Monitorare la rete con PF_RING e Count Min Sketches

Aldi Sulova

L'obbiettivo del progetto è studiare gli "sketches".

Gli sketches sono particolari strutture dati introdotte nell'area degli algoritmi di streaming.

Gli algoritmi di streaming sono algoritmi per processare informazione, che si presenta in input come una successione di dati. Generalmente sono una rappresentazione di funzioni statistiche, che possono risultare utili quando : 1) si vuole mantenere una visione real-time dei dati, anche se vengono salvati in grandi database per elaborarli successivamente (es. parole chiavi sulle ricerche in internet); 2) l'informazione si presenta con alta velocità e quantità. Possibilità di avere una visione dei dati anche con limitate risorse spazio e tempo, necessarie per la loro elaborazione(es. monitoraggio di traffico di rete).

Nel secondo caso è richiesto che il contenuto dell'elemento i-esimo dello stream sia processato in pochi passi (tipicamente uno), utilizzando poca memoria veloce. L'obbiettivo di questi algoritmi è di calcolare funzioni utilizzando meno risorse di calcolo rispetto ad un algoritmo che calcola la stessa funzione in modo preciso.

Questi vincoli fanno che l'algoritmo produce un risultato approssimato basato su un sommario dei dati (sketch) che si presentano in input.

L'approccio è diverso da NetFlow ed sFlow, è più simile ad SNMP Monitoring. Non c'è trasferimento del flusso ad un collezionatore. L'analisi della rete è effettuata direttamente sul router utilizzando tutti i pacchetti che transitano. Sarebbe più corretto dire tutti quelli che riesce a elaborare, che in teoria sono in quantità maggiore di NetFlow e naturalmente anche di sFlow.

NOTA: Gli algortmi di streaming sono divisi in tre grandi categorie, a seconda delle strutture dati utilizzate: a) counter based algorithms , b) quantiles based algorithms, c) sketches based algorithms.

Tipiche funzioni calcolate da questo approccio sono :

Heavy hitters, l'informazione che si ripete con maggiora frequenza,

Entropy, la distribuzione della informazione nel suo range.

1.Il modello

La maggior parte della letteratura definisce lo streaming come "il calcolo di particolari funzioni statistiche sulle frequenze delle distribuzioni degli elementi che compongono lo stream" (es. vedi il paragrafo 2).

Si considera un vettore **a** di dimensione **n**:

$a = (a_1, \ldots, a_n)$

dove *n* è il range dei dati in ingresso. Lo stato del vettore nell' istante del tempo *t* è:

$a(t) = (a_1(t), \ldots, a_n(t))$.

Inizialmente a è il vettore 0, quindi $a_i(0)$ è uguale a 0 per ogni *i* in *n*.

Gli aggiornamenti sul vettore a sono presentati come uno successione di coppie (i_t, c_t), con il significato seguente:

$a_i(t) = a_i(t-1) + c_t$

$a_{i'}(t) = a_{i'}(t-1) \quad per\ i' \neq i$

Ci sono due metodi per effettuare l'aggiornamento del vettore , la "cash register" e la "turnstile". Nel primo metodo il valore di c_t non può essere negativo, mentre nel secondo metodo non c'è questo vincolo.
In ogni istante *t*, una query specifica una particolare funzione da calcolare sul vettore a. Per esempio:

a Point Query, restituisce il valore di a_i,

a Range query, restituisce il valore di $\sum_{i=1..r} a_i$.

Un problema molto studiato nella letteratura degli algoritmi gli streaming è la "frequency moments" . La k-esima "frequency moment" di un insieme di frequenze definito nel vettore a è:

$F_k(a) = \sum_{i=1..n} a_i^k$

La F_0 calcola gli elementi diversi che compongono lo stream, la F_1 può essere utilizzata per calcolare gli elementi che eccedono una specificata soglia, problema riconosciuto nella letteratura con il nome di Heavy Hitters. Nel campo di monitoraggio di rete trova corrispondenza nell' identificare gli Elephant flows.

2.Heavy hitters

Il problema si definisce come : data una sequenza di elementi, l'obbiettivo è di elencare quelli che compaiono nello stream con una frequenza che supera una soglia prestabilita.

Come semplice esempio si considera un insieme di possibili oggetti {A, B, C} e lo stream X = {A, A, A, B, A, A, B, A, C}. Il numero degli elementi è m = 4 + 3 + 2 = 9. Con una soglia uguale a 20% (frequenza > 1.8) gli elementi che l'algoritmo deve visualizzare in output sono A e B.

Ritornando alla definizione del metodo delle frequenze si ha:

$OUTPUT\ (t) = \{k \mid a_k(t) > \Theta * F_1(t)\}$

dove $F_1(a) = \sum_{i=1..n} a_i = ||a||_1$ e Θ è la soglia

3.Count-min sketches

La Count-Min sketch con parametri (ε, δ) è rappresentata da un array bidimensionale con w colonne e d righe: `count[1][1] … count[d][w]`.

Per definizione $w=\lceil e/\varepsilon \rceil$ e $d=\lceil ln\, 1/\delta \rceil$. Ogni elemento dell'array inizialmente è zero.

Successivamente sono definiti d funzioni hash

$h1 .. hd : \{1 \ldots n\} \rightarrow \{1 .. n\}$

presi due a due indipendenti in maniera random.

