Assignement01

Marco Raggini
Matricola:0001141793
Email: marco.raggini2@studio.unibo.it

Francesco Carlucci
Matricola:0001136938
Email: francesco.carlucci6@studio.unibo.it

12 maggio 2024

Indice

Capitolo 1

Prima parte

1.1 Analisi del problema

Il progetto è costituito da simulazioni che riguardano il comportamento di macchine che percorrono strade, contenenti anche semafori. In particolare era richiesto l'utilizzo di un approccio asincrono e task-based.

1.2 Stategia risolutiva

In questo caso la strategia risolutiva era dettata dalla consegna stessa, ovvero l'utilizzo di un approccio **task-based**.
L'implementazione proposta prevede l'utilizzo di `virtual threads` tramite un `Executor` che ne gestisce il funzionamento (approccio master-slave). Per ogni macchina o semaforo è presente un virtual thread che si occupa di gestirla.
In particolare i task creati sono:

- **Thread manager**: è il thread che in cui è presente l'`Executor` e gestisce quindi la creazione di tutti gli 'slave' che andranno poi ad eseguire i relativi task.
- **Task macchina**: gestisce il funzionamento di una singola macchina
- **Task semaforo**: gestisce il funzionamento di un singolo semaforo

In questo caso non sono state utilizzate `Future`, in quanto nessun task aveva necessità di ritornare un valore alla fine del proprio funzionamento.

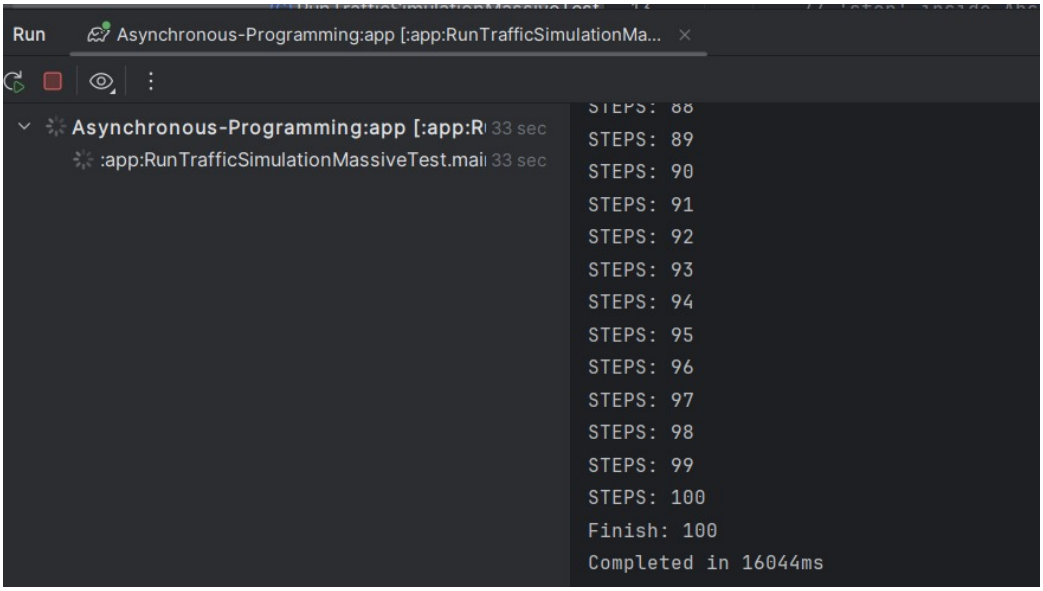

Figura 1.1: tempo di esecuzione di massive test con virtual thread

1.3 Performance

Il risultato ottenuto è in linea con le aspettative, infatti è notevolmente inferiore rispetto alla soluzione senza l'utilizzo di therad (38 s), e poco superiore rispetto a quella sincrona (10 s).

Capitolo 2

Seconda parte

2.1 Analisi del problema

Il sistema deve restituire un report contenente l'elenco delle pagine che contengono una determinata parola e l'occorrenza della parola nella pagine collegate a partire da un indirizzo specificato, ricorsivamente, fino a un certo livello di profondità.

Il problema è stato risolto in tre modi diversi: con programmazione asincrona ad eventi/event-loop, con programmazione reattiva e con i virtual thread.

2.2 Event-loop

Nell'approccio event-loop c'è un singolo thread di controllo che aspetta che si verifichino degli eventi e li elebora.

La regola principale di event-loop consiste nel non bloccare mai gli handlers degli eventi e far si che essi terminino sempre per garantire un sistema sempre reattivo.

