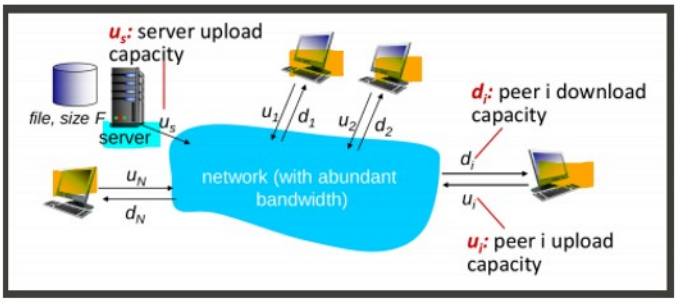
Applicazioni P2P

mercoledì 2 agosto 2023 16:29

Nel peer-to-peer ci sono coppie di host connessi, chiamati peer, che comunicano direttamente l'uno con l'altro. In una distribuzione di file client-server, il server deve inviare una copia del file a ciascun peer che fa richiesta, in PSP invece il carico viene distribuito aiutando in questo modo il server nel processo di distribuzione.
Supponiamo di avere una situazione …



- 1 server, N peer
- us è la banda di upload del collegamento di accesso alla rete del server
- ui è la banda di upload dell'i-esimo peer
- F è la dimensione (in bit) del file da distribuire
- Distribution time: tempo richiesto perché tutti gli N peer ottengano una copia del file

Internet ha abbastanza banda quindi i colli di bottiglia sono sulle reti d'accesso, comunque la banda di accesso in upload e download dei peer è completamente dedicata alla distribuzione di questo file. Analizziamo il tempo di distribuzione nelle diverse architetture.

Caso architettura client-server

Il tempo per distribuire 1 copia è L/R, dove L è F bit del file e R è us bps la velocità con cui il collegamento trasmette il file. Quindi per una copia il tempo è $\frac{F}{us}$ ne consegue che per N copie da inoltrare(1 per ogni peer) il tempo sia: $\frac{N*F}{us}$.
Ora, ogni i-esimo peer, deve scaricare(dowload) la copia del file, prendiamo la banda di dowload piu' bassa quindi dmin = min(d1, d2,…, dN), il peer con velocità piu' bassa di dowload non potrà ricevere tutti gli F bit in meno di $\frac{F}{dmin}$.
Mettiamo insieme queste due informazioni e otteniamo un limite inferiore, che corrisponde al tempo minimo che ci vuole a distribuire il file a tutti i peer:

Dcs >= max $\{\frac{N*F}{us}, \frac{F}{dmin}\}$

Caso architettura P2P

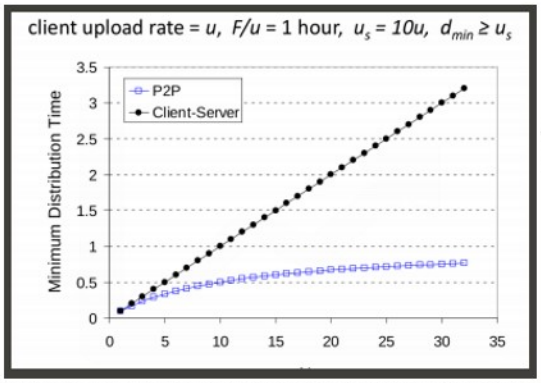
In questa architettura ogni peer assiste il server nella distribuzione del file. Quando un peer riceve alcune porzioni del file, può usare la propria capacità' di upload per redistribuire i dati agli altri peer.
Dobbiamo quindi calcolare il tempo **MINIMO** di distribuzione:

- All'inizio della distribuzione, solo il server dispone del file → deve inviare ciascun bit del file almeno una volta nel suo collegamento di accesso che ha banda di upload del server pari a us. Il tempo minimo di distribuzione del file ad almeno un peer è di $\frac{F}{us}$;
- Come nell'architettura client-server, ogni i-esimo peer deve scaricare la copia del file, analogamente prendiamo la banda di dowload minima tra i peer e viene calcolato $\frac{F}{dmin}$;
- La capacità totale di upload del sistema nel suo complesso è uguale alla velocità di upload del server + la velocità di upload di ciascun peer, quindi utot = us + $\sum_{i=0}^{N} ui$;
- Sapendo che il sistema deve consegnare(upload) F bit a ciascuno degli N pee, consegno quindi *N·F* bit e questo posso farlo con una velocita' massima di upload di utot. Di conseguenza il tempo di distribuzione minimo e' almeno: $\frac{N*F}{utot} = \frac{N*F}{us+\sum_{i=0}^{N} ui}$.

