

R-HTTP: una breve panoramica

Andrea Burattin

27 aprile 2006

Sommario

Questo breve approfondimento, preparato per il corso di Tecnologie WEB, dell'Università degli Studi di Padova (Prof. Moreno Marzolla, A.A. 2005/2006), vuole essere nient'altro che una breve panoramica sulla tecnologia ideata da IBM per consentire connessioni HTTP affidabili. Dopo una breve introduzione alla situazione attuale, illustrerò la proposta di R-HTTP, proverò a confrontarla con altre tecnologie e ne esprimerò un'opinione personale.

Indice

1	HTTP ed il livello applicazione	1
1.1	Inaffidabilità di HTTP	2
2	La proposta di IBM: R-HTTP	3
2.1	HTTP affidabile	3
2.2	Funzionamento di R-HTTP	4
3	Confronti	5
3.1	HTTPS	5
3.2	Le <i>sessioni</i> , dei linguaggi <i>server side</i>	6
4	Impressioni finali	6

1 HTTP ed il livello applicazione

Il protocollo HTTP (HyperText Transfer Protocol) è un protocollo che risiede al livello più alto nella gerarchia dei protocolli di comunicazione. Questo rappresenta un lampante esempio di organizzazione della comunicazione tramite un "servente" che accetta le connessioni dei "clienti": questi ultimi richiedono delle risorse al server che soddisfa le loro richieste. Le richieste effettuate tramite HTTP riguardano, essenzialmente, pagine di tipo ipertestuale (come

dice l'acronimo stesso). La prima versione del protocollo, descritta nell'RFC 1945, fu implementata da Tim Berners Lee nel 1991 (RFC pubblicato nel 1996). Presto seguì la versione 1.1 (1997, RFC 2068) la cui principale differenza risiede nella possibilità di effettuare più richieste nella stessa connessione.

Come detto, HTTP 1.1 ha il vantaggio di poter esprimere più richieste nella stessa connessione quindi si può avere un *pipelining* di operazioni di richiesta, senza che siano ancora arrivate le risposte. Non è dunque necessaria una sincronia “uno a uno” fra domanda e risposta. È fondamentale, però, che le risposte siano recapitate con il medesimo ordine d'arrivo delle domande, seguendo una politica FCFS (first come first served).

1.1 Inaffidabilità di HTTP

Uno dei principali problemi di HTTP, che rendono sensate ricerche di varie soluzioni, è il fatto che HTTP, a differenza di altri protocolli dello stesso livello, quali, ad esempio, FTP, è un protocollo *stateless*, ovvero privo di stati. Ciò significa che un server che deve rispondere ad un insieme di richieste di un client non è tenuto a mantenere alcuna informazione su quest'ultimo fra una connessione ed un'altra. Di conseguenza, ogni connessione è completamente indipendente e ciò richiede la creazione, più volte, del medesimo contesto per soddisfare la domanda, con conseguente spreco di potenza di calcolo. Se da un lato ciò può rappresentare un vantaggio, in una navigazione che effettua vari *hop* fra siti WEB (collegamenti a server diversi, quindi con connessioni diverse), poichè le connessioni decadono immediatamente; dall'altro, l'indipendenza delle connessioni rappresenta un problema nel momento in cui si voglia interagire con gli utenti attraverso più pagine distinte. Tutto ciò è ben contornato da una quantità di possibili scenari di fallimento del caricamento di una o più pagine (o più in generale, di problemi di connessione) incredibilmente alto, che rende necessario lo studio del problema e delle relative soluzioni.

Ipotizziamo il caso di una attività di e-commerce che deve ricevere gli ordini attraverso il sito WEB dell'azienda. Se un cliente decide di effettuare l'ordine, dopo essersi connesso al sito ed aver individuato il prodotto, vorrebbe che, cliccando su un ipotetico pulsante “INVIA ORDINE”, la situazione si presentasse in una di queste due stati:

- l'ordine è stato correttamente ricevuto dal negoziante;
- l'ordine non è stato ricevuto.

In realtà ci possono essere moltissime altre situazioni a dir poco spiacevoli che potrebbero creare alcuni problemi (Figura 1), ad esempio: l'ordine non è stato recapitato e nemmeno la notifica di errore, da parte dell'azienda, è giunta al cliente. In questo caso, come si dovrebbe comportare l'utente?

