1 Lezione del 08-10-25

1.1 Confronto prestazionale fra CS e P2P

Vediamo di valutare quale approccio, fra client-server e peer-to-peer, per un job semplice come un trasferimento di file *minimizza* il **tempo di trasferimento**.

Poniamo di avere N peer (o client, comunque n host che vogliono ricevere il file) e un server che contiene il file (di dimensione F). All'istante in cui il file viene reso disponibile ogni peer richiede il file, e vogliamo minimizzare il tempo che passa affinché tutti lo abbiano ricevuto.

Sia u_s la capacità di upload del server, u_i la capacità di upload dell'i-esimo peer e d_i la capacità di download dell'i-esimo peer.

- Con l'approccio **client-server**, il server deve inviare N copie del file di dimensione F, che con capacità di upload di u_s dà un tempo minimo di NF/u_s. Ogni client deve quindi ricevere la sua copia del file, che con capacità di download di d_i diventa F/d_i. Visto che il più lento è quello che pregiudica tuta la statistica, prendiamo la sua capacità di download come $d_{\min}$ e rifiniamo il bound:

  $$T_{cs} \geq \max\left(\frac{NF}{u_s}, \frac{F}{d_{\min}}\right)$$

  Notiamo che questo bound è $O(N)$.

- Con l'approccio **peer-to-peer**, il server dovrà inviare al minimo una copia del file, per cui il bound diventa F/u_s. Il client più lento sarà comunque un bottleneck, per cui c'è il termine $F/d_{\min}$. A questo punto l'ultimo bound viene preso come il tempo impiegato a trasferire il file a tutti, se tutti distribuiscono, cioè $NF/(u_s + \sum u_i)$. Questo dà il bound:

  $$T_{p2p} \geq \max\left(\frac{F}{u_s}, \frac{F}{d_{\min}}, \frac{NF}{u_s + \sum u_i}\right)$$

  Vediamo che il termine significativo è l'ultimo: il primo non dipende da N, mentre il secondo lo fa in maniera "lasca" (aumentando N troveremo host più lenti, ma non linearmente e prima o poi stabilizzandoci). Il terzo termine dipende da N, ma ancora non linearmente. Posto $\bar{u}$ come la velocita media di upload dei peer, possiamo approssimare come:

  $$T_{p2p} \sim \frac{NF}{u_s + \sum_1^N u_i} \approx \frac{NF}{u_s + N\bar{u}}$$

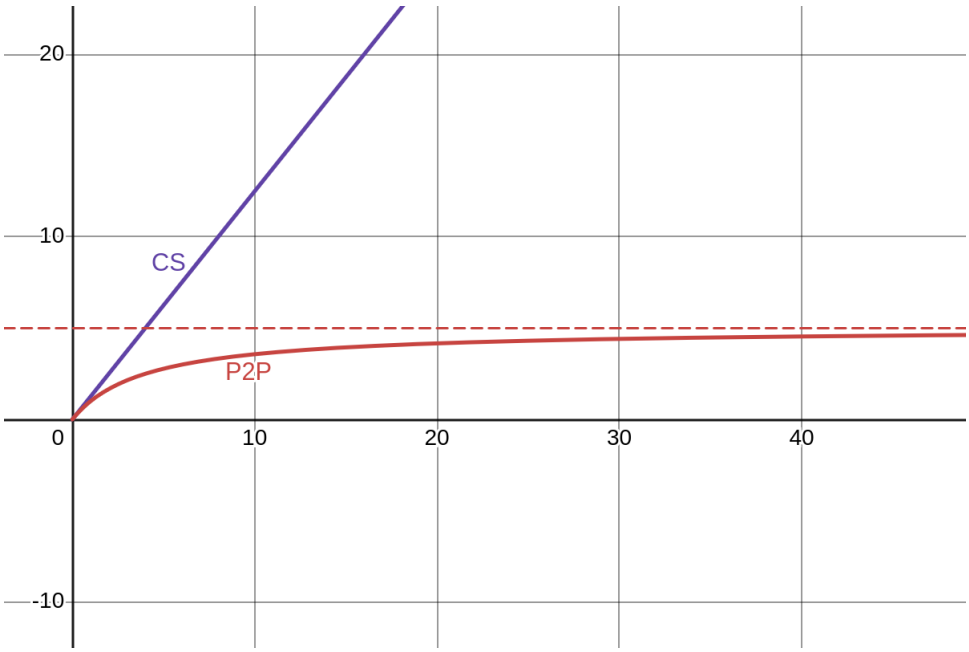
  che per $N \to \infty$ è:

  $$\lim_{N\to\infty} \frac{NF}{u_s + N\bar{u}} = \frac{F}{\bar{u}}$$

  ammesso di conoscere $\bar{u}$.

  Abbiamo quindi che il bound è migliore rispetto a quello dell'approccio client-server, in quanto prima o poi converge ad un valore finito.

Concludiamo tracciando su un grafico l'andamento complessivo dei bound più svantaggiosi dei tempi di trasferimento:



1.2 Protocollo BitTorrent

Approfondiamo le reti P2P, e le *overlay network*, studiando un protocollo reale: il **protocollo BitTorrent**.

