Servizi di Content Delivery per comunitá di utenti

Vittorio Manetti, Roberto Canonico, Walter de Donato, Giorgio Ventre
Dipartimento di Informatica e Sistemistica
Universitá di Napoli Federico II
via Claudio 21, 80125 Napoli, ITALY
Email: {vittorio.manetti, roberto.canonico, walter.dedonato, giorgio.ventre}@unina.it

I. Introduzione

L'eterogeneitá dei contenuti offerti in Internet é estremamente elevata, é possibile tuttavia individuare delle comunitá di utenti [1] definite come tali sia per quanto riguarda l'infrastruttura utilizzata per l'accesso alla rete, sia per quanto riguarda gli interessi dai quali sono accomunati. Molto spesso le comunitá di utenti sono organizzate in sottogruppi, e molto spesso si rivolgono a terze parti per ricevere un servizio di caching che migliori le prestazioni relative alla fruizione di contenuti messi a disposizione da uno o piú *Content Provider.*

Il generico Content Provider fornisce i contenuti a cui gli utenti della comunitá accedono con elevata frequenza, e la cui fruizione é molto spesso soggetta a stringenti requisiti in termini di qualitá del servizio (*Quality of Service - QoS*). Generalmente é il Content Provider stesso ad occuparsi di problematiche relative alla QoS per i servizi offerti ai propri clienti; nello scenario che consideriamo é invece la comunitá a farsi carico di garantire ai propri utenti elevate prestazioni in relazione alla fruizione dei contenuti. Le ragioni di questa scelta potrebbero ad esempio essere legate ad un costo troppo elevato da riconoscere al Content Provider per la fruizione di determinati servizi, o magari all'assoluta impossibilitá di quest'ultimo a garantire la QoS richiesta dagli utenti della comunitá.

Obiettivo del nostro lavoro é quello di realizzare una infrastruttura di rete che implementi servizi per il caching e la distribuzione di contenuti Web a comunitá di utenti. Entrambi i servizi vengono eseguiti sfruttando delle cache messe a disposizione da terze parti, quindi uno o piú *Internet Service Provider* (*ISP*), ed offerti solo per determinati domini, quelli a cui gli utenti fanno accesso con maggior frequenza. Lo scenario che immaginiamo é quello di una comunitá costituita da utenti suddivisi in un determianato numero di cluster, dove utenti appartenenti al medesimo cluster utilizzano la stessa infrastruttura di accesso alla rete (es.: connessione ADSL) e condividono i medesimi interessi, come nel caso di studenti della medesima Facoltá.

L'obiettivo é quello di incrementare la qualitá percepita dal generico utente implementando meccanismi di caching che riducano la latenza relativa al meccanismo di accesso ai contenuti richiesti. Il modello proposto é organizzato in maniera tale che gli utenti della comunitá abbiano la possibilitá di sottoscrivere il servizio presso il Service Provider in maniera dinamica.

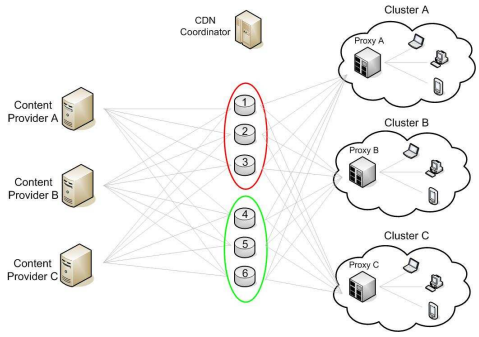

Fig. 1. Architettura Centralizzata

La restante parte dell'articolo é organizzata nel seguente modo: presenteremo nella sezione II una architettura centralizzata che fornisce servizi di caching e delivery di contenuti Web a comunitá di utenti, ed in quella successiva i risultati ottenuti da alcuni studi sperimentali eseguiti su tale architettura. Nella sezione IV presenteremo invece un modello *peer-to-peer based* ottenuto come evoluzione di quello introdotto in precedenza.

II. Un sistema centralizzato per il Content Delivery

L'architettura che proponiamo é composta da un certo numero di entitá alle quali é assegnato il compito di gestire alcune repliche di contenuti Web all´interno di un sistema di cache, al fine di ottimizzare la qualitá percepita dal generico utente della comunitá.