Per queste ragioni nella soluzione proposta è stato utilizzato un event-loop, creato con vertx, che gestisce la ricerca delle parole e dei link nelle pagine web. Il main loop aspetta che si verifichino un evento, ovvero l'aver trovato la parola corrispondete a quella da cercare, e aggiorna una mappa che tiene traccia per ogni link visitato il numero di occorrenze della parola trovate. Poiché riuscire ad ottenere la pagina web in modo da poterla usare comodamente è un'operazione piuttosto lunga è stato utilizzato un callable che assolve questa funzione in modo da non bloccare il thread principale. Per aggiornare la GUI in tempo reale è stato utilizzato un consumer che ogni

volta in cui viene trovata la parola corretta restituisce il numero di occorrenze attuale della parola tramite il verticle.

Alla fine dell'escuzione del verticle si ottiene il report finale con tutte le pagine e le occorrenze relative e il programma termina.

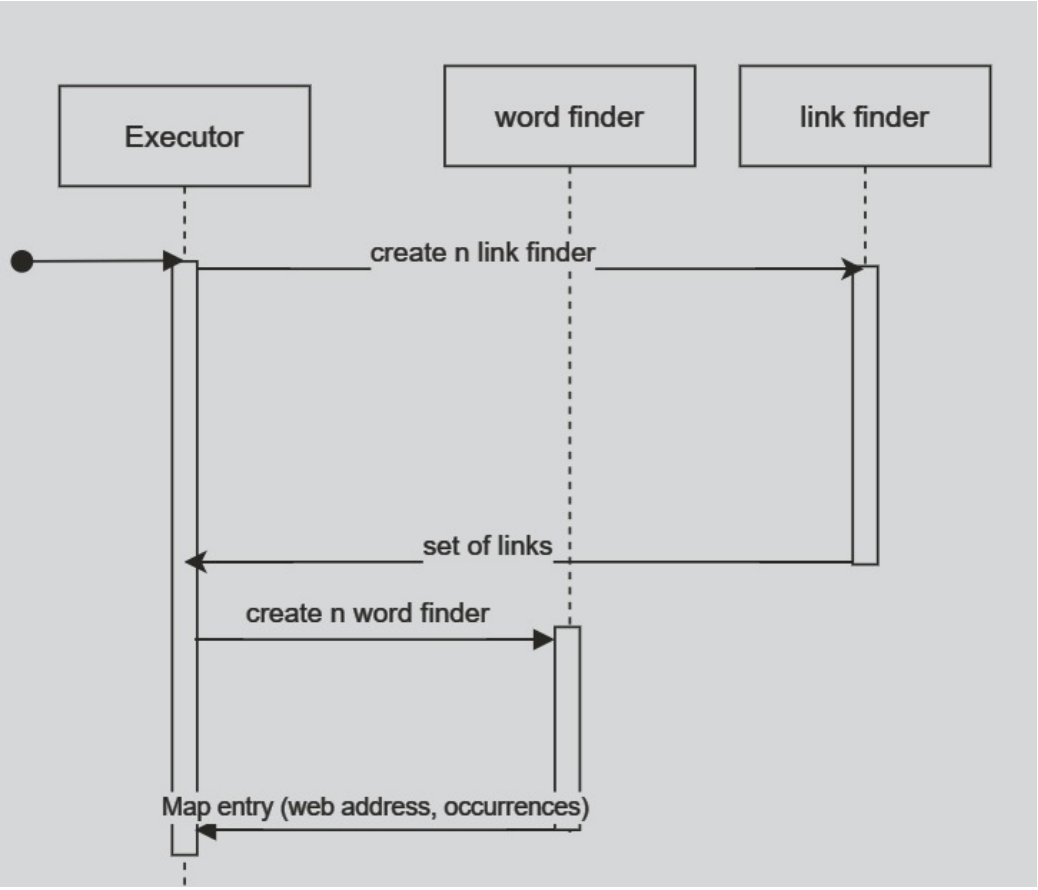

Figura 2.1: Diagramma di sequenza di event-loop

2.3 Virtual thread

Nell'approccio con i virtual thread il programma presenta due fasi:

1. **Ricerca dei link**: In questa prima fase viene creato un virtual thread per ogni pagina web a cui connettersi per cercare altri link.
2. **Conta delle occorrenze**: In questa seconda fase, dopo aver raccolto tutti i link da visitare, viene creato un virtual thread per ogni pagina in cui bisogna contare le occorrenze.

