Secure Compression and Pattern Matching Based on Burrows-Wheeler Transform

Raffaele Ceruso Giovanni Leo

November 25, 2018

Protocollo

Il protocollo prevede le seguenti fasi:
(1) **Inizializzazione.** Il client genera le strutture dati *c* e *occ*. *occ* è generata nel seguente modo.

- *superblock*:
  scegli un primo p_1 leggermente più grande di n = L2, un numero diverso da zero β_1 e scegli un generatore g_1 di $Z^*_{p_1}$. Il client mantiene $g_1\beta_1$ segreti. Per *superblock* i, memorizziamo quei dati nel $(g_1^i\beta_1)^{th} mod\ p_1$ (omettiamo $mod\ p_1$ nella parte restante) della tabella. Per le entries vuote inseriamo alcuni numeri casuali.

Protocollo

Il protocollo prevede le seguenti fasi:
(1) **Inizializzazione.** Il client genera le strutture dati c e occ. occ è generata nel seguente modo.

- *block*:
  scegli un primo p_2 che è poco più grande di n/L^2, un numero diverso da zero β_2 e scegli un generatore g_2 di $Z^*_{p_2}$. Il client mantiene $g_2\beta_2$ segreti. Per *block* i, memorizziamo quei dati nel $(g_2^i\beta_2)^{th} mod\ p_2$ (omettiamo $mod\ p_2$ nella parte restante) della tabella. Per le entries vuote inseriamo alcuni numeri casuali.

Protocollo

Il protocollo prevede le seguenti fasi:
(1) **Inizializzazione.** Il client genera le strutture dati c e occ. occ è generata nel seguente modo.

- $Hash\_value$:
  Invece di memorizzare immagini di MTF tabelle, memorizziamo il valore hash di $(i-1)^{th}$ block per il blocco i^{th}. Il valore hash di $(i-1)^{th}$ block è memorizzato nella $(g_2^i \beta_2)^{th}$ entry della tabella $Hash_value$. Per le entries vuote inseriamo alcuni numeri casuali.

Protocollo

Il protocollo prevede le seguenti fasi:
(1) **Inizializzazione.** Il client genera le strutture dati c e occ. occ è generata nel seguente modo.

- *Block_inner*:
  La riga $Block\_inner(mtf[i], BZ_i, e, h - L * i)$ viene spostato su $Block\_inner(Hash\_value[g_2^i \beta_2], BZ_{g^i \beta_2}, e, \tau(key, h - L * i))$ dove $i = \lfloor h/L \rfloor$ e $\tau : Z_L \rightarrow Z_L$ è una funzione di permutazione casuale.

Protocollo

Il protocollo prevede le seguenti fasi:
(1) **Inizializzazione.** Pertanto, per calcolare $occ(e, h)$, il client deve inviare un vettore di posizione
$pv(h) = (g_1^{h/L^2}\beta_1, g_2^{h/L}\beta_2, \tau(key, h - [h/L] * L))$ al server.
Pertanto l'algoritmo è Algo = sBWT + bMTF + RLE + PC +P dove P è la permutazione. Dopo aver criptato i dati in *occ* con AHE il client li invia al server il quale mantiene c. Successivamente sceglie un numero adeguato R come round di comunicazione.

Protocollo

Il protocollo prevede le seguenti fasi:
(2) **Client:** invia un vettore posizione *pos* e legge il vettore V in accordo di $P[i-1]$

Protocollo

Il protocollo prevede le seguenti fasi:
(3) **Server:** per ogni carattere dell'alfabeto e del y^{th} vettore in *pos*, computa $B_y[\zeta]$ e ritorna $a_{y\,y\in[1,4]}$

Protocollo

Il protocollo prevede le seguenti fasi:

(4) **Client:** trova l'aspettato a_{i1}, $a_{i2} \in a_{y\,y\in[1,4]}$ e computa

- $occ(P[i-1], begin-1)$
- $occ(P[i-1], end)$
- $begin$
- end

  Se $begin \leq end$ e $i \geq 2$ vai allo step 2, altrimenti invia pos al server.
  Se il round di comunicazione è R vai al prossimo step.

Protocollo

(5) **Client:** se $end < begin$ il pattern non è stato trovato, altrimenti l'occorrenza del pattern è $end - begin + 1$