Cache-Control: *immutable*

Marco Colognese VR386474
Mattia Rossini VR386327

Indice

Capitolo 1

Uno spreco di risorse

Circa un anno fa gli sviluppatori di Facebook portarono al "*IETF HTTP Working Group*" un problema che stavano riscontrando: una grande quantità (circa il 20%) del traffico di dati verso i loro server era rappresentato da richieste di aggiornamento per risorse con un lungo "*cache lifetime*" ed il numero era destinano a crescere.

Queste richieste, poiché riguardano risorse che non subiscono modifiche per periodi molto lunghi, vengono sempre liquidate con un "*304 - Not Modified*".

Ciò rappresenta un inutile aumento di traffico nella rete e uno spreco di tempo per client e server.

1.1 La soluzione ideale

Tale problema è facilmente riscontrabile su siti quali social network (es. *Facebook*) o siti di news in tempo reale, in cui si aggiorna spesso la pagina per avere sempre nuovi contenuti; però vengono aggiornate anche risorse "statiche" come i loghi del sito.

La soluzione ideale, per evitare questo inutile spreco, consiste nel mandare richieste solamente per le risorse dinamiche, tralasciando quelle che non subiranno modifiche per periodi di tempo molto lunghi.

Capitolo 2

L'estensione *immutable*

Per implementare questa soluzione si è pensato di aggiungere un token nel campo Cache-Control dell'header Http: *immutable* (inizialmente chiamato *never-revalidate*).

Questa modifica (presente solamente da *Firefox 49*) ha un grosso vantaggio: può essere attuata senza andare a modificare le cache già esistenti poiché è un'estensione dell'informazione, non del funzionamento. Infatti tale token deve essere considerato solo dalle cache che lo riconoscono, altrimenti viene semplicemente ignorato.

2.1 Il suo utilizzo

Questa estensione deve essere applicata a tutte quelle risorse che, a detta dei *server administrator*, non subiranno mai modifiche (o almeno non le subiranno per tempi molto lunghi).

Una cache che si trova di fronte ad una richiesta per una risorsa *immutable*, deve mandare al client la copia in essa salvata senza inviare mai nessuna richiesta al server per verificare che essa sia aggiornata.

Per trarne effettivamente dei vantaggi, però, è necessario che tale token non venga assegnato a risorse con un "*max-age*" troppo breve, altrimenti è molto probabile che si verifichi una inconsistenza di informazioni.

L'implementazione corretta di questo token richiede delle modifiche a lato client e server:

- i client devono installare dei plug-in che permettano alle loro memorie cache di riconoscere l'estensione "*immutable*";
- i server administrator devono aggiungere il token al campo Cache-Control dell'header Http di ogni risorsa che non sarà modificata per periodi di tempo molto lunghi;

Capitolo 3

Una novità poco diffusa

Immutable è attualmente un token sperimentale e poco diffuso. Dopo aver analizzato le risorse principali di molte pagine web, si è potuto constatare che il suo utilizzo è limitato solamente al sito principale di *Facebook*.

Tale estensione, infatti, non si è diffusa (al momento) neanche nei siti più frequentati al mondo quali:

- Google;
- YouTube;
- Yahoo;
- Amazon;
- Wikipedia;
- Twitter;
- Instagram e WhatsApp (nonostante siano entrambi di proprietà del fondatore di *FacebookInc.*).

Oltretutto, il token non è stato implementato nemmeno nei siti affiliati a *Facebook* (appartenenti al dominio *Fb.com*) come:

- nonprofits.fb.com;
- insights.fb.com;
- newsroom.fb.com;
- workplace.fb.com;
- live.fb.com.

Capitolo 4

I primi dubbi

Un problema che era stato sollevato riguarda l'utilizzo del "*max-age*": se esso scade, la risorsa viene aggiornata? No, poiché i *server administrator* assicurano che l'estensione *immutable* viene utilizzata solamente per informazioni che restano appunto immutate.

