Relazione di consegna per il progetto
Prefetch Proxy

Sviluppato da Alain Di Chiappari

Progetto per il corso di Reti, scritto, sviluppato e testato su Intel x86 con sistema operativo Ubuntu Linux v10.04.

ABSTRACT

Il proxy può essere visto concettualmente a livelli.

Ad un **primo livello** il proxy si pone ai client come un vero e proprio server, che rimane in ascolto, aspettando connessioni, fino ad un limite massimo. Per ognuna di queste connessioni, il proxy fa partire una "sessione" (realizzata come thread a sé stante), che ha lo scopo di:

- Ricevere la richiesta dal client.
- Scandire la richiesta, verificandone la correttezza dei protocolli mhttp/mhtml.
- Verificare la presenza in cache del file richiesto. (vedi livello successivo)
- Inoltrare la richiesta al server adeguato, per la risorsa specificata dall'URL, o mandando un messaggio di errore al client in caso di incoerenze di protocollo.
- Aspettare la richiesta dal server, gestendo eventuali errori/blocchi/ritardi.
- Scandire la risposta per verificare la correttezza.
- Rispondere al client (comprese risposte d'errore e gestione ritardi).
- Salvare il file in cache se non presente. (vedi livello successivo).
- Mettere in moto il prefetching.

Ad un **secondo livello** il proxy si occupa del prefetching delle risorse.
Ciò che accade è essenzialmente questo:

- Alla ricezione di un file, il primo livello, mette in moto il parsing deL file, (controllando preventivamente l'eventuale presenza in cache dello stesso, vedi livello successivo) cercando URL di tipo REF, IDX/REF.
- Per ogni URL trovato, viene fatto partire un thread che si occuperà del download sul proxy della risorsa richiamata, ovviamente con gestione degli errori, e quant'altro.
- Il punto precedente attua quello che si chiama "prefetching di 1° e 2° livello", che dovrebbe migliorare le prestazioni del proxy con l'aumentare del tempo, riuscendo a rispondere ai client senza un'intera sessione per oguno di questi.
- Questo livello si occupa anche di caricare in cache le risorse indicate nelle sequenze URL REF presenti nelle pagine mthml raccolte in prefetching (implementando quindi il "prefetching di 3° livello"), anche queste mediante threads che lavorano separatamente, parallelizzando molto il lavoro del proxy.

Il **terzo livello** del proxy, è quello che comunica direttamente con i file di cache, e le strutture di manutenzione ad essi collegati.
Le funzioni di questo livello infatti si occupano di:

- Creazione di una lista di strutture, manipolata continuamente, per il corretto funzionamento del caching.
- Creazione di strutture da inserire nella lista, per ogni file che viene inserito in cache.
- Eliminazione di strutture per file di cache eliminati.
- Ovviamente gestisce la creazione e l'eliminazione dei file stessi.

- Controllo dell'esistenza di un determinato file nella cache, sfruttando l'URL, come hash, essendo univoco per le risorse in rete.
- Controllo dell'expired dei file in cache, e qualora ne venga trovato qualcuno “scaduto” viene eliminato sia fisicamente dalla cache, che logicamente dalle strutture del proxy.
- Creazione di nomi univoci per i file all'interno della cache, al fine di evitare collisioni.

Esiste poi uno strato di **funzioni ausiliarie** ai livelli stessi, che si occupano della comunicazione di rete, della gestione degli errori, della comunicazione con i file, e del parsing.

Infine, (attualmente in via di ottimizzazione) è implementato un “**thread demone**”, attivabile mediante macro dal file “const.h” che si occupa della periodica pulizia delle strutture di manutenzione di cache, eliminando quei file, e strutture logiche collegate, che sono “expired” e quindi non inoltrabili ai client.
Questa funzione viene già assicurata comunque da una funzione del modulo “caching.c” che però a run-time riconosce questi file e li elimina.