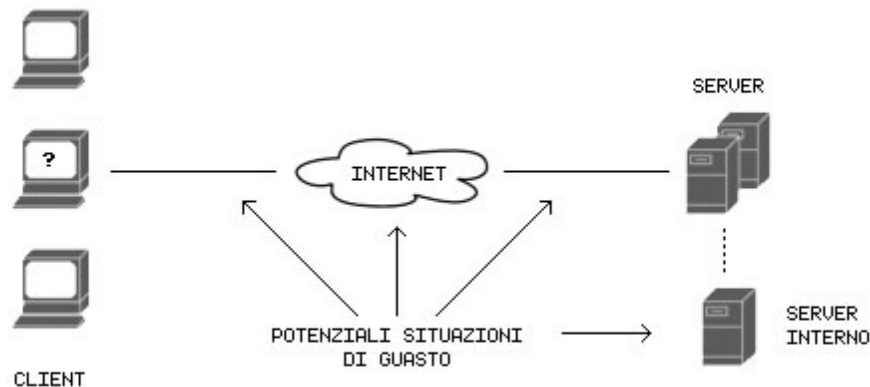


Figura 1: Situazioni di inaffidabilità, il client non sa' come comportarsi

Un'altra possibile situazione potrebbe essere quella in cui il negozio riceve due volte lo stesso ordine a causa di ritardi nella rete e solo una conferma di ordine ricevuto giunge all'utente (che crede di aver comprato un prodotto ma se ne vedrà recapitare due!).

I problemi, come detto, non scarseggiano, soprattutto per quanto riguarda la situazione dell'affidabilità della connessione durante una medesima sessione di lavoro. Se questa situazione è già stata risolta (sin dall'origine) da altri vari protocolli, altrettanto non si può dire né per HTTP 1.0, né per HTTP 1.1, anche perchè, come già detto, questa situazione ha vari vantaggi dal lato server.

2 La proposta di IBM: R-HTTP

Il problema dell'affidabilità di HTTP è stato colto anche alla IBM ed è qui che è stato risolto, a livello di protocolli. La soluzione, basata sull'invio di comandi POST HTTP, è stata chiamata R-HTTP o *reliable HTTP*.

2.1 HTTP affidabile

Come detto, l'IBM ha sviluppato R-HTTP, per ora solamente a livello teorico e di specifiche, con l'intento di rendere la trasmissione dei messaggi affidabile, ovvero senza il problema delle ambiguità e dei rischi interconnessi con il semplice HTTP. In particolare, è stato sviluppato un nuovo *livello* che si poggia ad HTTP. Effettivamente R-HTTP, allo stato attuale, è rappresentato solamente dalle specifiche di come sia possibile implementare questo protocollo.

L'idea di base consiste nel rispettare la proprietà di correttezza, la quale si compone di:

1. **liveness**: *at least once*, vale a dire che tutti i messaggi inviati, prima o poi, giungeranno a destinazione;
2. **safety**: *at most once*, ovvero che i messaggi sono stati sempre ricevuti nell'ordine con cui sono stati inviati ed in ugual numero.

Dal soddisfacimento delle due condizioni, arriviamo a capire che il messaggio deve essere ricevuto esattamente una volta.

2.2 Funzionamento di R-HTTP

Come detto, il punto essenziale sta nella creazione di un nuovo livello affidabile che venga costruito sopra ad HTTP. Per “costruire” questo livello, si possono seguire due strade differenti:

1. la prima soluzione consiste nel realizzare tale protocollo a livello *applicazione*, come fatto dai protocolli di comunicazione per i WEB services che estendono SOAP. Il vantaggio di questo approccio è chiaramente quello di poter essere utilizzato indipendentemente dal protocollo che si occupa del trasporto. Tuttavia ci sono alcune complicazioni per quanto riguarda i processi di basso livello (ad esempio quando devono trattare il reinvio dei messaggi), oltre, ovviamente, alla non possibilità di ottimizzazione degli stessi.
2. L'altra possibilità di locazione del protocollo risiede al livello *trasporto*. Questa soluzione consente una ottimizzazione dei messaggi. Inoltre tale approccio consente di adottare questo protocollo anche ad altri che già si appoggiano ad HTTP (SOAP su tutti) senza dover modificare nulla.