Le cache che compongono il sistema sono, come anticipato, fornite da terze parti, e dislocate all'interno della rete. Ogni cache é utilizzata per implementare le funzionalitá tipiche, ed alcuni moduli per il monitoraggio e la configurazione automatica.

Introduciamo le entitá *Proxy* e *CDN Coordinator.*

A. Il nodo Proxy

Obiettivo del modello é quello di mettere a disposizione degli utenti alcune politiche di *request routing* di tipo *content-aware.* La soluzione che adottiamo é quella di introdurre un nodo Proxy per ogni cluster di utenti. Compito del Proxy é quello di instradare le richieste sulla base di informazioni relative alla localizzazione dei contenuti nel sistema di cache.

I Proxy vengono periodicamente istruiti dal CDN Coordinator (che in seguito verrá introdotto) riguardo le politiche di routing da adottare per redirigere la specifica richiesta verso la cache opportuna.

Ogni nodo Proxy gestisce una *Content Routing Table*; la tabella realizza la corrispondenza tra i contenuti e le relative cache. Ogni riga della tabella é formata dalla coppia *chiave-valore*; il primo parametro costituisce la chiave di accesso alle informazioni, quindi l'URL della risorsa, mentre il secondo costituisce l'identificativo della cache verso cui la richiesta deve essere rediretta.

Essendo la destinazione di tutte le richieste inoltrate dagli utenti del generico cluster, il Proxy dev'essere inoltre in grado di redirigere la richiesta direttamente verso il Content Provider di origine, nel caso eventuale in cui lo specifico contenuto non sia stato preventivamente caricato all'interno del sistema di caching.

B. Il nodo CDN Coordinator

Il CDN Coordinator svolge nell'ordine le seguenti funzioni:

- raccolta di informazioni;
- calcolo dello schema di localizzazione ottimo;
- caricamento dei contenuti nelle cache;
- aggiornamento delle Content Routing Table dei proxy.

Il coordinatore si occupa della raccolta di informazioni relative a: stato della rete, banda a disposizione, traffico generato dagli utenti di ogni cluster, numero di accessi ad ogni risorsa. La misura di tali parametri viene eseguita da specifiche entitá e non direttamente dal CDN Coordinator. Un apposito software calcola la banda disponibile tra la rete di accesso utilizzata dagli utenti della comunitá ed il sistema di cache, ed il coordinatore rileva tali informazioni in maniera periodica. Per quanto riguarda invece le statistiche relative all'accesso dei contenuti memorizzati all'interno delle cache, vengono direttamente rilevate dalle cache stesse.

Il principale compito assegnato al CDN Coordinator é quello di calcolare lo schema di localizzazione ottimo dei contenuti Web all'interno del sistema di cache, e a tale scopo implementa un modello di programmazione lineare. Lo schema di localizzazione viene calcolato periodicamente, e a seguito di tale operazione il CDN Coordinator predispone la rete affinché vengano rispettate le politiche di request routing ottenute come risultato della computazione.

La strategia utilizzata per il caricamento dei contenuti nel sistema di caching é di tipo *push-based*, quindi il coordinatore provvede a caricare le cache ordinando loro di prelevare i contenuti presso i server d'origine, memorizzarli e renderli poi disponibili agli utenti.

Successivamente, il Proxy dev'essere messo al corrente dello schema di localizzazione dei contenuti appena calcolato, e sulle relative politiche di request routing. Interviene nuovamente il CDN Coordinator, il quale é a conoscenza di tali informazioni ed esegue l'aggiornamento delle tabelle di routing dei Proxy. Viene utilizzato a tale scopo un protocollo di comunicazione appositamente realizzato (*Content Routine Management Protocol - CRMP*), grazie al quale il CDN Coordinator accede alla Content Routing Table dei Proxy per inserire, aggiornare, eliminare righe della tabella.