Se tale informazione dovesse essere modificata, le verrà assegnato un nuovo URL.

4.1 *Hard-refresh*

C'è comunque un modo per aggiornare la pagina senza tener conto delle risorse in cache, si tratta di un *refresh* non condizionale che permette di mandare una richiesta per risorse aggiornate ignorando la cache (*cmd+shift+R* oppure *ctrl+F5*). Facendo ciò, possiamo avere la certezza di non avere inconsistenza di risorse.

Questa funzione viene utilizzata anche per ignorare *immutable* nel caso in cui fossero arrivate al client delle risorse troncate a causa di una cattiva connessione; un *hard-refresh* è necessario per ricaricare correttamente la pagina.

Però sono davvero pochi gli utenti che conoscono e fanno uso di questa funzionalità.

4.2 Problemi in caso di connessione lenta

Un altro problema si verifica quando, nel tentativo di aggiornare una pagina tramite semplice *refresh*, si presentano problemi di rete (connessione lenta) che impediscono nuovamente di ottenere il caricamento completo. Ciò accade poiché le richieste "*If-Modified-Since*" e "*If-None-Match*" effettuano controlli rispettivamente sui campi *Last Modified* ed *Etag*, non su eventuali troncamenti (es. *Content-Length*); la risposta, dunque, sarà sempre un "*304 - Not Modified*" anche se non è davvero così.

4.2.1 *OnStopRequest*: un token ausiliario

Una soluzione proposta consiste nel continuare ad utilizzare la copia già salvata in cache, incrementandone il tempo di validazione; il client rischia però di usufruire di risorse non aggiornate per un tempo non precisato (con possibili inconsistenze).

E' stato anche suggerito l'utilizzo del token "*OnStopRequest*" sulla entry della cache che contiene una risorsa che non deve essere riutilizzata, poiché corrotta.

4.3 *Immutable-subresources*

Venne inoltre proposto, in alternativa all'estensione *immutable*, l'utilizzo del token "*immutable-subresources*" sulla pagina principale del sito web. Esso avrebbe indicato che tutte le sottorisorse della pagina sono immutabili (in modo da non dover indicare *immutable* su ogni risorsa).

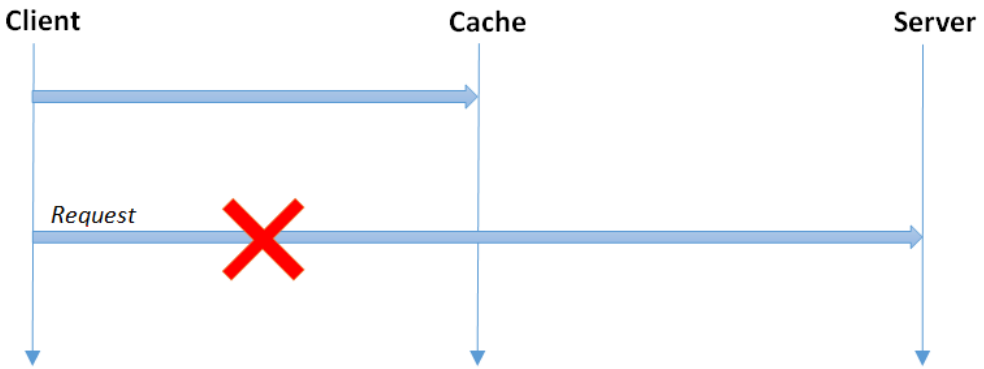
La proposta è stata subito bocciata poiché è molto raro trovare pagine con tutte le sottorisorse immutabili.

