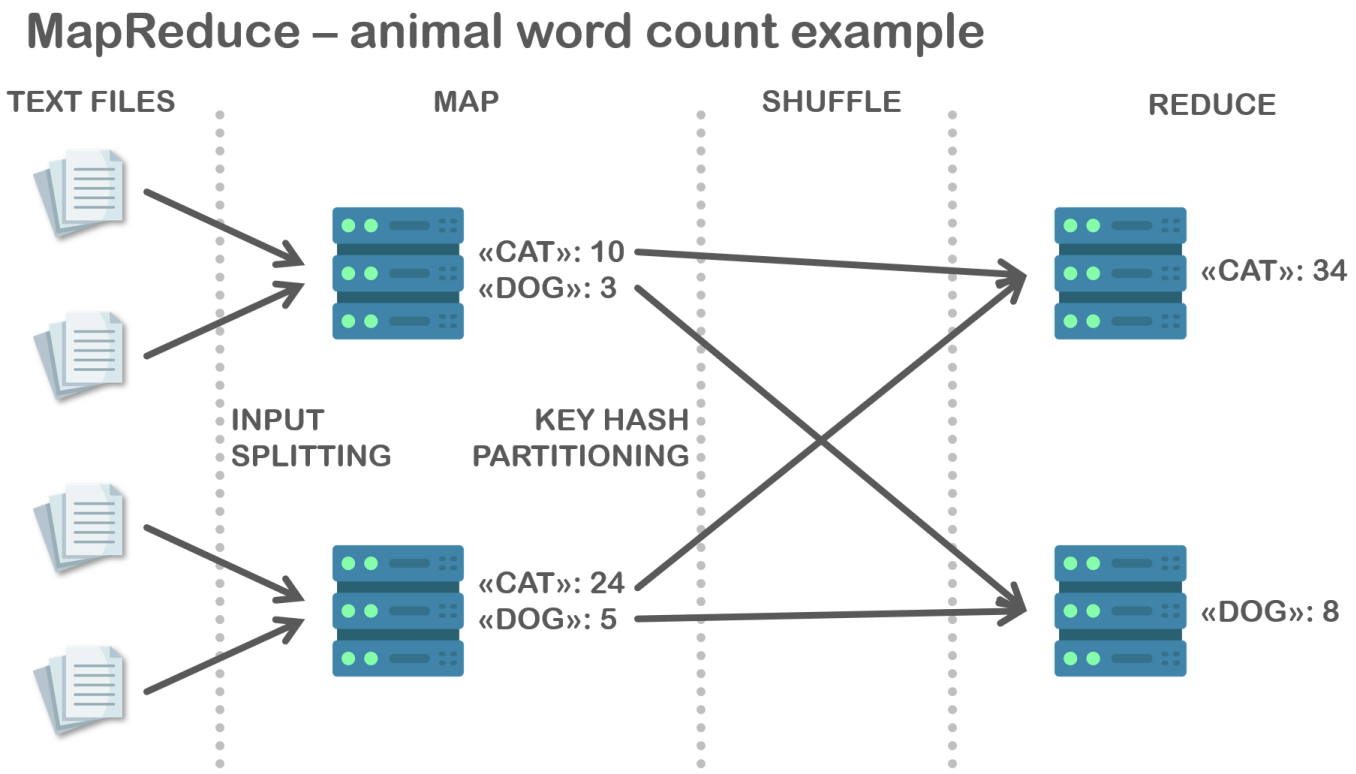
IVAN SOLLAZZO - ANDREA PISCOPO - NICOLAS BRUNO
PROFESSORE PIERLUIGI GALLO - INGEGNERIA CIBERNETICA
UNIVERSITA' DEGLI STUDI DI PALERMO

Fase di shuffle and sort

Le coppie chiave-valore restituite dai mapper vengono suddivise in base alla chiave e assegnate a dei **reducer**. Non possono esservi più reducer con la stessa chiave, ma ognuna di queste avrà il suo. Il partizionamento dei dati viene ordinato in modo tale che il processamento dei dati sia consecutivo.

Fase di reduce

Ciascun reducer avrà delle chiavi da elaborare, effettuando delle operazioni necessarie in base al compito stabilito nella funzione reduce. Anche in tal caso si trattano di coppie **chiave-valore**. L'operazione potrebbe essere: una media dei valori raccolti in base alla chiave, una somma, ecc.



