Relazione Assignment 2

Programmazione Concorrente e Distribuita

Mariano Sciacco (1142498)

2018 — 2019

Corso di Laurea in Informatica, Scuola di Scienze, Dipartimento di Matematica, UniPD

Diffie-Hellman

Introduzione

Il secondo assignment richiede di eseguire un metodo della classe `Diffie-Hellman.crack` col fine di indovinare le chiavi private di *Alice* e *Bob*, avendo a disposizione le chiavi pubbliche e sapendo che l'algoritmo è meno sicuro dal momento che si usano 2^{32} combinazioni di possibili valori di **a** e **b** (con $0 \leq a \leq 65536$ e $0 \leq b \leq 65536$).

Realizzazione

La risoluzione del corrente esercizio si basa su un approccio di tipo *brute force* e richiede di fare uso dei thread al fine di velocizzare il processo di crack.

L'approccio impiegato dal sottoscritto per la realizzazione della funzione `crack()` è stato il seguente:

1. Creo due array vuoti sincronizzati. Il primo array verrà utilizzato per salvare le possibili combinazioni di **a** che soddisfano il risultato del logaritmo discreto. Il secondo, analogamente, per le possibili combinazioni di **b**.
2. Imposto il carico di lavoro del processore basandomi sul numero di cores disponibili e imposto il numero di thread da far eseguire in parallelo.
3. Realizzo un `newFixedThreadPool()` col fine di gestire automaticamente il lavoro dei threads.
4. Avvio i lavori di computazione dei threads nel `PoolThreads`.
5. Con i risultati trovati, calcolo il valore di **a** e **b** e li ritorno.

Nel dettaglio si è deciso di valutare un range di numeri (`ThreadBound`) nei quali ciascun thread deve calcolare i possibili risultati di **a** e **b**. I risultati verranno riportati negli array realizzati all'inizio; alla fine di tutto si eseguiranno gli ultimi calcoli sempre tramite gli stessi array.

Il numero dei threads è stato deciso in maniera scalabile, selezionando un numero doppio di core disponibili con il processore, così da rendere la CPU sotto massimo sforzo.

Classi e metodi ausiliari usati

Col fine di realizzare l'esercizio, si è fatto uso di una classe e di un metodo ausiliario, in modo da avere più chiarezza del codice.

Class Calcolatore

La classe `Calcolatore` è una classe che estende `Threads` e che esegue l'override del metodo `run()` (come di consueto). Nel costruttore vengono presi alcuni parametri che verranno successivamente

analizzati appositamente col fine di trovare tutte le possibili combinazioni candidabili come **a** e **b**.

Il metodo `run()` viene invocato in modo parallelo e la computazione sfrutta tutta la potenza del processore col fine di eseguire i seguenti calcoli di controllo:

- $g^x \bmod p == A$
- $g^x \bmod p == B$

dove `x` è la mia incognita che sto cercando e che soddisfa il risultato dell'equazione. Se la `x` soddisfa l'equazione, viene salvata negli array delle **a candidabili** o **b candidabili** come possibile soluzione.

Metodo computeResult(...)

Questo metodo viene richiamato quando i thread hanno concluso i calcoli e fa uso degli array ausiliari col fine di trovare finalmente le chiavi private di Alice e Bob. Il funzionamento è molto basilare sebbene abbia una complessità $O(n^2)$ e si sviluppi nel seguente modo:

- Se $(B^a \bmod p) == (A^b \bmod p)$ allora ho trovato le chiavi **a** e **b**.
- Continuo a cercare, altrimenti.

Il metodo infine ritorna un array di interi contenenti le chiavi trovate.

Risultati

Il test ha impiegato **33s** per ottenere le chiavi, impostando la CPU con un workload massimo ed è stato eseguito su un *Macbook Air 2011* con le seguenti specifiche:

- **Processore:** Intel Core i5 (dual core) 1.7Ghz
- **RAM:** 4 GB

Per completezza vengono riportati anche i risultati dello script durante e alla fine della computazione insieme ad uno screen che identifica il workload della CPU.

Print dello script

```
Avvio..
(/) Esecuzione Thread per range: [0, 8192]
(/) Esecuzione Thread per range: [8192, 16384]
(/) Esecuzione Thread per range: [16384, 24576]
(/) Esecuzione Thread per range: [24576, 32768]
(/) Esecuzione Thread per range: [32768, 40960]
(/) Esecuzione Thread per range: [40960, 49152]
```

```
(/) Esecuzione Thread per range: [49152, 57344]
(/) Esecuzione Thread per range: [57344, 65536]
chiave a = 45663
chiave b = 55427
-> Tempo di esecuzione: 33s

Process finished with exit code 0
```

CPU Workload

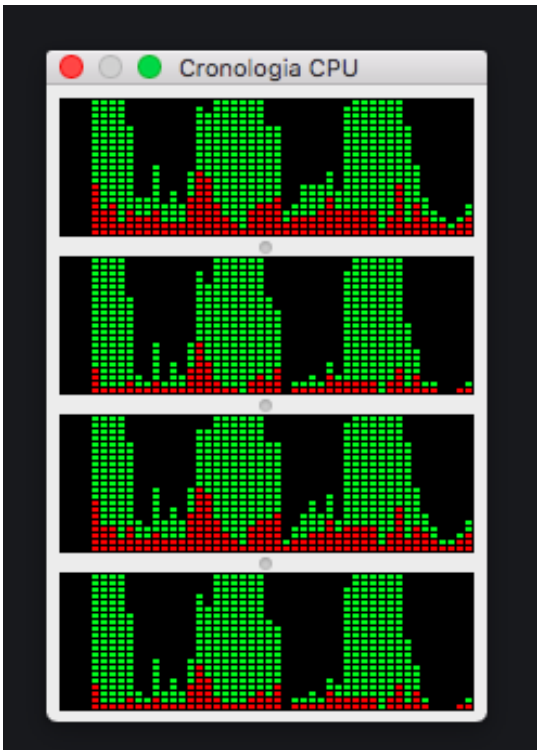

Figura 1: Workload della CPU. In questo caso il test è stato avviato 3 volte. Il colore rosso indica l'uso della CPU per il sistema operativo, il verde per il resto.