Capitolo 5

Diagrammi sul funzionamento

5.1 Risorsa *immutable*

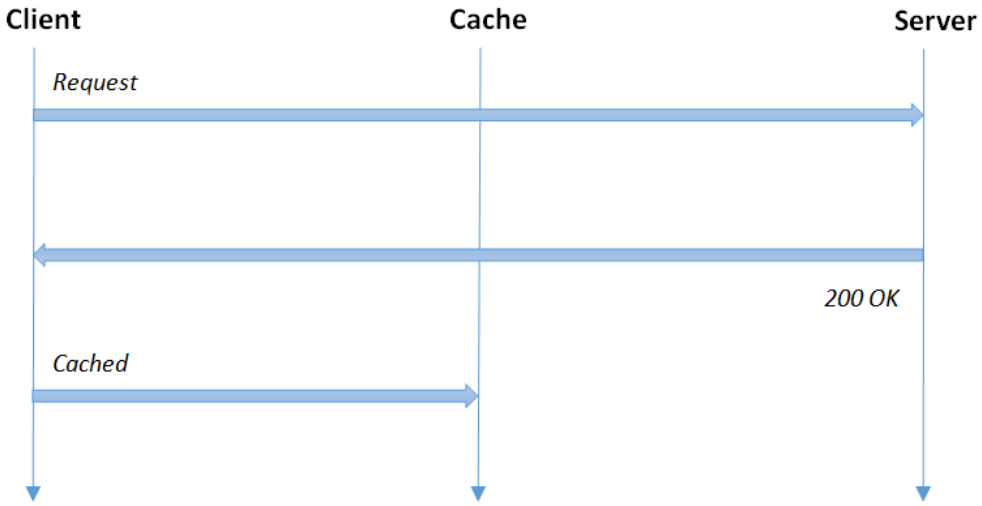
5.1.1 *Refresh* condizionale



Supponiamo che la risorsa richiesta sia stata precedentemente salvata in cache e che venga effettuato un *refresh* condizionale:

- Il client, al momento del *refresh*, verifica che la copia della risorsa sia presente in cache;
- Dopo aver verificato che sia presente il token *immutable*, viene utilizzata la risorsa senza effettuare ulteriori controlli riguardo la sua validità;
- Il client non andrà **mai** a richiedere una copia aggiornata della risorsa al server.

5.1.2 *Hard-refresh*



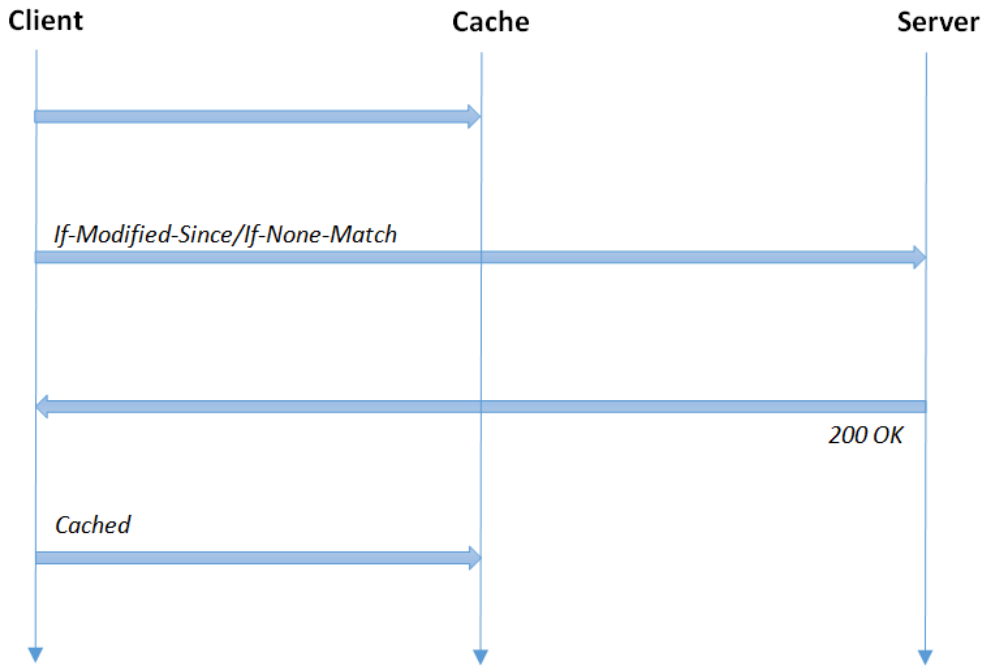
Supponiamo ora che la risorsa richiesta sia stata precedentemente salvata in cache ma che venga effettuato invece un *hard-refresh*:

