Università degli Studi di Padova

Programmazione Concorrente e Distribuita

A.A. 2014/2015

Risolutore di Puzzle - Parte 2

Autore:

GIACOMO CUSINATO
Mat. 1054137

17 gennaio 2015

Indice

1 Ambiente di sviluppo

Il progetto è stato realizzato in ambiente Mac OS (versione Yosemite 10.10) sia per la parte relativa al codice Java, sia per quella relativa alla documentazione. È stato inoltre testato con successo nella macchine di laboratorio con i dovuti comandi per impostare Java 7 come versione principale del sistema. Per la stesura della relazione, invece, è stato utilizzato il linguaggio LaTeX tramite la ditribuzione MacTex. Nel progetto, come richiesto, è stato incluso un makefile per semplificare la complilazione dei sorgenti ed un script bash accetta due parametri in ingresso (ovvero l'input path e l'output path dei file richiesti) e si occupa di compilare le sorgenti Java ed avviare il main del programma. Lo script è stato reso eseguibile e può essere lanciato semplicemente col comando `"./puzzlesolver.sh inputPath outputPath"` dalla cartella in cui è presente il file.

2 Differenze con la versione precedente

Il progetto non ha subito molte variazioni rispetto alla versione precedente in quanto le operazioni effettuate dall'algorimo scelto sono risultate facilmente portabili ad un'architettura concorrente. Di seguito sono riportate le modifiche effettuate alla classe principale, contentente l'algorimo di risoluzione, ed una descrizione della nuova classe `PuzzleThread`.

2.1 La classe PuzzleThread

La classe `PuzzleThread` definisce un un oggetto la cui istanza può essere lanciata dal programma in un thread a supporto della risoluzione dell'algorimo. La classe implementa l'interfaccia `Runnable` che rappresenta l'attività logica eseguibile dal thread e con cui è possibile creare oggetti di tipo `Thread`. È stato scelto di implementare tale interfaccia invece che estendere la classe `Thread` dato che l'unico scopo della classe in questione è quello di effettuare l'override del metodo `run()` per definire la logica eseguibile, metodo implementato appunto da `Runnable`, scelta che aumenta inoltre l'estensibilità del classe considerato il vincolo di ereditarietà multipla in Java. La classe è composta del seguente costruttore e metodo:

- `public PuzzleThread(int, PuzzleSolver)`: il costruttore della classe accetta come parametri formali un intero, che indica la riga del puzzle dove il thread andrà ad operare, e l'istanza dell'oggetto di tipo `PuzzleSolver`, che espone il metodo principale per riordinare la riga. Entrambi i valori vengono memorizzati in due campi dati, entrambi utilizzati dal metodo `run()`.

- `public void run()`: è il metodo dichiarato dell'interfaccia `Runnable` che sarà lanciato una volta in esecuzione il thread. Si occupa di chiamare il metodo `reoderRow()`, illustrato in seguito, utilizzato dall'oggetto di tipo `PuzzleSolver` per ordinare una determinata riga del puzzle.

2.2 La classe `PuzzleSolver`

La classe `PuzzleSolver` è la classe di riferimento del programma, ovvero quella che contiene l'algorimo di risoluzione ed elabora i dati in accesso ed uscita.

2.2.1 L'algoritmo di risoluzione

L'algorirmo di risoluzione presenta essenzialmente lo stesso funzionamento di quello illustrato nella precedente relazione, sebbene sia sia stato reso parallelo attraverso la creazione di un thread per ogni riga del puzzle. Il thread principale, come spiegato in precedenza, si occupa di riempire la prima colonna del puzzle così da avere un punto di partenza per i threads, ognuno dei quali si occupera di terminare l'ordinamento di ciascuna riga. Ogni thread opera in una singola ed unica riga del puzzle, proprio per questo non è stato necessario gestire l'utilizzo di risorse condivise tramite i costrutti di cui dispone Java.

