Controllo delle Query Distribuite

Parte III

Indice

Capitolo 1

Introduzione

Torniamo a preoccuparci del problema di confidenzialità, nel contesto di computazione di query distribuite; l'assunzione è che non tutti siano autorizzati a vedere tutti i dati, ma ci sono dei vincoli di confidenzialità che devono essere rispettati.

1.1 Join sovrani

Questo approccio sfrutta la presenza di un hardware fidato (nel senso che nessuno può vedere cosa fa), che può ricevere i dati per eseguire le computazioni. Lo scenario è:

- si hanno due *data owner* che non si fidano l'uno dell'altro
- c'è una terza parte, che ha a disposizione dell'hardware fidato, che esegue la computazione

L'idea è che le parti criptano i dati e li mandano all'hardware, che si occupa di:

- decriptare i dati
- eseguire la computazione
- recriptare i dati e darli al client

Un osservatore potrebbe inferire sulla base del risultato qualcosa, come ad esempio sulle dimensioni del risultato o sul tempo richiesto ad eseguire la computazione.
$\Rightarrow$ l'output deve avere più o meno sempre la stessa dimensione e tempo di computazione, per cercare di ridurre l'inferenza

1.2 Access patterns

Cercano di specificare come le fonti informative devono essere accedute.
Definiamo un *access pattern* con un esempio:

- abbiamo 3 relazioni, ciascuna con un access pattern, ovvero dei vincoli di accesso
- si ha una lettera per ciasuno attributo delle relazione
  - o per output
  - i per input

$$\text{Insurance}^{oi}(\underline{\text{holder}},\text{plan})$$
$$\text{Hospital}^{oioo}(\underline{\text{patient}},\text{YoB},\text{disease},\text{physician})$$
$$\text{Nat\_registry}^{ioo}(\underline{\text{citizen}},\text{YoB},\text{healthaid})$$

per accedere all'attributo "o" mi deve dare l'attributo "i"; si pongono dei vincoli, l'accesso non è libero

Questa tecnica presenta alcuna svantaggi:

- limitata espressione delle limitazioni
- tipicamente ci sono due entità, non un vero scenario distribuito
- può essere difficile da usare nella pratica

1.3 Autorizzazioni basate su viste

La peculiarità di questo approccio è che le restrizioni di accesso dipendono dal contenuto del dato.

- Relations:
  Treatment(ssn,iddoc,type,cost,duration)
  Doctor(iddoc,name,specialty)
- Integrity constraint: each treatment is supervisioned by a doctor
- Authorization view:

```
CREATE AUTHORIZATION VIEW TreatDoct AS
SELECT D.name, T.type, T.cost
FROM Treatment AS T, Doctor AS D
WHERE T.iddoc=D.iddoc
```

- Query: `SELECT type, cost FROM Treatment`

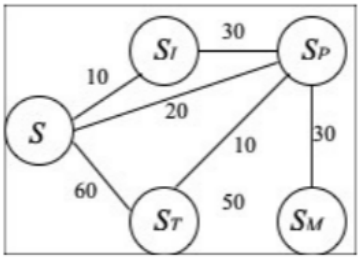
Verifico se una query può essere eseguita sulla base delle autorizzazioni che ho definito; il client scrive la sua query, e il server cerca di rielaborarla sulla base delle viste che sono state definite.
Nel caso in cui una query non possa essere eseguita, ci sono due scenari possibili:

- *truman:* ti restituisco un risultato parziale, che corrisponde non alla query che mi hai chiesto ma alla vista che è stata definita (facendotelo passare come completo)
- *non-truman:* non ti restituisco nulla e ti dico che non sei autorizzato ad accedere al risultato

1.4 Pairwise authorizations

Ci sono diversi *providers* che si conoscono e che formano delle *coalizioni*; sono disposti a condividere le proprie informazioni per un obiettivo comune.
Ciascun provider ha:

- una o più relazioni
- uno o più server



I server formano una rete, possono comunicare tra di loro; una computazione vuole essere effettuata **minimizzando il costo** (ciascun canale a un costo associato) e **rispettando le restrizioni** sul flusso di informazioni (chi può vedere che cosa).
⇒ Si definisce un ***safe query plan***, ovvero un modo per soddsifare la query in modo sicuro e che minimizzi i costi:

- per le operazioni unarie non ci sono problemi, dato che non richiedono alcun trasferminento di dati
- per le operazioni di join, viene richiesto la cooperazione tra i due server:
  - uno funge da *master*, ha il compito di eseguire il join
  - uno funge da *slave*, aiuta il master

1.4.1 Broker join

Ci sono due relazioni su due server, su cui dobbiamo eseguire il join. Tipicamente, uno dei due server funge da master e l'altro da slave; se però sono state definite delle restrizioni che impediscono di accedere all'altra relazione, questa architettura non può esssere utilizzata.
Con il broker-join si usa (se esiste) un terzo server che sia autorizzata ad accedere alle relazioni ed eseguire il join; se ne esistono più di uno, seleziono quello con il costo minore.

1.4.2 Peer-join

Sfrutto entrambi i server, perché entrambi sono autorizzati ad accedere all'altra relazione; il join viene eseguito in modo che uno dei due server manda la tabella all'altro, che eseguirà il join per poi dare il risultato al client.