Spiegazione:

1. In un algoritmo MapReduce, i file vengono suddivisi in pezzi più piccoli chiamati "input split", che sono poi inviati ai vari mapper per l'elaborazione parallela. Il criterio utilizzato per la suddivisione dei file dipende dall'implementazione specifica di MapReduce e dalle caratteristiche del sistema di archiviazione dei dati. In genere, i file vengono suddivisi in input split di dimensioni fisse o variabili, a seconda dell'implementazione di MapReduce. La dimensione degli input split può essere configurata in base alle esigenze del progetto e alle prestazioni del sistema.

2. I mapper filtrano i dati, rimuovendo quelli superflui e li formattano per avere una forma chiave-valore adeguata alle elaborazioni successive

3. Si fa l'hash della chiave applicando la formula: HASH(chiave) * MODULO(N° NODI TOT). Otterremo così una equa assegnazione dei dati da far elaborare ai nodi reduce

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

Esempio: il dataset del meteo

Un esempio per il quale potrebbe essere valido l'algoritmo MapReduce è quello di identificare in un dataset di varie stazioni meteorologiche qual è stata la temperatura più alta nei vari anni. Per ciascun anno, vengono raccolte alcune righe come le seguenti:

- **2016-06-09,19:00:00,23.3**
- **2016-06-09,20:00:00,22.5**
- **2016-06-09,21:00:00,22.0**
- **2016-06-09,22:00:00,22.0**
- **2015-05-01,22:00:00,34.0**
- **2015-05-02,20:00:00,37.0**
- **2014-01-01,20:00:00,38.0**

I mapper, una volta che hanno raccolto queste stringhe, possono procedere col rimuovere i dati irrilevanti, quali possono essere ad esempio: giorno, mese e orario. Infine, l'associazione delle coppie chiave-valore può avvenire nella forma anno-temperatura.

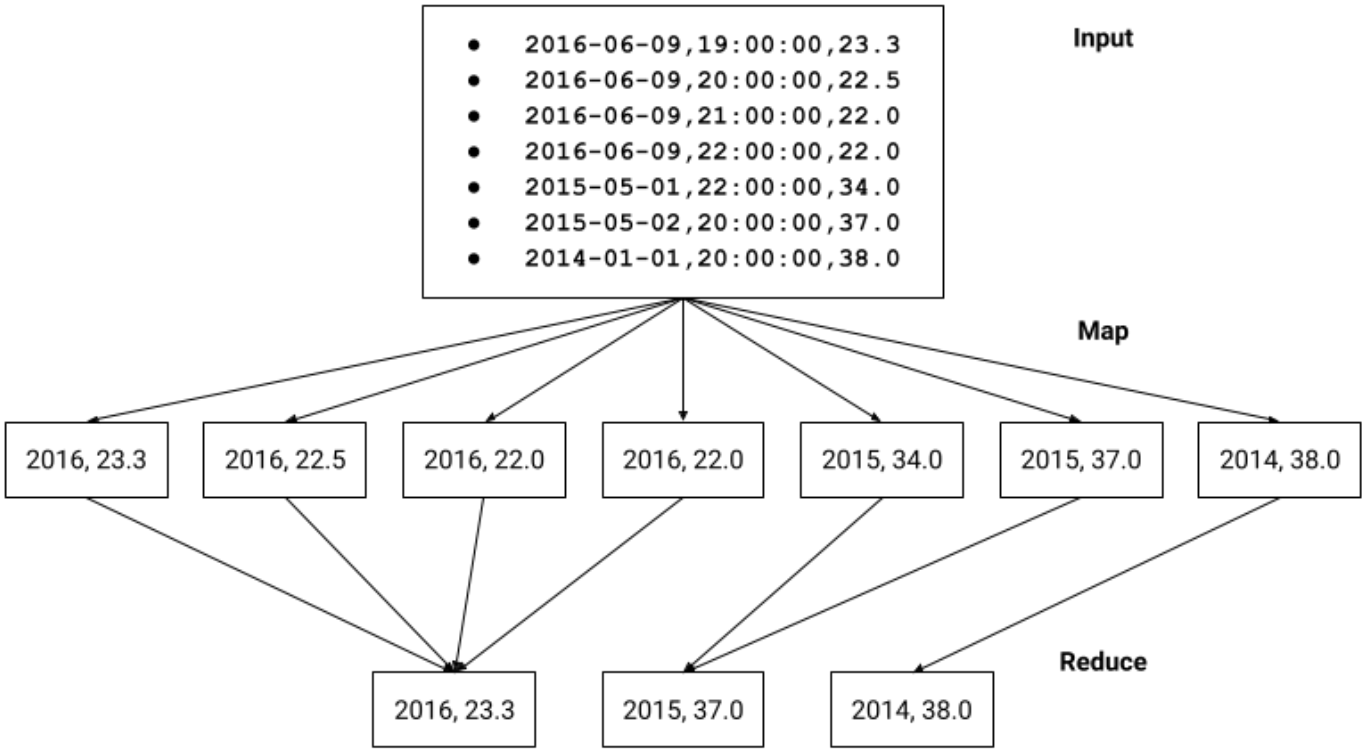
- **(2016,23.3)**
- **(2016,22.5)**
- **(2016,22.0)**
- **(2015,34.0)**
- **(2015,37.0)**
- **(2014,38.0)**