- Il client, al momento del *refresh*, richiede la risorsa direttamente al server ignorando la cache e di conseguenza il token *immutable* e il "*max-age*";
- Il server restituisce la risorsa richiesta dal client aggiornata a quel momento (*response = 200 OK*);
- Il client salva una copia della risorsa aggiornata nella memoria cache.

In caso di *hard-refresh* si verificherà lo stesso comportamento anche in assenza del token *immutable*.

5.2 Risorsa *non-immutable*

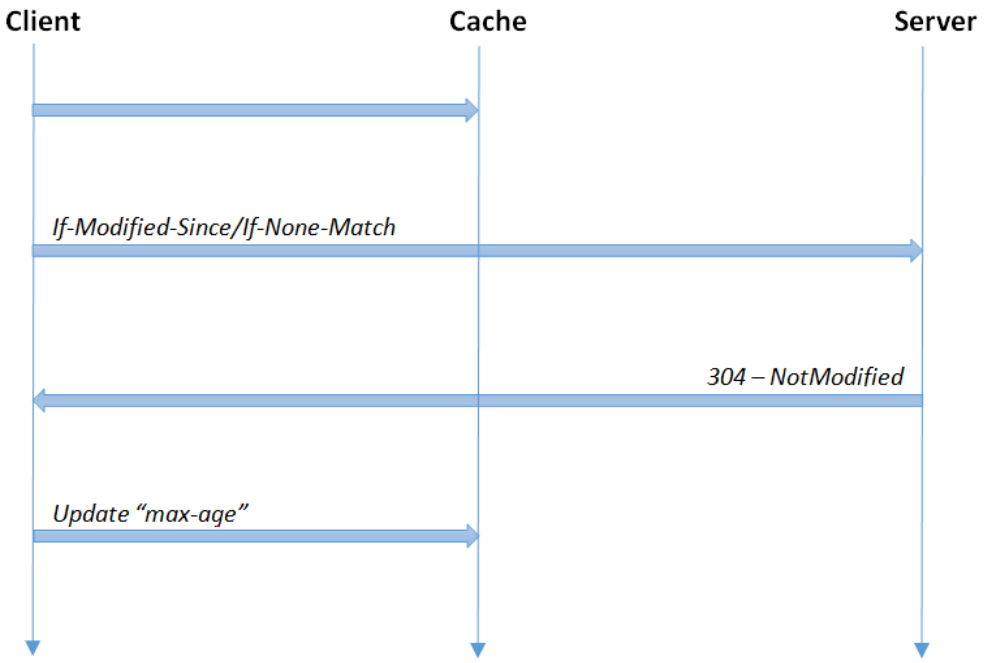
5.2.1 *Refresh* condizionale con risorsa in cache non più valida



Supponiamo che la risorsa richiesta sia stata precedentemente salvata in cache e che venga effettuato un *refresh* condizionale:

- Il client, al momento del *refresh*, verifica che la copia della risorsa sia presente in cache;
- Successivamente viene fatta una richiesta "*If-Modified-Since*" (oppure "*If-None-Match*" inviando anche l'*ETag* dell'informazione da confrontare con quello presente nel server) per verificare se la risorsa è cambiata;
- Dopo aver confrontato la data dell'ultima modifica (o l'*ETag*), il server risponde inviando la nuova informazione (*response = 200 OK*);
- Il client salva in cache la nuova risorsa sostituendo la precedente, ormai scaduta.