R-HTTP è costruito sopra ad HTTP 1.1, ereditandone tutte le caratteristiche (connessioni sicure con SSL, supporto per firewall e proxy). Un ulteriore vantaggio di basarsi su HTTP è che tutte le aziende che già hanno a disposizione un loro server WEB che supporta traffico HTTP potrà benissimo ospitare connessioni R-HTTP.

Il funzionamento di R-HTTP inizia con l'invio, da parte dell'agente lato client di un comando HTTP POST in cui include un suo identificativo e, nello stesso, il comando R-HTTP da eseguire ed un insieme di messaggi. Successivamente, il server risponderà con un messaggio nel cui payload sono contenute informazioni sullo stato e, se richiesta, la lista dei comandi assegnati al client. Ciascuna serie di informazioni in uscita è associata ad un identificatore, simile ad un *nonce*. Per rendere effettivamente possibile che la comunicazione avvenga *at most once* si rende necessario l'uso di supporti

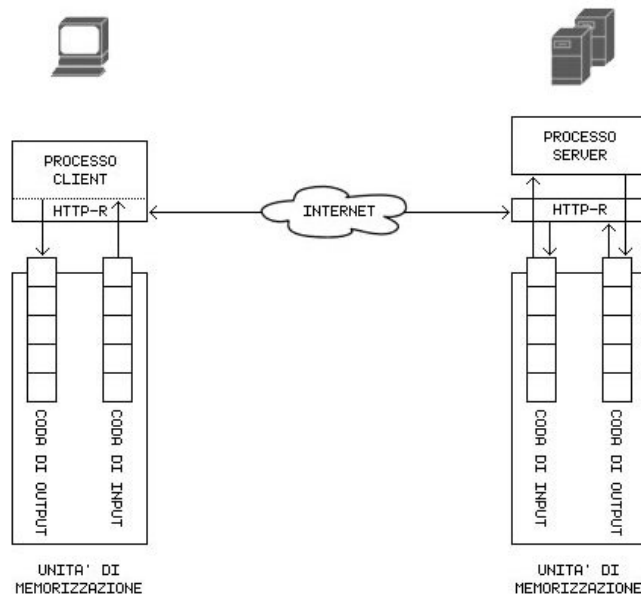


Figura 2: Schematizzazione della soluzione proposta da IBM

persistenti (disco, DBMS, ...) sia dal lato server, che dal lato client. In dettaglio, i due agenti atti alla comunicazione dovranno conservare una doppia lista di messaggi (in ingresso ed in uscita) e dei relativi stati.

In sostanza, R-HTTP definisce come i metadati ed i messaggi delle varie applicazioni possano essere incapsulati nel payload di richieste e di risposte HTTP. Concetto fondamentale ed alla base di tutta l'idea è il creare un insieme di regole che rendano possibile assicurare che un messaggio sia stato recapitato o, in caso contrario, ne sia notificato il mancato arrivo.

3 Confronti

Il ruolo svolto da R-HTTP può essere, a mio avviso, facilmente confrontato con altri due sistemi già esistenti, i quali, seppur non risolvono pienamente i ruoli del protocollo IBM, rappresentano delle soluzioni parziali.

3.1 HTTPS

HTTPS è un protocollo, nato nel 2000, il quale ha il principale scopo di creare delle connessioni protette, tramite SSL. Si può paragonare ad R-HTTP in quanto il protocollo utilizzato per il trasporto include un controllo dell'integrità del messaggio tramite un checksum (MAC: Message Authentication Code) creato tramite funzioni hash (MD5, SHA-1, ...). Ciò consente di verificare che i dati non siano alterati durante il trasporto e quindi, in senso lato, che un pacchetto non sia ricevuto più volte.

Avremmo quindi soddisfatto ad una delle due proprietà richieste per avere correttezza, in particolare la *safety*. Con solo questa proprietà non riusciamo a fare molto, poichè manca la garanzia di ricezione del messaggio *almeno* una volta. Inoltre, è doveroso ricordare che HTTPS è nato con l'unico scopo di offrire delle connessioni sicure per quanto riguarda la protezione del contenuto, e non la garanzia di recapito.