A questo punto, supponendo di avere tre nodi, ciascuno di essi potrebbe gestire il singolo anno. Tale fase è determinata dal shuffle and sort.

Dunque, se l'hash code di 2016 sarà 95192c98732387165bf8e396c0f2dad2 e il numero di macchine è di 100, la macchina di destinazione nella quale andranno i dati di temperatura legati al 2016 sarà la numero **16**. Infine, nella funzione di reduce, l'algoritmo specifica che devono essere scelte le temperature più alte, dunque il task di riduzione svolto dalle macchine assegnate sarà del seguente tipo:

- **(2016,23.3)**
- **(2015,37.0)**
- **(2014,38.0)**

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



YARN

è il sistema di gestione delle risorse di Hadoop che consente alle applicazioni di richiedere e utilizzare le risorse del cluster in modo efficiente.

E un modulo di Hadoop che si occupa di gestire i sistemi decentralizzati, è costituito da **Resource Manager** e **Node Manager**.

- **Resource Manager** gestisce e coordina l'allocazione delle risorse **a livello di cluster**, ed è responsabile della pianificazione e dell'allocazione delle risorse
- **Node Manager** gestisce l'esecuzione delle risorse **su ciascun nodo** del cluster e si occupa dell'esecuzione e del monitoraggio dei container sul **nodo locale**.

HDFS (Hadoop Distributed File System)

E' il file system **distribuito** di Hadoop, che consente di archiviare grandi quantità di dati su un cluster di macchine. HDFS fornisce un'alta affidabilità e tolleranza agli errori grazie alla replicazione dei dati su più nodi. MapReduce e YARN utilizzano entrambi HDFS come file system di archiviazione dei dati.

Un file system distribuito si differenzia da un file system locale per la possibilità di trattare una directory presente in un'altra posizione fisica come se facesse parte della gerarchia del file system.

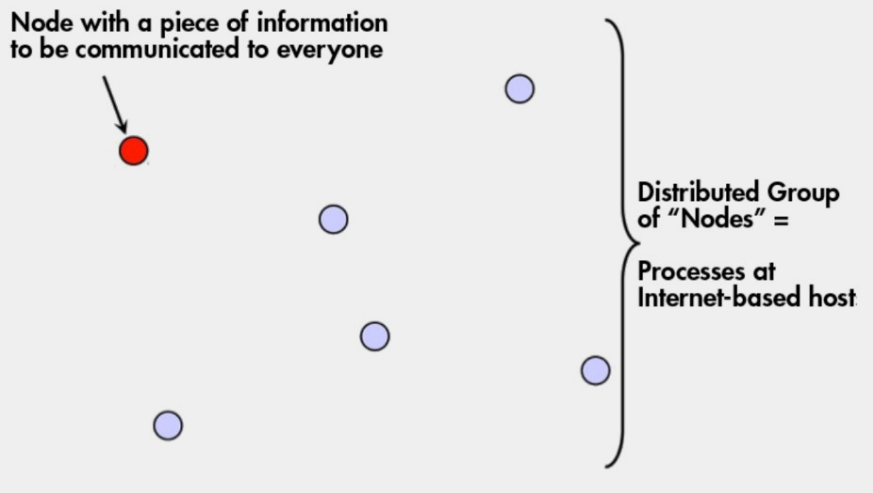
Gossip

Gli algoritmi di **Gossip** sono stati pensati con l'intento di risolvere il cosiddetto **Multicast Problem**.

Il problema del Multicast

Supponiamo di avere un gruppo di **processi**/**nodi**, ognuno dei quali è connesso alla rete. Questi processi devono essere in grado di scambiare messaggi tra loro.

Ad esempio, il nodo marchiato in **rosso** può avere una particolare informazione che intende condividere con altri nodi della rete, e non è detto che sia l'unico nodo a volerlo fare (può anche ricevere). Il **multicast** consente di inviare una data informazione solo a un determinato gruppo di nodi, a differenza del **broadcast**.