2.2.2 Il metodo `void reoderRow(int row)`

Il metodo `reoderRow(int row)` ha la funzione di riordinare un riga del puzzle, rappresentato da un array bidimensionale. È il metodo utilizzato da ogni thread per accedere al puzzle ed ordinare la riga desidarata grazie all'instanza di `PuzzleSolver` passata come parametro formale al costruttore dell'oggetto di tipo `PuzzleThread`.

2.2.3 L'utilizzo del Thread Pool

La creazione di un thread per ogni riga è una scelta che può causare un utilizzo di memoria particolarmente elevato e basse performance, in particolar modo durante la manipolazione di puzzle a dimensioni elevate. Il pool di thread è un modulo che permette la gestione di più di thread creando una coda di task in esecuzione o in attesa in modo da limitare il più possibile l'overhead causato dalla creazione ed esecuzione dei thread. In supporto all'algorimo di risoluzione, quindi, è stato scelto di utizzare un `FixedThreadPool`, un tipo di thread pool che fissa un massimo numero di thread operanti in contemporanea e gestisce in automatico tutti i task assegnati in attesa.

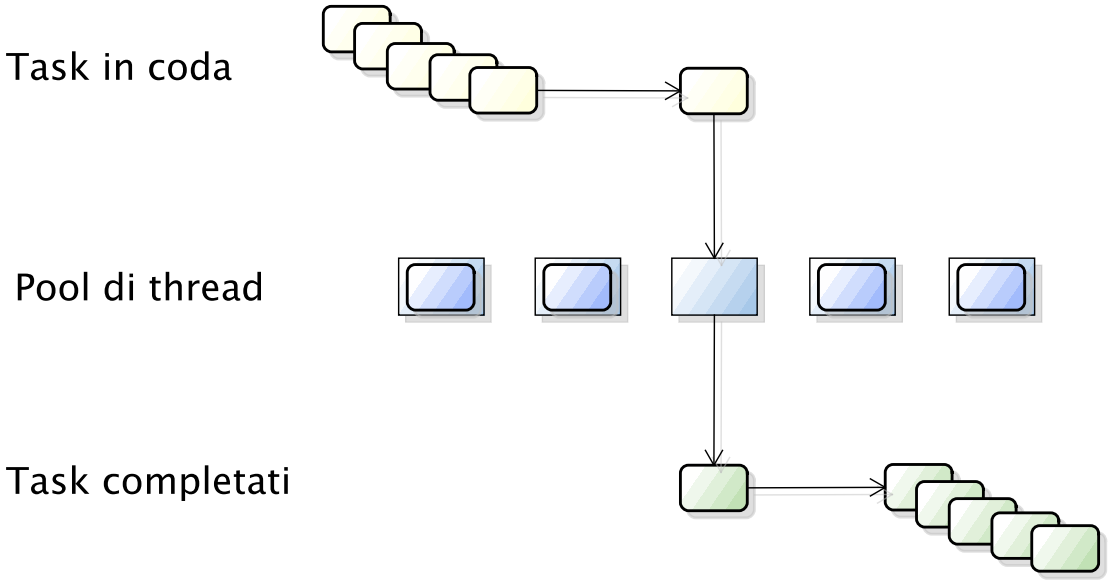

Figura 1: Thread Pool con 5 thread fissi

Il metodo `reorder()` della classe `PuzzleSolver` utilizza un oggetto di tipo `ExecutorService` per gestire un pool di 5 thread fissi, costruito tramite il metodo statico `newFixedPoolThread(5)`. In questo modo è possibile creare un oggetto di tipo `PuzzleThread` per ogni riga e passare l'istanza creata al pool di thread tramite il metodo `execute()`. Il thread pool gestisce ora in automatico la coda dei task e l'esecuzione di un thread non appena un altro in esecuzione termina il metodo `run()`. Una volta assegnati tutti i task, viene invocato il metodo `shutdown()` per terminare il thread pool una volta completati tutti i task, situazione che si

verificherà al termine del successivo ciclo while, che assicura il completamento di tutti i task di riordinamento prima di proseguire con le istruzioni del thread principale. In questo modo si ha la certezza di ricevere il puzzle completato prima di utilizzare l'array ordinato per restituire il risultato del programma.