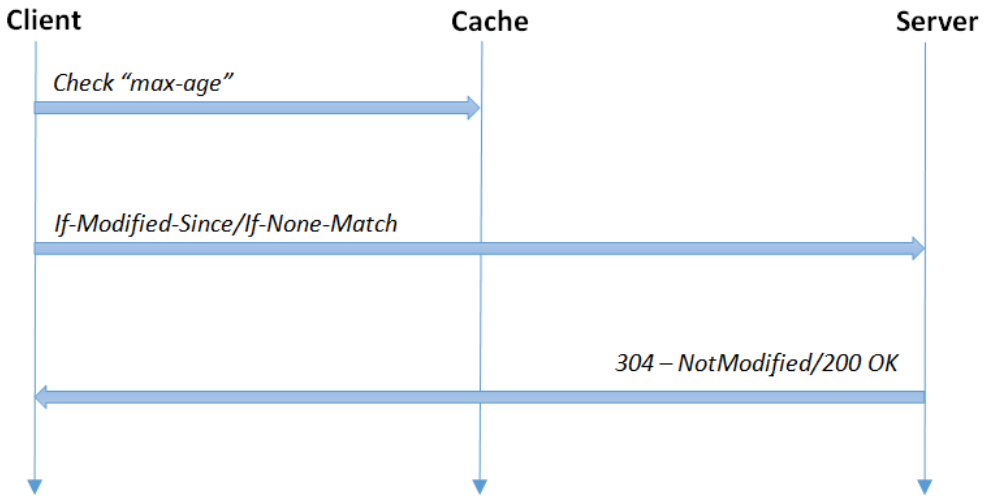
5.2.2 *Refresh* condizionale con risorsa in cache ancora valida



Supponiamo che la risorsa richiesta sia stata precedentemente salvata in cache e che venga effettuato un *refresh* condizionale:

- Il client, al momento del *refresh*, verifica che la copia della risorsa sia presente in cache;
- Successivamente viene fatta una richiesta "*If-Modified-Since*" (oppure "*If-None-Match*" inviando anche l'*ETag* dell'informazione da confrontare con quello presente nel server) per sapere se la risorsa in suo possesso è ancora valida (così è possibile evitare trasferimenti inutili);
- Dopo aver confrontato la data dell'ultima modifica (o l'*ETag*), constatando che la risorsa è ancora valida, risponde con un "*304 - NotModified*";
- Il client andrà dunque a ripristinare il "*max-age*" della copia già presente nella memoria cache.

5.2.3 Apertura di pagina tramite URL senza *refresh*



Supponiamo che la risorsa richiesta sia stata precedentemente salvata in cache e che il client richieda nuovamente tale URL (senza alcun *refresh*):

- Il client verifica che la copia della risorsa sia presente in cache e farà un controllo sulla sua scadenza;
- Se il "*max-age*" è scaduto, fa una richiesta "*If-Modified-Since*" (oppure "*If-None-Match*" inviando anche l'*ETag* dell'informazione) al server per sapere se la risorsa in suo possesso è ancora valida;
- Dopo aver confrontato la data dell'ultima modifica (o l'*ETag*), se non è più valida risponde con la nuova informazione aggiornata (*response = 200 OK*), altrimenti risponde con "*304 – NotModified*";
- Nel primo caso il client andrà a salvare in cache la nuova informazione, altrimenti si limiterà ad aggiornare il "*max-age*" della risorsa già in suo possesso.

Se il "*max-age*" della risorsa presente in cache non è scaduto, il client continuerà ad utilizzarla fino alla sua scadenza o fino ad un *refresh*.