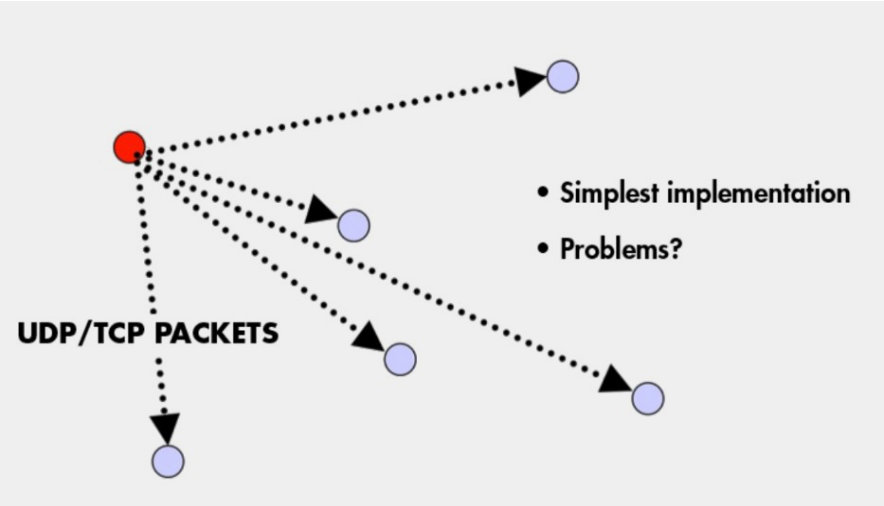
Requisiti

Nel cloud computing vi sono due **requisiti** fondamentali:

- La **scalabilità:** i nodi possono moltiplicarsi, triplicarsi, e così via;
- La **tolleranza ai guasti:** uno o più nodi potrebbero improvvisamente crashare, o alcuni pacchetti (TCP/UDP) potrebbero andare persi durante il tragitto.

L'approccio centralizzato, come quello appena introdotto, è l'implementazione più semplice. Tuttavia non offre alcun rimedio contro eventuali guasti e/o perdite di pacchetti.

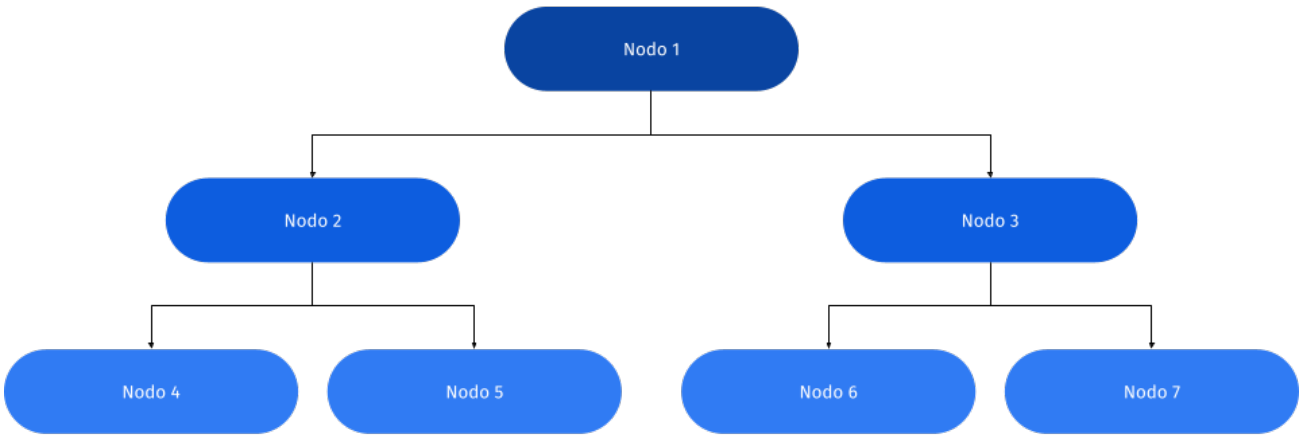
Esempio



Ad esempio, supponendo di avere un gruppo di sei nodi, qualora l'invio di un messaggio da parte di un singolo dovesse interrompersi a metà, solo quattro (escluso il nodo mittente) riceveranno il messaggio, situazione indesiderata, specialmente in presenza di processi **critici**.

Inoltre, il sovraccarico sul mittente è molto alto, poiché tutto viene gestito da tale mittente, aumentando anche la latenza e i tempi di ricezione, i quali possono essere nell'ordine di $O(N)$.