C. Alcuni dettagli implementativi

Al fine di implementare le funzionalitá svolte dalla generica componente cache, utilizziamo un proxy cache server Open Source, *Squid*. Per eseguire invece la misura della banda disponibile tra i nodi Proxy e le Cache, adoperiamo il software *PathChirp*. Il CDN Coordinator é costituito da un framework che ha lo scopo di gestire una serie di plugin che si occupano della raccolta di informazioni, tra cui la lista dei Proxy e delle Cache che compongono il sistema che il coordinatore ha il compito di gestire. Nel prototipo realizzato sono stati implementati due plugin: lo *SquidStatsPlugin* e il *PathChirp-Plugin*. Il primo interroga periodicamente le Cache per rilevare le statistiche di accesso ai contenuti selezionati ottenute analizzando i file di log di Squid, il secondo si occupa invece di rilevare le informazioni misurate da PathChirp riguardo la larghezza di banda disponibile sul percorso che va dal *sender* al *receiver*.

III. Studi sperimentali

Per valutare le prestazioni dell'architettura proposta in uno scenario realistico, é stata effettuata una serie di esperimenti sfruttando l'infrastruttura Planetlab. Il testbed realizzato a tale scopo conta cinque nodi di rete collocati:

- presso l'Universitá Federico II di Napoli
  - client Wget (CL)
  - server Proxy (PX)
- presso l'Universidad Carlo III di Madrid
  - server Cache Squid (C1)
- presso l'Universite De Technologie De Troyes
  - server Cache Squid (C2)
- presso il centro Supelec di Rennes
  - server web Apache (WS)

La scelta dei nodi é stata effettuata in maniera tale da evidenziare i benefici introdotti dall'architettura. In particolare, il web server é stato collocato in una rete con accesso ad Internet a banda limitata per rendere evidente la convenienza del caching dei contenuti da esso forniti, e le due cache sono state posizionate in luoghi geograficamente lontani tra loro e con connettivitá di capacitá differente.

Gli esperimenti sono stati condotti valutando i tempi di consegna dei contenuti in diverse situazioni. Per fare ció si é collocato sul web server un oggetto di dimensione pari a 4.162.048 bytes e sono state effettuate 8 richieste per tale contenuto da parte del client in diversi scenari. Gli scenari sono stati predisposti in modo tale che il contenuto sia ottenuto:

- direttamente dal Web Server (CL→WS)
- dalle Cache in caso di miss (CL→C1→WS e CL→C2→WS)
- da una Cache in caso di hit (CL→C1 e C1→C2)
- dal server web attraverso il Proxy (Cl→PX→WS)
- dalle Cache attraverso il Proxy in caso di miss (CL→PX→C1→WS e CL→PX→C1→WS)

	Scenario	$\mu[s]$	$\sigma[s]$
A	CL → WS	47.9	9.6
B	CL → PX → WS	73.2	7.7
C	CL → C1 → WS	69.9	10.6
D	CL → C1	3.5	0.6
E	CL → PX → C1 → WS	58.7	7.5
F	CL → PX → C1	3.4	0.2
G	CL → C2 → WS	128.6	14.1
H	CL → C2	78.7	13.5
I	CL → PX → C2 → WS	125.5	25.0
L	CL → PX → C2	72.8	7.9

TABLE I
TEMPO DI RISPOSTA OTTENUTO NELLA RICHIESTA DEL CONTENUTO

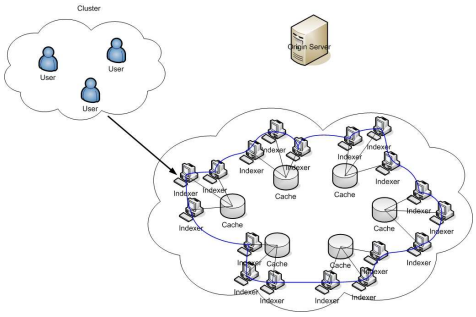

Fig. 2. Architettura peer-to-peer

- dalle Cache attraverso il Proxy in caso di hit (CL→PX→C1 e CL→PX→C1)

Di tali richieste é stato valutato il tempo di risposta, cioé quello che intercorre tra l'invio della richiesta e la completa ricezione del contenuto. In tabella I si riportano la media e la deviazione standard dei tempi di risposta ottenuti nei diversi scenari.