Capitolo 6

Prove sperimentali

6.1 Tempi di caricamento

6.1.1 Risorsa *immutable*

Per testare i vantaggi del token *immutable* sono stati misurati i tempi di risposta da parte del server relativi ad una risorsa immutabile. Questi test sono stati effettuati attraverso la rete Wi-Fi dell'Università di Verona con il browser *Firefox48* (la versione precedente al rilascio del plug-in per riconoscere la nuova estensione) in modo da poter fare una media dei tempi che intercorrevano tra la richiesta "*If-Modified-Since*" da parte del client e la risposta "*304 – NotModified*" del server.

Il test si basa su 100 rilevazioni rappresentate in millisecondi.

La risorsa che si vuole analizzare è la scritta *facebook* della pagina principale del sito facebook.com, avente "*max-age*"=31536000 secondi (1 anno).
URL della risorsa: *https://www.facebook.com/rsrc.php/v3/y4/r/gf6iHxsw8zm.png*
Media dei tempi: $23,76ms$

Con l'implementazione del token *immutable* è possibile risparmiare completamente il tempo speso per inviare la richiesta di aggiornamento di questa risorsa.

6.1.2 Pagina con risorse *immutable*

Sono stati inoltre misurati i tempi di risposta relativi al caricamento dell'intera pagina principale di *Facebook*, avente risorse con "*max-age*"=31536000 secondi (1 anno). Il test si basa su 100 rilevazioni rappresentate in secondi, effettuate anch'esse su *Firefox48*.
URL della pagina: *https://www.facebook.com*
Media dei tempi: $1,97s$

Come si può notare attraversi i dati raccolti, ad ogni refresh condizionale si spende del tempo per verificare se la risorsa è ancora valida, nonostante si tratti di informazioni che resteranno tali per un periodo molto lungo (come indicato dal "*max-age*"). Implementando il token *immutable* è possibile risparmiare completamente il tempo speso per inviare molte richieste, abbassando notevolmente il tempo di caricamento dell'intera pagina.

Per dimostrare questo miglioramento sono stati effettuati altri 100 test sullo stesso URL, utilizzando però la versione aggiornata di *Firefox* (che implementa appunto l'estensione sul Cache-Control).
Media dei tempi: $1,77s$

6.2 Simulazione Client/Server

Per testare i vantaggi "lato client" e "lato server" sono stati riprodotti tramite linguaggio Java un Client ed un Server (in esecuzione su host diversi) in grado di interagire tra loro tramite la rete per simulare lo scambio di richieste per certe risorse.

Ogni richiesta generata dal Client viene incapsulata in una sorta di pacchetto Http contenente i seguenti campi:

- <*timestamp*>: tempo di arrivo della richiesta;
- <*objID*>: identificativo dell'oggetto;
- <*size*>: dimensione dell'oggetto (byte);
- <*dati_inviati*>: quantità di dati effettivamente serviti (byte);
- <*header_only*>: se viene settato a 1 indica che l'oggetto è stato richiesto con "*If-Modified-Since*" e la risposta dovrà essere "*304–NotModified*";
- <*is_fragment*>: se viene settato a 1 indica che il pacchetto iniziale è stato frammentato (campo ignorato durante i test).

La richiesta viene poi inviata al server che, dopo aver analizzato i campi del pacchetto, genererà la risposta corrispondente.

Nei primi test gli oggetti richiesti con "*header_ only*=1" genereranno una risposta del tipo "*304 – NotModified*". Successivamente è stato implementato il token *immutable*, evitando dunque di inviare tali richieste.