Multicast tree-based



La soluzione al problema potrebbe essere un approccio di tipo **tree-based**, in cui ciascun nodo genitore invia l'informazione ricevuta ai nodi figli. Questo approccio è detto **spanning tree**.

In tal modo, se si costruisce un albero di N nodi, diremo che l'altezza dell'albero sia dell'ordine di $O(log_2(N))$, così come la latenza per inviare il singolo messaggio ai singoli nodi.

Inoltre, a differenza del caso centralizzato, il carico si distribuisce su ciascun nodo, in quanto ciascuno di essi inoltra ai nodi del livello di gerarchia inferiore.

Il problema di questo approccio è senza ombra di dubbio la dipendenza dalla gerarchia: se si guasta, per esempio, il **Nodo 4**, non si verificheranno compromessi al resto della rete, poiché tale rappresenta il

livello più basso dell'albero. Tuttavia, se a guastarsi è il **Nodo 2**, o peggio ancora, il **Nodo 1**, anche i figli

rimarranno disconnessi dalla rete, fintantoché il nodo genitore non torni in rete o venga sostituito da un altro nodo.

I fallimenti sono sempre la norma nei sistemi distribuiti, per cui sarà la normalità incorrere in tali problematiche. Il problema è appunto come gestirle.

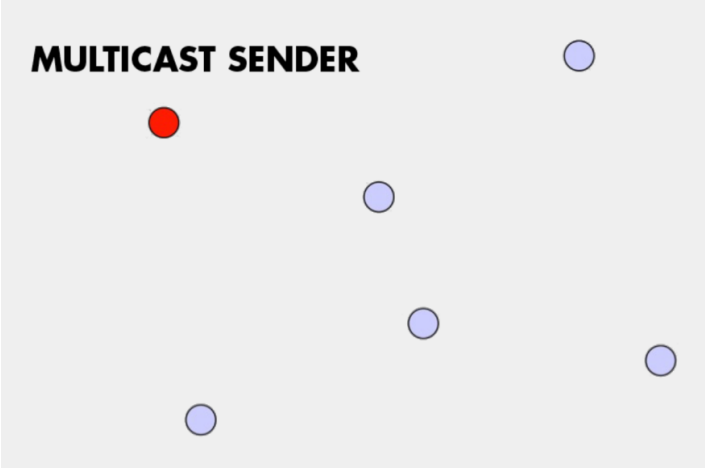
I protocolli multicast che sfruttano l'approccio tree-based sono:

- **SRM (Scalable Reliable Multicast):** è un protocollo di comunicazione utilizzato per trasferire dati in modo affidabile e efficiente tra mittenti e destinatari su una rete di computer. Quando un destinatario non riceve correttamente un pacchetto di dati, invia un NACK al mittente specificando il numero di sequenza del pacchetto mancante. Il mittente utilizza quindi i NACK ricevuti per determinare quali pacchetti non sono stati ricevuti correttamente e ritrasmetterli.
- **RMTP (Reliable Multicast Transport Protocol):** quando un messaggio multicast viene perso, invia degli ACK a dei ricevitori appositi, i quali provvederanno poi a ritrasmettere i multicast andati persi.

I suddetti protocolli, tuttavia, non sono scalabili così come erano stati concepiti, motivo per il quale introdurremo ora un nuovo approccio detto **epidemico**.

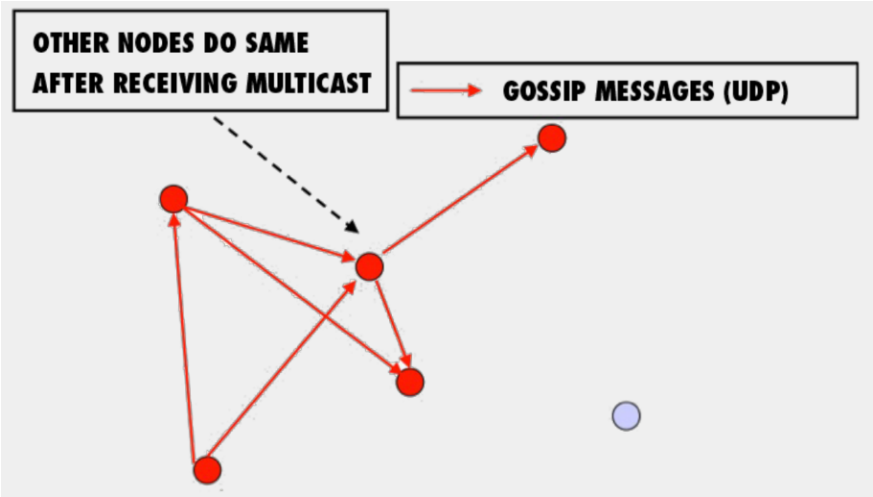
L'approccio epidemico

Dati n nodi nella nostra rete, supponiamo che il nodo **rosso** sia intenzionato ad inviare una data informazione a tutti gli altri.



