Secure Compression and Pattern Matching Based on Burrows-Wheeler Transform

Raffaele Ceruso Giovanni Leo

November 24, 2018

Idea

- Le “compressed data structure” permettono di creare una indicizzazione di grandi dataset in maniera efficiente. Tali strutture dati dovranno essere sicuramente memorizzate in qualche server di terze parti come ad esempio il cloud, questo però porta sicuramente problemi di privacy.
- L’ idea quindi é quella di costruire una variante di tali strutture più sicura, basata sulla trasformata di Burrows-Wheeler, e che sia in grado anche di eseguire il pattern matching.

Introduzione - 1

- La compressione dati non solo riduce lo spazio occupato dai file ma serve anche a migliorare la velocità di trasmissione in alcuni protocolli.
- Essendo in era in cui i dati sono diventati veramente importanti di conseguenza anche la compressione di tali dati è diventata sempre più necessaria.
- Gli autori del paper si sono focalizzati sull'utilizzo di algoritmi di compressione, i quali non solo supportano una compressione di tipo lossless ma che garantiscono anche sicurezza.
- Il pattern matching è una operazione fondamentale nel processing delle stringhe che permette di trovare tutte le occorrenze di un dato pattern in un dato testo.

Introduzione - 2

- Il pattern matching si vuole applicare anche ai file compressi
- Una possibile strategia potrebbe essere quella di decomprimere il file e cercare sul file decompresso ma tale strategia è poco efficiente.
- Un approccio migliore sarebbe quello di ricercare direttamente sul file compresso
- Per fare ciò abbiamo bisogno di indici di dati compressi i quali vengono salvati in server di terze parti causando problemi di privacy.
- Una semplice soluzione potrebbe essere quella di di comprimere e poi cifrare i dati ma tale soluzione è stata dimostrata non tanto sicura.
- Una altra soluzione potrebbe essere quella di integrare compressione e cifratura in un unico passo ma anche questa soluzione presenta problemi di sicurezza.

Introduzione - 3

- La soluzione proposta dagli autori si basa sulla trasformata di Burrows-Wheeler. Oltre la compressione tale trasformata può essere utilizzata anche per eseguire la ricerca. Se consideriamo due tabelle che forniscono un certo tipo di informazioni come per esempio la frequenza dei simboli e le posizioni, la trasformata permette di estrarre le sottostringhe che matchano i pattern in modo semplice.
- Gli autori in questo paper forniscono un "secure compression algorithm" e un "secure compressed pattern matching", entrambi basata sulla BWT. Inoltre viene uno schema di cifratura omomorfico additivo per proteggere gli indici dei dati compressi e per sfruttare le potenzialità del cloud. Per ottenere i dati dal server viene utilizzato il "Private Information Retrival Read" (PIR_Read).

Preliminari - 1 - BWT and compression

- BWT riorganizza una stringa di caratteri in una serie di caratteri simili quindi la si può vedere come un algoritmo che prepara i dati per usarli con delle tecniche di compressione dati.
- Adesso consideriamo una tale trasformazione in maniera generale divisa in tre passi:
  1. Aggiungiamo un carattere speciale \$, il quale non è nell'alfabeto Σ, alla fine della stringa T e assumiamo che \$ è il più piccolo carattere in Σ considerando un ordine lessicografico.
  2. Costruiamo una matrice M le quali righe sono degli shift ciclici di T.
  3. Ordiniamo le righe della matrice M in ordine lessicografico e diamo come output l'ultima colonna della matrice M, il quale è il risultato della BWT.

Preliminari - 1 - BWT and compression (Esempio BWT)

order	first character	medial strings	last character
1	$	mississipp	i
2	i	$mississip	p
3	i	ppi$missis	s
4	i	ssippi$mis	s
5	i	ississippi$	m
6	m	ississippi	$
7	p	i$mississi	p
8	p	pi$mississ	i
9	s	ippi$missi	s
10	s	issippi$mi	s
11	s	sippi$miss	i
12	s	sissippi$m	i

Figure: Consideriamo l'esempio per la stringa "mississippi$"

Preliminari - 1 - BWT and compression

- BWT è solo il primo passo di un algoritmo di compressione, di solito i passi sono: *BWT+MTF+RLE+PC.*
- In maniera generale possiamo dire che la BWT cerca di raccogliere gli stessi caratteri insieme; MTF vuole rimpiazzare questi caratteri successivi uguali con degli zero; RLE vuole codificare questi zero con meno bit e infine PC vuole codificare il risultato della fase precedente in forma binaria.

Preliminari - 1 - BWT and compression

- BWT è solo il primo passo di un algoritmo di compressione, di solito i passi sono: *BWT+***MTF***+RLE+PC.*
- **MTF**(Move to front): Sia T il risultato della BWT, *T=BWT(T)*. C'è una *MTF_table* contenente tutti i caratteri dell'alfabeto. La tabella mappa i caratteri alla loro relativa posizione nella tabella(per esempio il primo carattere è mappato a "0"). Dopo aver codificato il primo carattere esso verrà spostato in prima posizione nella *MTF_Table* quindi tale carattere verrà mappato a "0". Sia *Z =MTF(T)* ovvero la stringa codificata dopo aver applicato la MTF a *T*.