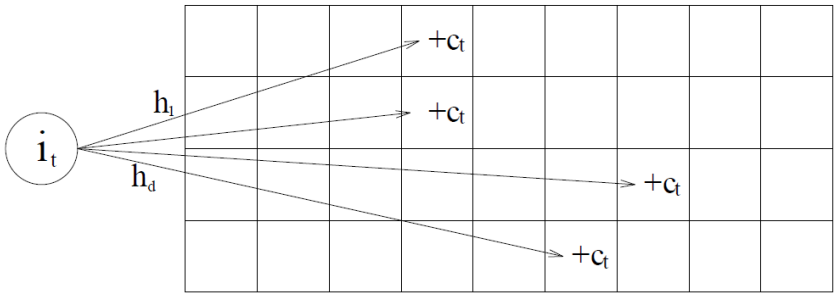
Sul perché la scelta di questi parametri si invita la lettura di [1].

Procedura di update.

Quando in input si presenta il dato (i_t, c_t) significa che all'elemento ai devo aggiungere c_t. (definizione paragrafo 1). Nel caso di CM- sketches abbiamo:

per ogni j: 1 <= j <= d

$count[j, h_j(i_t)] \leftarrow count[j, h_j(i_t)] + c_t$



Lo spazio occupato da CM-sketch è quello necessario per contenere la matrice wxd e le funzioni hash.

Come CM- sketch risponde alle query introdotte nel paragrafo 1?

Point query. La risposta è $P(i) = \min_j \{count[j, h_j(i)]\}$.

Il risultato ha la seguenti proprietà:

a) $ai <= P(i)$,

b) con probabilità al minimo $1-\delta$ si ha $P(i) <= a_i + \varepsilon||a||_1$

ovvero che $ai <= P(i) <= a_i + \varepsilon||a||_1$

4. Algoritmo

Una semplice procedura per il problema degli Heavy Hitters con gli CM-sketches si può definire così:

Per ogni dato in input (i_t, c_t),

- aggiorno il valore di $F_1 = ||a||_1$, questo è possibile perché $F_1 = \sum_{i=1..n} c_i$

- aggiorno il valore di a_i

- eseguo una point query per a_i. Se questo valore supera la soglia, ovvero se

  $P(i) > \Theta * ||a||_1$ allora inserisci il valore di $P(i)$ in uno heap.

- quando lo stream termina stabilisci se ogni elemento nello heap verifica ancora la condizione.

5.La realizzazione

Una implementazione di CM-sketches, scritta nel linguaggio di programmazione C, si può trovare in questo indirizzo: http://www.cs.rutgers.edu/~muthu/massdal-code-index.html

L'autore è la persona che gli ha descritti per la prima volta. Per fare i test viene utilizzato uno stream, che è una successione di numeri interi generati da un algoritmo che garantisce delle proprietà statistiche (che ci sia almeno un Heavy Hitter). Il test per CM-sketches non viene fornito. Ma non è difficile realizzarne uno con i passi descritti nel paragrafo 4.

Un'atra realizzazione delle CM-sketches viene fornita in questi paper [2][3]. Lo stream in questo caso è una successione di dati di rete presi da un router. Il codice è scritto in C++ e la libreria utilizzate per catturare i pacchetti è PCAP. E' scritto sempre dalla stessa persona e con piccole modifiche è uguale al codice C.

La mia implementazione di CM-sketches mantiene la struttura del codice scritto in C. Sono rimosse le parti del codice che non servono.

Per catturare i pacchetti viene utilizzata la libreria PF_RING. Per ogni pacchetto che passa nell'interfaccia di rete scelta (-i ethX), viene analizzato l'indirizzo IP della sorgente. Questo significa che se il programma "CM_Sketches" parte con l'opzione RX (-e 1) allora trovo gli IP che generano più traffico verso il host dove è stato eseguito il programma. Se parte con l'opzione TX (-e 2) allora trovo gli IP verso i quali il host genera più traffico. Naturalmente la seconda modalità sarà possibile se PF_Ring parte con l'opzione "enable_tx_capture" uguale a 1.

Per quanto riguarda PF_RING, viene eseguita con l'opzione "transparent_mode = 2". Questo perché voglio catturare il più grande numero dei pacchetti possibile.

Il codice riutilizza funzioni generali definiti nel file pfcount.c; intoa(addr), printstats(), etc.. . Nel main viene inizializzato la "Count Min Sketch", CM_Init(width, height, 12), dove la width e la height sono i parametri dello sketch come specificati nel paragrafo 3. La parte più significativa nel codice di "cm_sketches.c" è il

metodo dummyPacketProcess. Qui viene aggiornato lo sketch ed eseguito l'algoritmo descritto nel paragrafo 4.

L'intervallo di tempo per effettuare il monitoraggio è di 60 secondi.

7. Installazione ed esecuzione

Scaricare e scompattare l'archivio CMSketches_PFRING.zip. Copia i file *.c nella cartella "examples" della installazione di PF_RING. Compilare con il seguente commando:

```
gcc -c prng.c countmin.c
```

Aprire il file "Makefile" ed inserire nella sezione "Main Targets", sotto "PFPROGS". Successivamente aggiungere il target "cm_sketches":

```
cm_sketches: cm_sketches.o ${LIBPFRING}
	${CC} cm_sketches.o countmin.o prng.o ${LIBS} -o  $@ -lm
```

Eseguire con il commando :

```
sudo ./cm_sketches  -h
```

8. Bibliografia

[1] An Improved Data Stream Summary: The Count-Min Sketch and its Applications

Graham Cormode and S. Muthukrishnan

[2] Methods for finding frequent items in data streams, Graham Cormode and Marios Hadjieleftheriou

http://www.springerlink.com/content/u5106x1223212882/fulltext.pdf

[3] Il codice delle funzioni implementate si trova in: http://www.research.att.com/_marioh/frequent-items

[4] PF_RING User Guide. Version 4.6.1 http://www.ntop.org/PF_RING.html