L'utilizzo di questo approccio implica che il nodo scelga b nodi casuali ai quali inviare un messaggio ogni intervallo di tempo t (ad esempio, cinque secondi). In particolare, b è detto **fan out**, ed è un valore tipicamente molto piccolo. È inoltre possibile che gli stessi nodi possano essere scelti dall'algoritmo più di una volta per un successivo invio.

Ogni volta che un nodo riceve il messaggio, si dice che è stato **infettato**, e tale provvederà, con un solito intervallo temporale, a scegliere altri b nodi ai quali inoltrare tale messaggio.



In figura, i nodi infetti sono indicati in **rosso**, mentre quelli non infetti sono indicati in **blu** (in basso a dx).

Ancora una volta, l'algoritmo di gossip gira su ciascun nodo in modalità **asincrona**, pertanto, ciascun nodo avrà i suoi tempi di gossip e i suoi intervalli temporali.

Tipi di protocolli di gossip

I protocolli di gossip si dividono sostanzialmente in due tipi:

- **Push:** il nodo in possesso di un messaggio multicast, lo invia ai b nodi scelti a random. Tale è l'approccio descritto precedentemente;
- **Pull:** un nodo sceglie b nodi a random ai quali chiedere se ci sono stati dei messaggi multicast nuovi e che esso non ha ricevuto. Se sì, tali nodi provvederanno con l'invio al richiedente;
- **Variante ibrida (Push-Pull):** come il nome suggerisce, funziona in ambo i modi.

Il protocollo Push

I protocolli di Push sono molto veloci e leggeri anche in presenza di grandi reti di nodi, oltre che molto tolleranti ai guasti.

Un'analisi matematica

Questa analisi deriva da un vecchio ramo della matematica chiamato **epidemiologia**, il quale si occupata di studiare l'andamento di un'eventuale epidemia nella società.

Nell'approccio più classico:

- Si ha una popolazione di $(n + 1)$ individui mischiati in maniera omogenea;
- Il **contact rate**, cioè il range di contagio, è pari a β, il quale è un numero compreso tra 0 e 1;
- In qualsiasi momento, un individuo può essere infetto o non infetto. Indicheremo con la variabile x gli individui **infetti**, mentre con y gli individui **non infetti**.

Inizialmente, solo un individuo è infetto, tale da avere $y = y_0 = 1$, mentre $x = x_0 = n$, dove n è il numero totale di nodi, escluso il nodo infetto (da qui $n + 1$). Col tempo, i nodi non infetti vengono infettati da

nodi infetti e non guariscono.

L'andamento epidemiologico si può scrivere sotto forma di equazione differenziale, nel seguente modo:

$$\frac{dx}{dt} = -\beta xy$$

Nel dettaglio:

1. L'equazione differenziale è **negativa** poiché il numero di individui non infetti tende chiaramente a diminuire nel tempo;
2. xy è il **numero totale dei potenziali contatti infetti/non infetti per unità di tempo**. Questo prodotto viene moltiplicato per β poiché tale potenziale contatto dipende proprio dal contact rate.
    a. Inizialmente $x >> y$, ciò ne consegue che $\frac{dx}{dt} \to 0$;
    b. Al crescere di y, il prodotto produce una curva sempre più ripida;
    c. Ad un certo punto $y >> x$, la situazione si è sostanzialmente capovolta e si ha nuovamente $\frac{dx}{dt} \to 0$.

L'equazione differenziale ha le seguenti soluzioni:

$$x = \frac{n(n+1)}{n + e^{\beta(n+1)t}} \quad ; \quad y = \frac{(n+1)}{1 + ne^{-\beta(n+1)t}}$$

Da queste soluzioni si può constatare che:

- Per $t \to 0 \implies x = n+1 \land y = 1$;
- Per $t \to +\infty \implies x = 0 \land y = n+1$.

Contact rate

Nel gossip epidemico, il **contact rate** è pari a:

$$\beta = \frac{b}{n}$$

- b è il **fan out**;
- n è il **numero totale di nodi**.

Ovvero, uno sui b nodi complessivi. Dal momento in cui ci sono n nodi potenzialmente infettabili per unità di tempo, la probabilità che vada a infettare tale nodo non infetto sarà pari a b/n.