MODULI

---------- Modulo “proxy.c” ----------

Questo modulo contiene il main() del proxy, incaricato di essere il “lato server” del proxy, con un socket in stato listen, e un ciclo di accept, uno per ogni connessione con client e quindi sessioni gestite da thread appositi, il modulo inoltre inizializza le strutture e prepara ciò di cui ha bisogno il funzionamento del proxy.
Oltre al main() è definito qui anche la funzione Session, che dà vita ai thread di sessione, come sopra definito.
In stato attualmente instabile vi è la definizione del thread daemon_cache_clean, per la pulizia costante della cache.

---------- Modulo “caching.c” ----------

Il modulo di caching, definisce il terzo livello di astrazione del proxy, ovvero quello che comunica con la cache, i suoi file, e le strutture logiche ad esso associate.
Gli altri moduli si rifanno ad esso per interrogazioni ed azioni sulla cache.

---------- Modulo “prefetching.c” ----------

Qui è definito il solo thread get_and_cache, che realizza il prefetching vero e proprio, parsando le pagine in arrivo, e identificando le sequenze URL da caricare, come definito nell'abstract, per il 1°, 2° e 3° livello di prefetching.

---------- Modulo “util.c” ----------

Le funzioni utili alle gestione dei parsing, della comunicazione di rete e tutto di quanto ausiliario ai moduli “operativi” è implementato in questo modulo/libreria.

---------- Moduli “struct.h / const.h” ----------

Vengono qui definite le strutture necessarie al proxy, e le costanti utilizzate al suo interno.
Vi è possibilità di attivare le funzioni di mutua esclusione sulla cache (stabilizzando il server, a patto di una minore prestazione), di attivare un demone di pulizia cache, come sopra specificato, e di modificare parametri del proxy, che incidono sulle prestazioni a seconda delle situazioni, come

ad esempio la tolleranza agli errori o ai ritardi, indicando i secondi di timeout concessi o i tentativi disponibili, il numero di thread massimi consentiti e così via.

---------- Modulo “extern.e” ----------

Questi moduli servono solo come interfaccia tra i vari moduli, basta quindi includere questi file all'interno del modulo voluto, per usare come “libreria” le funzioni messe a disposizione dai sorgenti.

BUG RILEVATI

Sono state rilevati due situazioni di errore o malfunzionamento del proxy; una riguardante il parsing di richieste GET/INF, nel caso in cui i pacchetti TCP in arrivo siano stati riempiti per pochi byte dal kernel a lato client, il server non riconosce come valide stringhe non errate, ma solo momentaneamente incomplete; un altro caso è la mancata chiusura dei socket descriptor nella simulazione di errore in risposta o blocco del server, cosa che a lungo andare porta il kernel all'indisposizione nell'apertura di nuovi file descriptor.

NOTE e TODO

Si potrebbero unire alcune funzioni molto simili, pulendo codice quasi duplicato, ottimizzare alcune situazioni, talvolta restringendo l'area di azione delle mutue esclusioni, o migliorando alcuni controlli.
Si dovrebbe inoltre stabilizzare l'azione del demone di pulitura in cache, che all'attivazione, dopo un certo periodo di tempo, manda in crash il proxy.
Si nota che a seconda delle costanti definite in “const.h” il proxy si comporta in modo molto diverso, bisogna quindi calibrare i valori in base al carico, e a quanta affidabilità richiedono i client nell'arrivo delle pagine.

CONCLUSIONI

Si nota che ovviamente il proxy in partenza ha una velocità di risposta inferiore rispetto all'avanzare del tempo, in quanto il prefetching, rispondendo in maniera adeguata alle richieste, permette di non sostenere l'intero carico di sessioni complete tra client, proxy e server, utilizzando i file già caricati.
Tutto ciò è vero non prescindendo però da ciò che è l'andamento della rete, simulato dal server, e dai parametri interni del proxy, che possono far variare non poco le prestazioni.
Spesso si verifica infatti che a prezzo di un calo di velocità si ha una maggiore affidabilità verso il client, ad esempio “sprecando” più tempo e tentativi nella richiesta di file al server; viceversa si ottiene più velocità e reattività dal proxy se si permette ad esso di adoperare meno tempo nel perseverare in richieste al server.