Il tempo minimo di distribuzione in questo caso è:

Dcs >= max $\{\frac{F}{us}, \frac{F}{dmin}, \frac{N*F}{us+\sum_{i=0}^{N} ui}\}$

Caso architettura client-server vs Caso architettura P2P



il tempo dell'architettura client-server cresce linearmente quello dell'architettura P2P cresce logaritmicamente

Ci sono diversi protocolli che implementano P2P, *i primi tre che vedremo riguardano solo la fase di richiesta, quella di dowload viene fatta con http*

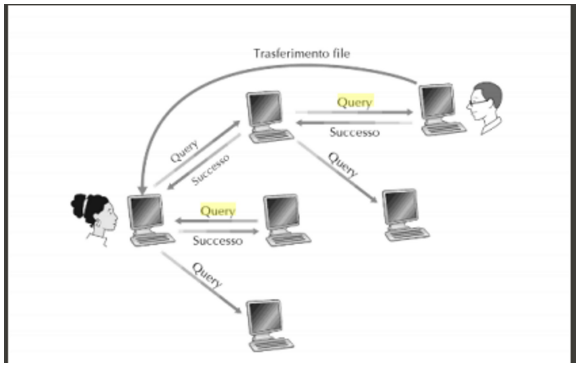
Directory centralizzata(Napster)

È l'approccio piu' diretto alla localizzazione dei contenuti, si basa sul concetto di indice centralizzato ovvero un potente server di directory centralizzato.
Quando un utente lancia l'applicazione per la condivisione di file P2P, l'applicazione informa il server, indice centralizzato, del proprio indirizzo IP e del nome dei file resi disponibili per la condivisione.
Successivamente il server raccoglie queste informazioni dai peer attivati, creando un database centralizzato e dinamico che associa ciascuna copia del file a un insieme di indirizzi IP.
Svantaggio: unico punto di guasto.

Query flooding (Gnutella)

Non abbiamo un server centralizzato bensì dei peer: i peer inviano messaggi a quelli vicini sulla rete di copertura (overlay

network, rete logica e astratta creata dai peer) su connessioni TCP persistenti.



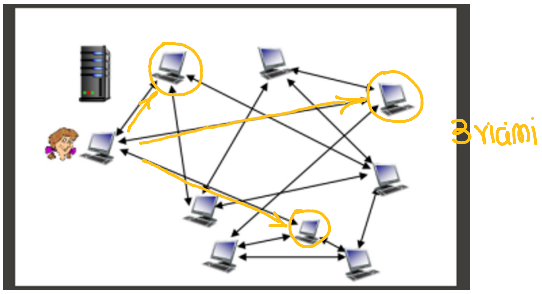
Alice vuole localizzare ad esempio "Network love", il suo client invia ai propri vicini un messaggio di richiesta che include parole chiave "Network love". A loro volta tutti i peer vicini di Alice inoltrano i messaggi di richiesta ai propri peer vicini, i quali a loro volta instradano il messaggio ai vicini e cosi via.
Quando un peer riceve un messaggio di richiesta, controlla se le parole chiave sono presenti in qualche file disponibile per la condivisione, se trova la corrispondenza invia ad Alice un messaggio di successo (query-bit), che contiene il nome del file e la dimensione.
Questo messaggio segue il percorso inverso rispetto al messaggio query, usando sempre connessioni TCP persistenti.

Nota: *tra i problemi di questo modello sono dovuti all'overhead a causa della poca scalabilità, tutte le volte che un peer fa partire una richiesta genera una significativa quantità di traffico che si accumula.*
Questo problema si risolve specificando quando, Alice invia il proprio messaggio iniziale di richiesta, un campo di conteggio che funziona come limite di richieste a catena.

Modello di copertura gerarchica
Questo modello combina le caratteristiche migliori degli altri due approcci: usa un approccio distribuito per localizzare i file, tuttavia non tutti i peer sono uguali, ci sono peer con connessioni a banda più larga e con elevata Disponibilità, designati come ***leader di gruppo***.
Un nuovo peer stabilisce una connessione TCP con uno dei leader di gruppo, a cui notifica tutti i file che sta mettendo in condivisione, consentendo ai leader di gruppo di mantenere un database che include gli identificatori di tutti i file condivisi dai loro "figli", le informazioni mantenute sono: **i metadati sui file** e i **corrispondenti indirizzi IP dei figli che li detengono.**
In questo modo il leader di gruppo diventa un mini indice.

Ora introduciamo **i protocolli P2P di seconda generazione**, viene eliminata la fase di ricerca e presa in considerazione solo la fase di dowload.