In BitTorrent non è previsto un server centrale che distribuisce il file. Ogni file è diviso in più **chunk**, solitamente di 256 KiB ciascuno.

Un **torrent** è un gruppo di peer che si impegnano a scambiarsi chunk di un file. Un server più o meno centralizzato detto **tracker** traccia i peer che partecipano ad un torrent.

- Quando un utente vuole scaricare un file, chiede al tracker la lista dei peer che partecipano al torrent dedicato a quel file. Solitamente questo è un *metafile* che contiene l'IP del tracker, a cui si può quindi richiedere il torrent vero e proprio.
- I peer che l'utente riceve vengono detti **vicini**: sono connessi da una connessione TCP, e possono quindi scambiarsi dei messaggi (chunk del file). Possiamo definire così gli utenti collegati ad un torrent:
  - **Leecher**: (*"sanguisughe"*), non hanno una copia completa del file e lo stanno ancora scaricando;
  - **Seeders**: (*"alimentatori"*), hanno già il file, e lo rendono disponibile agli altri.
- Il trasferimento da seeder a leecher si fa un chunk per volta, adottando una politica **Rarest-First** (*RF*): si inizia a scaricare dal chunk che meno peer sul torrent hanno disponibile. Questo è chiaramente mirato a ottenere i chunk che potrebbero sparire dalla rete il prima possibile.

  La lista dei chunk disponibili è fatta richiedendo periodicamente a ogni peer quali chunk possiedono.

- Man di mano che i leecher ottengono chunk, devono anche iniziare a rimetterli nella rete ai leecher nuovi arrivati. Su BitTorrent si usa la politica **Tit for Tat** (*TT*): si inviano chunk ai 4 peer che ci stanno inviando chunk a frequenza più alta. Questi top 4 vengono ricalcolati ogni 10 secondi, e gli altri vengono *"strozzati"* (non più serviti).

  Questo significa che i peer potrebbero legarsi eccessivamente fra di loro: ogni 30 secondi si sceglie allora un'altro peer a caso, facendo una previsione *ottimistica* (quel peer potrebbe arrivare fra i nuovi top 4 più frequenti).

  In ogni caso, l'obiettivo è quello di trovare buoni partner per il trasferimento file, in modo da ottenere il file più velocemente possibile.

La politica *Tit for Tat* di BitTorrent è chiaramente pensata per scoraggiare i *freeloader*: non si può scaricare se non si dà qualcosa in cambio, cioè si partecipa attivamente al trasferimento rimettendo chunk in circolo.

1.3 Streaming video e CDN

Vediamo come si possono creare tecnologie per il trasferimento massiccio di contenuti multimediali, con riferimento allo **streaming video** e i **CDN** (*Content Distribution Networks*).

I problemi sono la **scala** (come raggiungere molti utenti?) e l'**eterogeneità** (diversi utenti hanno diverse caratteristiche di trasmissione, dispositivi di visualizzazione, ecc...).

La soluzione è infrastruttura **distribuita** al livello application.

1.3.1 Video

Un **video** è una sequenza di immagini dette **frame** mostrate a frequenza costante (standard 24, 30, e 60 **FPS** (*Frames Per Second*)). Un **frame** è un'array di pixel, dove ogni pixel è rappresentato da 3 valori (corrispondenti ai colori primari additivi *rosso*, *verde* e *blu*).

La tecnica del **coding** consiste nell'usare ridondanza intrinseca dentro e fra i frame per ridurre i bit usati nel processo di codifica:

- Si usa la correlazione **spaziale**, dentro i singoli frame;
- Si usa la correlazione **temporale**, fra diversi frame.

Esistono più tipi di coding, fra cui notiamo:

- **CBR** (*Constant Bit Rate*), a bitrate costante;
- **VBR** (*Variable Bit Rate*), a bitrate variabile;

1.3.2 Streaming

Lo **streaming** di video allocati in remoto consiste nel spedire contenuti video abbastanza velocemente da renderli disponibile in *tempo reale* (a differenza del semplice download, che prevede una fase iniziale di acquisizione e una seconda fase di visualizzazione quando il file è ormai già tutto sul disco locale).

Il server dovrà quindi inviare al client frame con una frequenza prefissata (quella del contenuto video), e il client potrà mostrarli appena arrivati. Assunta una rete ideale,

quindi a ritardo *costante*, questo ci permetterebe di visualizzare il video così com'è (al pari di tale ritardo costante).

Purtroppo la rete non è ideale, e il ritardo è quindi *variabile* nel tempo. Possiamo quindi sfruttare il processo del **buffering**: aspettiamo un po' lato client prima di mostrare i frame (per un tempo detto *client playout delay*), mettendoli nel frattempo in un *buffer* di memoria. Quando iniziamo a visualizzare i frame, ci aspettiamo che il buffer sia abbastanza *"pieno"* da permettere la visualizzazione di un segmento di video abbastanza lungo da permettere l'acquisizione dei frame successivi, e così via.