Preliminari - 1 - BWT and compression

- BWT è solo il primo passo di un algoritmo di compressione, di solito i passi sono: *BWT+***MTF***+RLE+PC.*
- **Esempio MTF:**

Iteration	Sequence	List
bananaaa	1	(abcdefghijklmnopqrstuvwxyz)
b**a**nanaaa	1,1	(bacdefghijklmnopqrstuvwxyz)
ba**n**anaaa	1,1,13	(abcdefghijklmnopqrstuvwxyz)
ban**a**naaa	1,1,13,1	(nabcdefghijklmopqrstuvwxyz)
bana**n**aaa	1,1,13,1,1	(anbcdefghijklmopqrstuvwxyz)
banan**a**aa	1,1,13,1,1,1	(nabcdefghijklmopqrstuvwxyz)
bananaa**a**	1,1,13,1,1,1,0	(anbcdefghijklmopqrstuvwxyz)
bananaa**a**	1,1,13,1,1,1,0,0	(anbcdefghijklmopqrstuvwxyz)
Final	1,1,13,1,1,1,0,0	(anbcdefghijklmopqrstuvwxyz)

Preliminari - 1 - BWT and compression

- BWT è solo il primo passo di un algoritmo di compressione, di solito i passi sono: *BWT+MTF+***RLE***+PC.*
- **RLE**(Run Length Encoding): Possiamo applicare RLE per ogni run(sequenza di caratteri tutti uguali) di zero in Z. Per essere più precisi possiamo rimpiazzare m successivi zeri con il valore binario del numero $(m+1)$, scartando il bit più significativo, in ordine inverso. Inoltre introduciamo altri due caratteri **a** e **b** per fare la codifica degli zero. Per esempio 5 zeri vengono codificati come ab. Inoltre il risultato è su di un alfabeto $\{$**a**,**b**,1,...,$|\Sigma - 1|\}$ dove $|\Sigma|$ indica il numero di caratteri presenti nell'alfabeto.

Preliminari - 1 - BWT and compression

- BWT è solo il primo passo di un algoritmo di compressione, di solito i passi sono: *BWT+MTF+RLE+***PC**.
- **PC**(Variable Length prefix code): Possiamo applicare Variable Length prefix code a questi simboli dell'alfabeto $a, b, 1, ..., |\Sigma| - 1$. Codifichiamo a con 10 e b con 11. Per gli altri simboli i usiamo $1 + 2 * \lfloor log(i+1) \rfloor$ dove $\lfloor log(i+1) \rfloor$ bit sono 0 seguiti dal numero (i+1) nella forma binaria costituita da $1 + \lfloor log(i+1) \rfloor$ bit, per esempio 00110 per il simbolo "5". Quindi il risultato finale è sull'alfabeto {0,1} e denotiamo il risultato finale $BZ = PC(RLE(Z))$.

Preliminari - 2 - Strutture dati ausiliarie e backward pattern matching

- Per supportare il backward pattern matching abbiamo bisogno di due strutture dati ausiliarie:
  1. **c(e)**:la quale è una tabella che memorizza l'indice minimo di una stringa che inizia con il carattere e nella matrice ordinata M e denotiamo il prossimo simbolo il quale è un po più grande del simbolo e in ordine lessicografico, con $e+1$. Quindi $c[e+1]- 1$ restituisce la posizione finale di e.
  2. **occ(e,h)**: la quale memorizza le occorrenze di un carattere e dalla prima fino ad una certa posizione h nel risultato della BWT.

Preliminari - 2 - Strutture dati ausiliarie e backward pattern matching

- **Lemma 1**: Data una stringa $T_1 = t_1 ... t_{n-1} t_n$ la quale è la h-esima stringa nella matrice ordinata M, sia $T_2 = t_n t_1 ... t_{n-1}$ la k-esima stringa nella matrice ordinata M, dove $k = c[t_n] + occ(t_n, h-1)$.
- La correttezza di tale lemma é data dal fatto che la matrice M é ordinata
- Supponiamo di conoscere la stringa "sissippi$mis" che si trova nella posizione 10 e vogliamo sapere la posizione della stringa "ssissippi$mi" quindi si ha che $c(s) + occ(s,9) = 9+3 = 12$ e quindi sapremo che "ssissippi$mi" si trova in posizione 12.

order	first character	medial strings	last character
1	$	mississipp	i
2	i	$mississip	p
3	i	ppi$missis	s
4	i	ssippi$mis	s
5	i	ississippi$	m
6	m	ississippi	$
7	p	i$mississi	p
8	p	pi$mississ	i
9	s	ippi$missi	s
10	s	issippi$mi	s
11	s	sippi$miss	i
12	s	sissippi$m	i