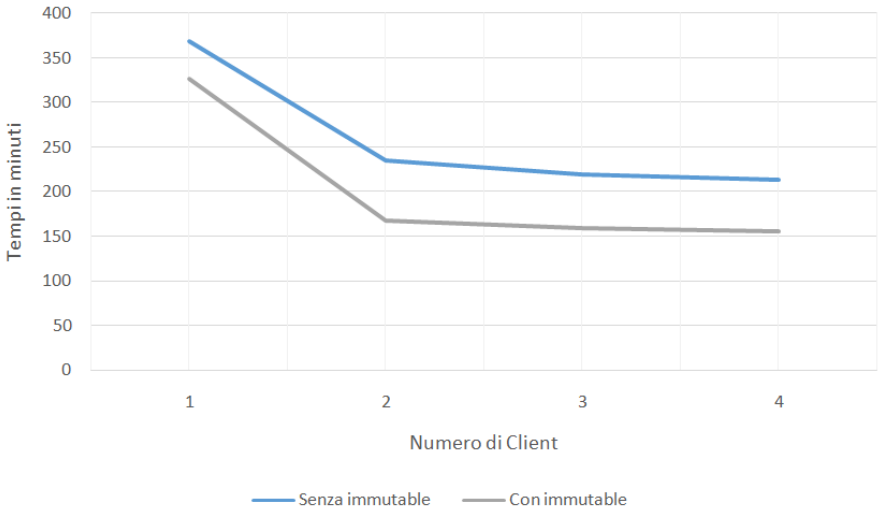
Entrambi gli host implementano l'utilizzo delle thread, rendendo possibile l'invio e l'elaborazione di più richieste in parallelo.

Le varie prove sono state avviate inizialmente su un file contenente 100 mila richieste e in seguito su altre 20 milioni. Al termine di ogni prova è stato salvato il rispettivo tempo di esecuzione, che ci ha permesso di constatare gli effettivi miglioramenti dovuti all'utilizzo dell'estensione sul Cache-Control.

Ulteriori ottimizzazioni in termini di tempo sono dovute anche all'utilizzo di più thread che però non devono superare il numero di "core" del processore che ospita il programma in esecuzione, in quanto andrebbero ad incrementare il numero di *Context Switch*, tempo che va a sommarsi all'effettiva elaborazione delle richieste.

I tempi possono essere ulteriormente falsati dalla presenza di altri programmi in esecuzione sullo stesso host, dal numero di "core" del processore, dall'eventuale traffico presente nella rete e dalla velocità di trasferimento di quest'ultima.

Di seguito è riportato il grafico che mette in relazione il numero di Client_Thread con i tempi di elaborazione delle 20 milioni di richieste (con e senza *immutable*).



Capitolo 7

Conclusioni

7.1 Lo scarso interesse degli altri browser

Attualmente "*immutable*" è stato implementato solo da *Firefox 49* poichè è ancora in fase di sviluppo e ad altri browser, come Chrome, ha suscitato poco interesse.
Ciò è dovuto alle imperfezioni ancora presenti:

- non è possibile ignorare "*immutable*" nel caso in cui scada il "*max-age*" della risorsa;
- richiedere un "*hard-refresh*" per avere risorse aggiornate non è una buona soluzione, essendo una funzionalità ancora sconosciuta alla gran parte degli utenti;
- necessità di indicare le pagine corrotte in cache (a causa della connessione) che non devono essere riutilizzate per evitare di inviare ai client dei contenuti non più validi.

7.2 L'alternativa proposta da *Chrome*

A partire dal 21 dicembre 2016 è stato reso disponibile un aggiornamento di *Chrome* che tiene conto del "*max-age*" delle risorse ed invia richieste di aggiornamento solo allo scadere di questo timeout. Prima della scadenza il client continuerà ad utilizzare solamente la copia salvata in cache. Viene comunque mantenuta la funzione di *hard-refresh* che permette di inviare richieste ai server ignorando le risorse già presenti in cache.

Proprio quest'ultima scelta degli sviluppatori di *Chrome* sembra chiudere quasi completamente le porte all'utilizzo di *immutable* in questo browser, avendo ideato una soluzione che sembra essere più *user-friendly* poiché non è obbligatorio l'*hard-refresh* per aggiornare i dati ma è sufficiente attendere la scadenza del "*max-age*".

In termini di efficienza vi è un miglioramento sostanziale rispetto agli schemi adottati in precedenza, avvicinandosi molto ai risultati ottenuti dall'implementazione delle risorse immutabili.