Dai risultati si evince che le prestazioni ottimali si ottengono prelevando il contenuto da C1, poiché essa dispone di una migliore connettivitá rispetto a WS e C2. Inoltre, si evidenzia come la vicinanza geografica non implichi prestazioni migliori, essendo C1 in Spagna e C2 e WS entrmbi in Francia. Infine, si puó notare dai risultati degli scenari C, E, G e I che l'introduzione del Proxy tra il Client e le Cache non degrada le prestazioni. Tali risultati confermano che un opportuna configurazione della tabella di content routing del Proxy puó migliorare significativamente le prestazioni percepite dagli utenti.

IV. EVOLUZIONE VERSO UNA INFRASTRUTTURA PEER-TO-PEER

L'architettura peer-to-peer che proponiamo é strutturata come una *Stealth DHT* [2]; i nodi che compongono una rete di questo tipo possono essere classificati come:

- *service node*: possiede le medesime caratteristiche del generico peer di una DHT classica, é quindi responsabile dello storage di contenuti, del routing e del forwarding di richieste;
- *stealth node*: non esegue storage di contenuti. Gestisce una routing table con una singola entry ed é in grado di identificare solo il *first hop* verso cui inoltrare una richiesta; non puó ricevere richieste perché non é in grado di gestirle. Durante il *join* ad una DHT esegue solo la fase di *state gathering* e non esegue la fase di *announcement*, per cui rimane trasparente ai service node.

Utilizziamo l'organizzazione delle Stealth DHT perché non soffrono di problemi dovuti al *churning* dei nodi trasparenti, ed il modello che proponiamo é caratterizzato dalla presenza di un elevato numero di nodi non stabili.

Nella nostra architettura il generico terminale utente puó essere identificato come la combinazione di due componenti:

- *browser*: é l'interfaccia utilizzata dall'utente per accedere ai servizi offerti dall'architettura;
- *stealth node*: é il punto di accesso alla DHT.

Il browser possiede un plug-in che, in riferimento allo specifico contenuto richiesto, é in grado di verificare se il servizio di caching e delivery é fornito o meno; in caso negativo, invia una richiesta direttamente verso il server di origine.

Identifichiamo il generico service node come *Indexer*; ad ogni Indexer é associata una Cache secondo una relazione del tipo 1 a 1 oppure n a 1. L'Indexer possiede una interfaccia DHT mediante la quale comunica con altri service node e con gli stealth node, ed una interfaccia HTTP mediante la quale comunica con la Cache. Gli utenti (stealth node) appartenenti al medesimo cluster fanno riferimento al medesimo Indexer (service node) come first hop per le loro richieste. L'Indexer non viene messo a disposizione dall'ISP, ma é una risorsa della comunitá: il generico terminale utente che verifica determinati requisiti di stabilitá, puó essere eletto Indexer.

La differenza sostanziale tra l'architettura che proponiamo ed una DHT classica, risiede nel fatto che il nodo Indexer a differenza del generico nodo di una DHT, non é responsabile del caching di contenuti ma solo del routing di richieste. Tale scelta é dovuta al fatto che, come anticipato, l'Indexer é un comune terminale utente che utilizza con buona probabilitá una connessione ADSL; in riferimento allo scenario descritto, fare in modo che il traffico relativo alle richieste gestite dall'Indexer passi attraverso il nodo stesso, risulta inopportuno sia per motivi legati alle prestazioni che per motivi legati alla privacy.

Eseguendo un rapido ed intuitivo confronto tra l'architettura centralizzata presentata in precedenza e quella peer-to-peer based appena introdotta, risulta evidente che in questo secondo caso non c'é piú l´esigenza di utilizzare un nodo singolo che si occupi della gestione dell'infrastruttura, con gli evidenti vantaggi che ne conseguono.

REFERENCES

[1] T. Plagemann, R. Canonico, J. Domingo-Pascual, C. Guerrero, and A. Mauthe, "Infrastructures for Community Networks," *Content Delivery Networks*, p. 367, 2008.

[2] A. Brampton, A. MacQuire, I. Rai, N. Race, and L. Mathy, "Stealth Distributed Hash Table: A robust and flexible super-peered DHT," in *Proceedings of the 2006 ACM CoNEXT conference*. ACM New York, NY, USA, 2006.