3.2 Le sessioni, dei linguaggi *server side*

Come già visto a lezione, ci sono vari meccanismi che consentono di conservare dati inviati dal server ai vari client che lo contattano (i.e. cookie), e ciò, può essere utilizzato per simulare varie operazioni (seppur ad un livello diverso) che sono comuni ad R-HTTP, il quale deve memorizzare informazioni sugli identificativi dei vari messaggi ed i relativi stati. Tale sistema è stato catturato da alcuni linguaggi orientati al WEB che hanno creato dei meccanismi che consentono di rendere “un po' meno stateless” HTTP. Questi sistemi, detti *sessioni*, consistono nell'invio, da parte del server, di informazioni che identificano univocamente ciascun client (SID: Session Identifier). Questo, dopo aver memorizzato la sua identità in un cookie, la ripresenta ad ogni richiesta fatta al server, il quale la può utilizzare per memorizzare informazioni varie e, analizzando, ad esempio, i timestamp delle varie richieste, capire se un utente è sempre nella medesima sessione di lavoro. Ovviamente questa tecnica non è un'alternativa assoluta ad R-HTTP, ma può essere utilizzata per risolvere già da subito alcuni degli stessi problemi che mira a risolvere R-HTTP.

Il confronto potrebbe sembrare un po' azzardato, ma se consideriamo alcuni esempi, possiamo notare che vari problemi possono essere ugualmente risolti anche senza ricorrere ad R-HTTP. Tornando al solito esempio dell'acquisto di materiale da un negozio sul WEB, vediamo chiaramente che il problema, con il semplice uso delle sessioni, non può essere risolto alla radice; tuttavia si posso costruire dei sistemi che rappresentano dei semplici *workaround* al problema. In effetti, un esempio potrebbe essere quello di impostare le sessioni ad una durata piuttosto breve, cosicché eventuali messaggi, arrivati con troppo ritardo, vengono scartati e non considerati, e, conseguentemente, l'ordine viene correttamente fatto una sola volta.

4 Impressioni finali

Anche se alcuni problemi pratici possono essere svincolati utilizzando, ad esempio, i cookie (o sistemi costruiti al di sopra di essi), è chiaro che la soluzione al problema di partenza (*un messaggio o è stato recapitato correttamente o se ne notifica il mancato arrivo al mittente*) non può essere risolto in maniera efficiente con alcuna soluzione che si appoggi al livello di HTTP. Gli stessi documenti rilasciati dagli ideatori di R-HTTP, alla IBM,

affermano che, appoggiandosi direttamente ad HTTP non è affatto possibile garantire alcun tipo di efficienza. Se si vuole dunque risolvere tale problema è necessario staccarsi da HTTP e passare ad un altro protocollo, il quale, dal mio punto di vista, dovrebbe risiedere ad un livello sottostante ad HTTP. La ragione che mi spinge a dire che la corretta collocazione di tale protocollo dovrebbe essere al livello trasporto è che il problema che abbiamo di fronte è un problema di *garanzia del trasporto* dell'informazione e non propriamente di "applicazione". Di conseguenza approverei più entusiasticamente un'implementazione di R-HTTP alla "seconda maniera". Inoltre, tale implementazione sarebbe perfettamente utilizzabile da applicazioni già esistenti, senza alcuna modifica di quest'ultime, dato che, ai loro occhi, la trasmissione è un aspetto assolutamente trasparente e del quale non si devono occupare.

Riferimenti bibliografici

- [1] Stephen Todd, Francis Parr, Michael Conner: *A Primer for HTTPR*, (2005)
- [2] Andrew Banks, Jim Challenger, Paul Clarke, Doug Davis, Richard P. King, Karen Witting, Andrew Donoho, Time Holloway, John Ibbotson, Stephen Todd: *HTTPR Specification* (2002)
- [3] Moreno Marzolla: Slides del corso di "Tecnologie WEB", Università degli Studi di Padova, (2006)
- [4] Paolo Romano: *Fault-tolerant Web systems* (2002) (idea della rappresentazione grafica di R-HTTP)