Preliminari - 2 - Strutture dati ausiliarie e backward pattern matching

- **Lemma 2**: Dato un pattern $P = p_1...p_n$ e un range (begin,end) nella matrice M dove queste stringhe iniziano con un subpattern $p_{i+1}...p_n$ ($i \geq 1$), allora il range di $p_i...p_n$ è *(begin',end')*, dove $begin' = c[p_i] + occ[p_i, begin - 1]$ e $end' = c[p_i] + occ[p_i, end] - 1$
- Il risultato T della BWT può essere suddiviso in blocchi(block) di L caratteri, inoltre L blocchi vengono chiamati superblocchi(superblock). *superblock(e,h)* restituisce l'occorrenza del simbolo e nei primi $\lfloor h/L^2 \rfloor$ superblocchi. *block(e,h)* restituisce le occorrenza di e a partire dall'ultimo blocco fino al $\lfloor h/L \rfloor$ - esimo blocco. Per l'occorrenza in un blocco viene memorizzata in $block\_inner(mtf(i), BZ_i, e, h - i * L)$, dove BZ_i è l'i-esimo blocco compresso e mtf(i) memorizza lo stato della tabella MTF all'inizio della codifica dell'i-esimo blocco.

Costruzione- 3 - Compressione

- Hanno provato a proporre un algoritmo di compressione che andava a combinare la sBWT con una variante del MTF dove la MTF table veniva inizializzata con lo stesso ordine lessicografico della sBWT, ma questo ha portato a problemi di sicurezza e quindi tale approccio è stato abbandonato.
- Per risolvere tale problema gli autori hanno cambiato, in maniera random, la MTF table per ogni L caratteri(ovvero ogni blocco). Per ridurre la archiviazione delle chiavi per la funzione pseudo casuale, viene utilizzato come input per tale funzione il valore hash dell'ultimo blocco compresso.

Costruzione- 3 - Compressione

- HL'algoritmo può essere visto in maniera generale come *Algo=sBWT+bMTF+RLE+PC.*
- Un utente sceglie una funzione funzione hash f e una pseudo casuale funzione di permutazione *Perm* basata su chiave. Poi sceglie un numero random per la funzione di permutazione come chiave(*key*) privata. Quindi quando viene inserita un stringa l'utente la processa prima attraverso la sBWT e poi attraverso la bMTF.

Costruzione- 3 - Compressione

- HL'algoritmo può essere visto in maniera generale come *Algo*=**sBWT**+*bMTF*+*RLE*+*PC*.
- **sBWT**(Scrambling BWT): Sia T la stringa originale e T = sBWT(T) il risultato della Scrambling BWT su T
  - Scegliamo un numero random r e dopo calcoliamo $Perm(key, r) \rightarrow \xi$, dove ξ denota l'ordine lessicografico segreto.
  - Aggiungiamo un carattere parziale \$ $\notin \Sigma$ alla fine di T e assumiamo che \$ è il più piccolo dei caratteri dell'alfabeto Σ nell'ordine lessicografico segreto ξ. Per semplicità chiamiamo la nuova stringa ottenuta T.
  - Costruiamo una matrice M le cui righe sono degli shift ciclici di di T.
  - Ordiniamo le righe della matrice M utilizzando l'ordine lessicografico segreto ξ e l'ultima colonna di M è T

Costruzione- 3 - Compressione - bMTF - 1

- L'algoritmo può essere visto in maniera generale come Algo=sBWT+**bMTF**+RLE+PC
- **bMTF**(blocky MTF): Raggruppiamo T in blocchi di L caratteri. Per ogni blocco scegliamo una permutazione dell'alfabeto prima a caso e dopo seguendo i passi della MTF per fare la codifica di qualsiasi carattere presente in ogni blocco. Qui verrà chiamata la funzione di permutazione pseudocasuale *Perm* con due parametri la chiave e un nounce. Il nounce è un vettore IV per il primo blocco ed è il valore hash dell'ultimo blocco codificato in altri casi.

Costruzione- 3 - Compressione - bMTF - 2

- Procedimento schematizzato
  - Costruisci blocchi di L caratteri a partire dalla stringa T
  - Per il blocco i vine prima generata una MTF_table:$Perm(key, IV \text{ or } f(block_{(i-1)})) \rightarrow bMTF\_table_i$, dove $bMTF\_table_i$ è una permutazione di tutti i caratteri dell'alfabeto.
  - Segui gli step generali della MTF

Costruzione- 3 - Compressione

- L'algoritmo può essere visto in maniera generale come Algo=sBWT+bMTF+**RLE**+**PC**
- I passi **RLE** e **PC** devono semplicemente andare a codificare utilizzando meno bit possibili.

Costruzione- 3 - Pattern Matching - 1

- Utilizzando strutture dati ausiliarie si ha la perdita di informazioni riguardanti frequenza e ordine lessicografico il che è un problema per la sicurezza.
- Per risolvere tale problema gli autori hanno pensato di cifrare tali strutture e di proporre un protocollo per eseguire il pattern matching. In maniera sintetica servono tre passi:
  1. Permutare la occ table al fine di nascondere le informazioni
  2. Adottare un metodi di ottenimento delle informazioni privato al fine di ottenere le entità dalla *occ* table al fine di ottenere un risultato conforme.
  3. Falsificare alcune richieste al fine di nascondere dati come *begin* e *end*