Perché ciò? Si considerino un nodo infetto e un particolare nodo non infetto. **La probabilità che il nodo infetto entri a contatto con questo particolare nodo non infetto** è pari a:

$$P(E) = \frac{1}{b}$$

Assumendo di aver derivato la soluzione y precedentemente determinata, si sostituisce al tempo $t = clog(n)$, il quale sta ad indicare che ogni unità di tempo sarà in ordine logaritmico moltiplicato per una costante. Si ottiene che il numero totale di nodi infetti sarà pari a:

$$y \simeq (n+1) - \frac{1}{n^{cb-2}}$$

In particolare, n + 1 è il numero totale di **infetti**, sottratto a una certa quantità dipendente da c e dal fan out b.

Supponendo che c e b siano due numeri molto piccoli, ad esempio pari a 2, il termine diventerà pari a $-1/n^2$, che è un valore molto vicino a zero.

In sostanza significa che: dopo un tempo pari a $2log_2(n)$, assunto b = 2, il numero di nodi infetti nel sistema sarà pari a $y \simeq (n+1) - 1/n^2$, dove quest'ultima è una quantità molto piccola tale da essere trascurata.

Pertanto: in scala logaritmica $y \simeq (n+1)$ se si scelgono due valori di c e b **molto piccoli**. Questo permette, non solo di avere una latenza bassissima, ma anche un'elevata affidabilità, in quanto:

- Entro $clog(n)$ round, tutti i nodi, ad eccezione della quantità $1/n^{cb-2}$, riceveranno il messaggio multicast **(affidabilità)**;
- Ciascun nodo avrà trasmesso non più di $cb\ log(n)$ multicast **(carico basso)**.

L'ordine $O(log_2(N))$

Perchè $O(log_2(N))$ è così sacrosanto? In teoria, tale valore non è proprio costante, ma è una funzione che cresce molto lentamente, a tal punto da farla sembrare costante. Facendo alcuni esempi in base 2, si ha infatti:

$$\log_2(1000) \simeq 10 \quad ; \quad \log_2(1M) \simeq 20 \quad ; \quad \log_2(1B) \simeq 30 \quad ; \quad \log_2(4294967296) \simeq 32$$

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

Fault tolerance nel gossip

Perdita di pacchetti

La distribuzione dei messaggi multicast nella rete avviene in modo lineare, ed è dettata dalla proporzione:

$$b : 100 = b' : percentuale\, pacchetti\, persi$$

Da cui il nuovo fan out è dato da::

$$b' = b \times \frac{percentuale\, pacchetti\, persi}{100}$$

Esempio

Supponendo che nella rete ci sia una perdita di pacchetti del 50%, l'analisi epidemica non dovrà essere fatta più con b, ma bensì $b/2$. Affinché tutti i nodi della rete ricevano un messaggio multicast ci vorrà il doppio del tempo, ossia $2 \times c \log_2(n)$ round.

Guasto a un nodo

L'analisi epidemica va effettuata anche sulla base della percentuale di nodi in vita. Il valore di n è dettato dalla proporzione:

$$n : 100 = n' : percentuale\, nodi\, attivi$$

Da cui il nuovo valore di n è dato da:

$$n' = n \times \frac{percentuale\, nodi\, attivi}{100}$$

In presenza di nodi guasti, anche il valore del fan out da analizzare sarà differente, poiché comunque si tenterà di inviare i messaggi multicast anche a quei nodi della rete che sono guasti, quindi risulterà anche un packet loss. Ciò è dettato dalla proporzione:

$$n : b = n' : b'$$

Da cui il nuovo valore di b è dato da:

$$b' = b \times \frac{n'}{n}$$

Esempio

Supponendo che il 50% dei nodi sia guasto, n' = $n/2$ e b' = $b/2$. In modo analogo, affinché tutti i nodi della rete ricevano un messaggio multicast ci vorrà il doppio del tempo, ossia $2 \times c \log_2(n)$ round.

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

Durata del gossip

La probabilità che la vita del gossip in una rete termini precocemente è molto bassa, per via della selezione randomica dei nodi a cui inviare. Inoltre, l'analisi effettuata precedentemente è stata effettuata nel caso peggiore, cioè nel caso in cui venissero infettati **tutti** i nodi. Nella realtà sono pochi i nodi a essere infettati.