Protocollo P2P di seconda generazione: BitTorrent
L'insieme di tutti i peer che partecipano alla distribuzione di un particolare file è chiamato **torrent**.
I bit peer di un torrent scaricano **chunk** (parti) del file di uguale dimensione (tipicamente 256 kbyte).
Quando un peer entra per la prima volta a far parte di un torrent non ha nessun chunk del file, con il passare del tempo accumula sempre piu' parti che mentre scarica invia agli altri peer. Quando ha ottenuto tutte le parti di un file, ha due vie: lasciare il torrent (potendo anche rientrare in seguito), oppure rimanere nel torrent e continuare a inviare chunk agli altri peer. *Come funziona?*



Alice entra a far parte di un torrent: successivamente il **tracker**(nodo di infrastruttura) seleziona in modo casuale un sottoinsieme di peer (es.50) dall'insieme dei peer che stanno partecipando a quel torrent e invia ad Alice l'indirizzo IP di questi 50 peer.
Alice cerca di stabilire delle connessioni TCP contemporanee con tutti i peer della lista, con chi ci riesce li chiameremo neighbour peer. *Nota: i peer vicini cambiano nel tempo.*
Ne consegue che in un determinato istante ciascun peer del torrent avrà un sottoinsieme dei **chunk** (parti) di un file.
A questo punto periodicamente Alice chiederà a ciascuno dei suoi vicini la lista dei chunk del file in loro possesso.
Sappiamo quindi che Alice ha un sottoinsieme dei chunk del file e sa quali sono i chuck che i loro vicini hanno, con queste informazioni Alice deve prendere due decisioni:

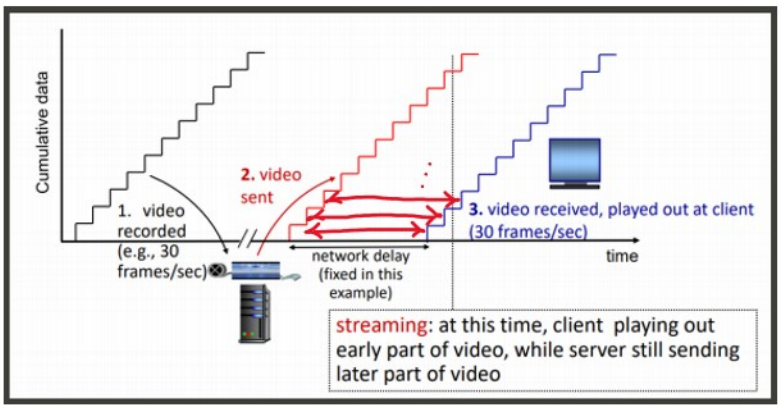
1. Quali chunk deve richiedere per prima ai suoi vicini: tecnica **rare first**, il piu' raro per prima;
2. A quali vicini dovrebbe inviare i chunk a lei richiesti: per determinare a queli richieste rispondere si usa l'algoritmo di trading;

Per ciascuno dei suoi peer vicini, Alice misura continuamente la velocità alla quale riceve i bit e determina i quattro peer che stanno passando i bit alla velocità piu' elevata.
Alice invia i chunk del file a quegli stessi quattro peer.
Ogni 10 secondi viene ricalcolata la velocità, quindi l'insieme dei peer varia nel tempo
I quattro peer che appartengono all'insieme vengono detti **unchoked**("non soffocati" o "non limitati"), il singolo peer **optimistically unchoked**.
Tutti gli altri peer che non ricevono alcun chunk da Alice vengono detti **choked**(limitati).

I video su internet
Un video è una sequenza di immagini, visualizzate a velocità costante (es. 30 immagini al secondo).
Un immagine non compresa e codificata digitalmente consiste in un array di pixel, ognuno dei quali codificato con un numero di bit per rappresentare il colore e sfumature.
I video possono essere compressi in modo tale da raggiungere un compromesso tra qualità del video e **bit rate** *(velocità di trasmissione dei bit)*: maggiore è il bit rate, migliore è la qualità dell'immagine.
Inoltre quando codifichiamo, possiamo utilizzare la **ridondanza** per ridurre il numero di bit utilizzati per codificare l'immagine, attraverso la **codifica spaziale**: invio solo due valori, valore del colore(es. viola) e numero di ripetizioni(es. cui si ripete il colore viola nelle immagini) e quella **temporale**: invio le differenze da un dato frame invece di inviare l'intero fotogramma.
Abbiamo le seguenti codifiche con cui vengono trasmessi i dati:

- **CBR (constant bit rate)**, codifica fissa;
- **VBR(Variable bit rate)**, si sfruttano i meccanismi della codifica spaziale e temporale.

Di base abbiamo una situazione in cui c'è un client che richiede a un server, in cui abbiamo memorizzati i video, un video e il server gli risponde inviandolo.
Dobbiamo però fare una distinzione tra video preregistrato e video in diretta. Ad esempio se ho richiesto un video registrato, ho che il **thorughput massimo**, maggiore di quello che mi serve, posso avere quindi più bit di quelli che sto consumando, i bit in eccesso vengono memorizzati in un buffer.
Quando il buffer è vuoto, parte la rotellina di caricamento.



Il vincolo di riproduzione continua è che questa deve essere temporizzata. Tuttavia questo non è possibile, i ritardi della rete possono essere variabili.
La soluzione è utilizzare un buffer lato client