Attraverso l'Executor vengono creati i thread virtuali e tramite le Future, in particolare `future.get()` viene ritornata una `Map.Entry` dove è presente nella chiave il link nel quale si è eseguita la conta e nel valore è presente il numero delle occorrenze. Quando tutti i thread hanno finito la loro esecuzione si crea la mappa finale con tutti i risultati e viene chiamata la funzione `Executor.shutdown()` per arrestare i thread generati. In questo modo la ricerca viene eseguita in modo concorrente e asincrona.

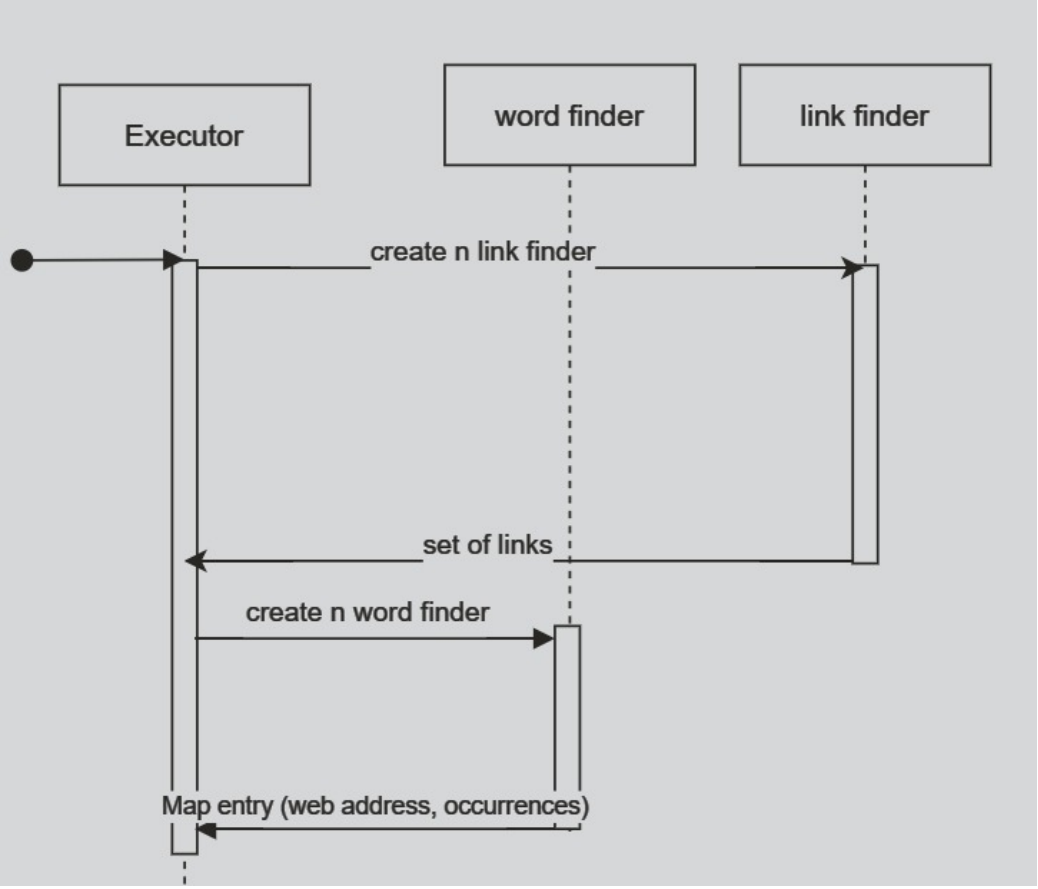

Figura 2.2: Diagramma di sequenza di virtual thread

2.4 Reattiva

Nell'approccio reattivo si è fatto uso delle callback per poter contare in modo asincrono le occorrenze all'interno della pagina web, in particolare sono state usati dei `Flowable`. All'interno del flowable il thread si connette alla pagina web e conta le parole, aggiornando così la mappa finale, una volta finita la

ricerca delle parole e dei link per le ricerce successive viene ritornato il set dei link trovato. Una volta che le callback sono terminate, la mappa è già aggiornata per essere presentata.

2.5 Performance

Nonostante spesso le prestazioni rilevate possono essere poco affidabili in quanto il socket a volte non riesce a connettersi ai siti, abbiamo rilevato che l'implementazione più rapida è quella con i virtual thred, mentre le altre due ottengono risultati simili (di ben dieci volte inferiori rispetto ai virtual thread). Questo è dovuto al fatto che in event-loop c'è un solo verticle che gestisce tutto, mentre in reactive è stato utilizzato un unico thread.