In tutte le forme di gossip ci vogliono $O(log_2(N))$ round affinché almeno la metà dei nodi venga coinvoltanel gossip. Questo perchè nel caso più veloce per diffondere un messaggio ci vuole uno spanning tree con un fan out costante dell'ordine di $O(log_2(N))$.

Quanto detto vale, sia per gli algoritmi di pull, che per quelli di push. Inoltre, dopo che almeno la metà dei nodi abbiano ricevuto il gossip, il pull gossip è **più veloce** del push gossip.

Per renderci conto di ciò, facciamo in modo che dopo l'i-esimo round, p_i sia frazione di processi infetti o non infetti nel sistema.

DA RIVEDERE QUI SOTTO

La probabilità che alla fine del round i + 1 si rimanga infetti/non infetti coincide con la probabilità che ciascuno dei target infettati all'inizio di tale round non fosse infetta già prima, ovvero:

$$p_{i+1} = (p_i)^{k+1}$$

Dove k è il numero di target dell'algoritmo gossip e coincide col **fan out**.

Questa relazione è **super esponenziale** e di ordine molto più veloce, in quanto $O(log(log(N)))$.

Dipendenza dalla topologia della rete

Quanto discusso finora non tiene conto della topologia della rete, chiaramente in una subnet possono esserci nodi collegati ad altri nodi, ecc. Più a fondo si va, più è densa la topologia della rete.

Se teniamo conto di tale topologia anche nell'algoritmo di gossip, bisogna pensare che i router che coordinano la subnet sono soggetti a eventuali sovraccarichi.

In tal contesto, si consideri uno scenario con due subnet e un router, ognuna delle quali gestisce $n/2$ nodi. La selezione randomica del fan out pesa sui router $O(N)$, un ordine abbastanza grande.

Un modo per ovviare a ciò è: considerare una subnet, che contiene n nodi e scegliere il nodo target per il gossip con probabilità pari a $(1 - 1/n)_i$.

In tale modo, il carico del router si porta nell'ordine di $O(1)$, mentre il tempo di disseminazione è pari a $t = O(log(N))$.

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

Implementazioni del gossip

Fino ad ora abbiamo visto dei concetti prettamente teorici circa il gossip, ma nella pratica, quali implementazioni esistono di tale algoritmo?

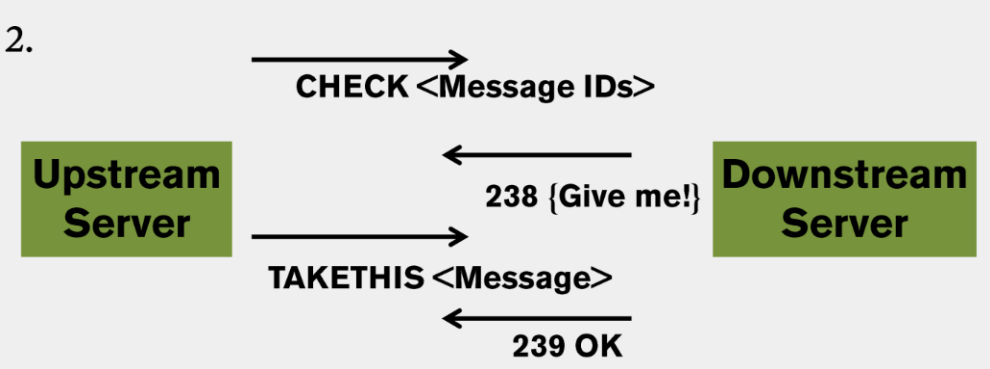
Essenzialmente, quei server che comunicano tra loro (si pensi ai server di posta). Anche Amazon implementa dei protocolli di gossip in AWS EC2 e S3, così come i sistemi con Cassandra.

Per ultimo, ma non per importanza, uno dei protocolli utilizzati per decenni a partire del '79, l'**NNTP** (Network News Transport Protocol) è un protocollo di comunicazione utilizzato per la distribuzione di articoli di news attraverso Internet. Il protocollo NNTP consente ai client di accedere e leggere le news dalle news server, e di pubblicare nuovi articoli su questi server. Inoltre, il protocollo consente anche la replica delle news tra diversi server, consentendo ai client di accedere alle news da diversi server.

Il protocollo NNTP è stato soppiantato da altre piattaforme di discussione su Internet come forum, mailing list e social media.

Il protocollo NNTP

Ciascun client carica e scarica news da un **news server**.



1. L'upstream server chiede al downstream server di verificare un certo *Message ID*;
2. Il downstream server risponde con una richiesta;
3. L'upstream fornisce il contenuto;
4. Il downstream risponde con una conferma.

Mappa capitolo: