RELAZIONE ASSIGNMENT-02

Luca Ragazzi 0000897452

1) Analisi del problema

Scopo del progetto è lo sviluppo di un tool che, dato un set di file, mostri in real-time le N parole più frequenti aventi più di K caratteri. Una caratteristica dell'applicazione è la possibilità di modificare il set, aggiungendo o rimuovendo i file durante la computazione. Si chiede inoltre di proporre 3 soluzioni che utilizzino tecnologie diverse, in particolare utilizzando:

1. Programmazione multi-tasking con Task ed Executor
2. Programmazione asincrona con event-loop
3. Programmazione reattiva

2) Descrizione della soluzione proposta

L'interfaccia grafica dell'applicazione è stata sviluppata con JavaFX.

PROGRAMMAZIONE MULTI-TASKING

Questa soluzione è stata realizzata con 2 approcci diversi:

1) Nel primo approccio è stato utilizzato il framework Fork/Join. Questo pattern utilizza una strategia del tipo "divide et impera". Nella fase di Fork il problema principale viene diviso ricorsivamente in sotto-task più piccoli fino a quando sono sufficientemente semplici da poter essere eseguiti in modo asincrono. La fase di Join raccoglie ricorsivamente i risultati di ciascun sotto-task unendoli nel risultato finale. Per fornire l'esecuzione parallela, il framework utilizza un pool di Thread chiamato ForkJoinPool. Nella soluzione proposta si sono utilizzati 3 tipologie diverse di Task : un "master Task" che ha il compito di delegare il lavoro ai "worker Task", uno per ogni file da computare, e un ulteriore Task con il compito di processare la lettura dei file per righe.

2) Nel secondo approccio è stato utilizzato il multi-tasking con l'Executor Service. Esso fornisce un meccanismo asincrono per la gestione dei task in background. Fornisce anche un controllo sui thread in esecuzione. La soluzione propone l'utilizzo di due Task, uno con il compito di leggere i file e uno con il compito di calcolare le parole più ricorrenti per ogni file. Entrambi sono gestiti da un thread che ha anche il compito di salvare in un monitor i risultati delle computazioni.

PROGRAMMAZIONE ASINCRONA CON EVENT-LOOP

Questa soluzione è stata realizzata utilizzando il framework Vert.x. Esso offre un modello di programmazione di tipo Event Driven, ovvero è progettato per lavorare in modo asincrono, registrando handler per gestire gli esiti delle computazioni. Importante è il concetto di Event Loop, nel quale un singolo thread esegue un loop infinito, dove riceve e invia gli eventi a tutti gli handler registrati. La logica di business dell'applicazione è eseguita all'interno di uno o più componenti chiamati Verticle. Ogni Verticle viene eseguito in modo concorrente rispetto agli altri senza alcuna condivisione di uno stato. In tal modo è possibile creare applicazioni multi-threaded senza preoccuparsi della gestione della concorrenza. Ogni verticle vive all'interno di un'istanza Vert.x ed è gestito da un Event Loop. I Verticle possono comunicare tra di loro mediante l'utilizzo dell'Event Bus.
Nella soluzione proposta è stato utilizzato un singolo Verticle che garantisce il corretto funzionamento dell'Event Loop. Inoltre, per gestire gli eventi provenienti dalla GUI si è optato per l'utilizzo dell'Event Bus, che, nello specifico, gestisce tre tipi diversi di eventi, per cui su tre canali diversi, in particolare si gestisce l'aggiunta di un file, la rimozione di un file e la comunicazione del risultato. Per quanto riguarda la comunicazione del risultato, è stato utilizzato il "MessageCodec" per poter inviare alla GUI, tramite il bus, la mappa del risultato.

PROGRAMMAZIONE REATTIVA

Questa soluzione è stata realizzata utilizzando la programmazione reattiva con rxjava. Ci sono due concetti fondamentali, quello dell'Observable e quello dell'Observer. Un Observable non è altro che un oggetto che emette zero o più elementi per poi terminare con successo, oppure durante il flusso degli elementi si interrompe a causa di un errore. Un Observer, invece, hanno il compito di "consumare" gli elementi emessi dagli Observable e gestire la loro terminazione.
Nella soluzione proposta è stato utilizzato il PublishSubject, ovvero un oggetto che rappresenta contemporaneamente sia un Observable che un Observer. Il PublishSubject può iniziare a emettere elementi immediatamente dopo la sua creazione, per cui vi è il rischio che uno o più elementi possano essere persi tra il momento in cui esso è stato creato e l'Observer che lo sottoscrive.
Per gestire il flusso reattivo sono stati utilizzati degli Scheduler e, a seconda della trasformazione diversa applicata, si è scelto di utilizzare Schedulers.io e Schedulers.computation (indicato per una computazione più dispendiosa rispetto il precedente). Anche in questa soluzione si è optato per l'utilizzo di un monitor per gestire e salvare i risultati.

3) Performance della soluzione proposta

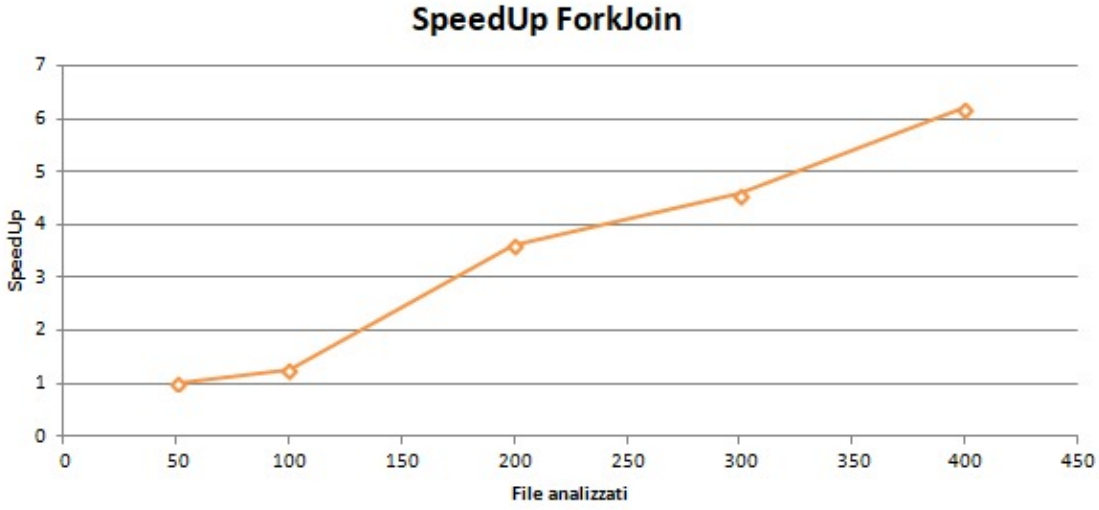
Per analizzare il comportamento delle tre soluzioni proposte, è stata realizzata anche una versione sequenziale con lo scopo di comparare i tempi di esecuzione in tutte le diverse casistiche, calcolando e mostrando grafici sulle performance.
Di seguito si mostra la tabella con i tempi di esecuzione delle diverse soluzioni proposte. I tempi sono stati presi considerando l'applicazione senza interfaccia grafica. I tempi scritti rappresentano un media su 10 misurazioni prese.

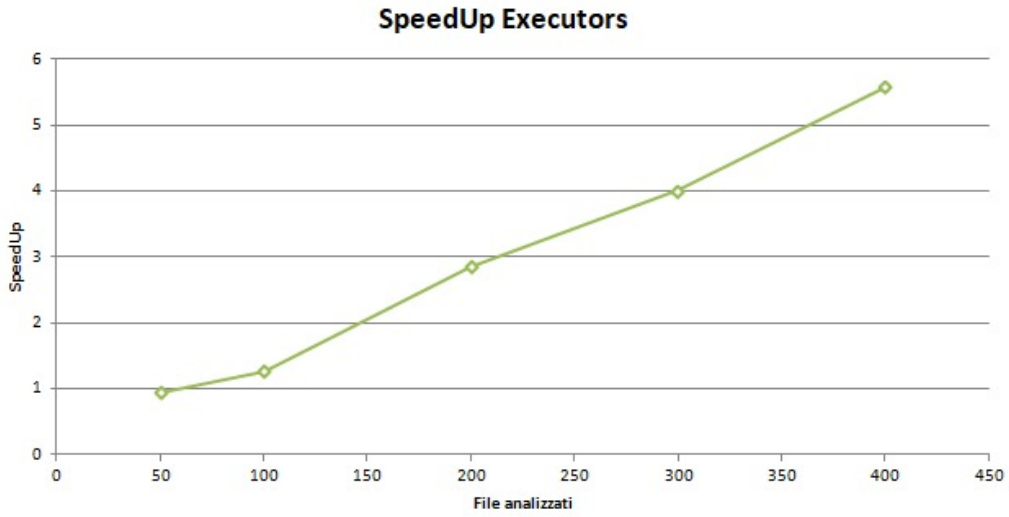
Numero di file	Tempo di esecuzione sequenziale (ms)	Tempo di esecuzione Fork-Join (ms)	Tempo di esecuzione Executors (ms)	Tempo di esecuzione Vertx (ms)	Tempo di esecuzione Reactivex (ms)
50	5424	5329	5784	4948	4784
100	15172	12054	12051	10254	8704
200	54269	15005	19005	14541	12054
300	102925	22487	25678	21875	19215
400	167929	27015	30118	27841	24655

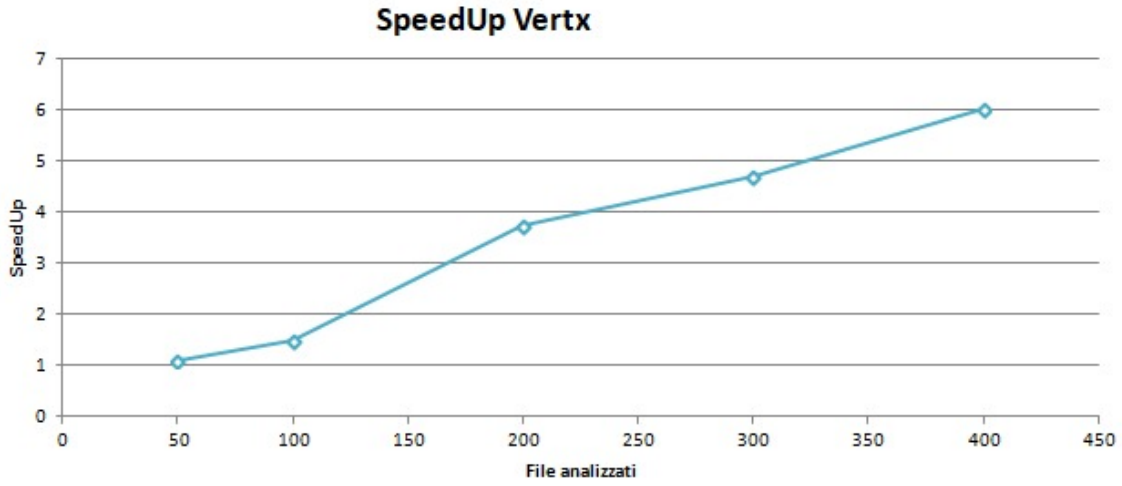
E la relativa tabella con gli speedup:

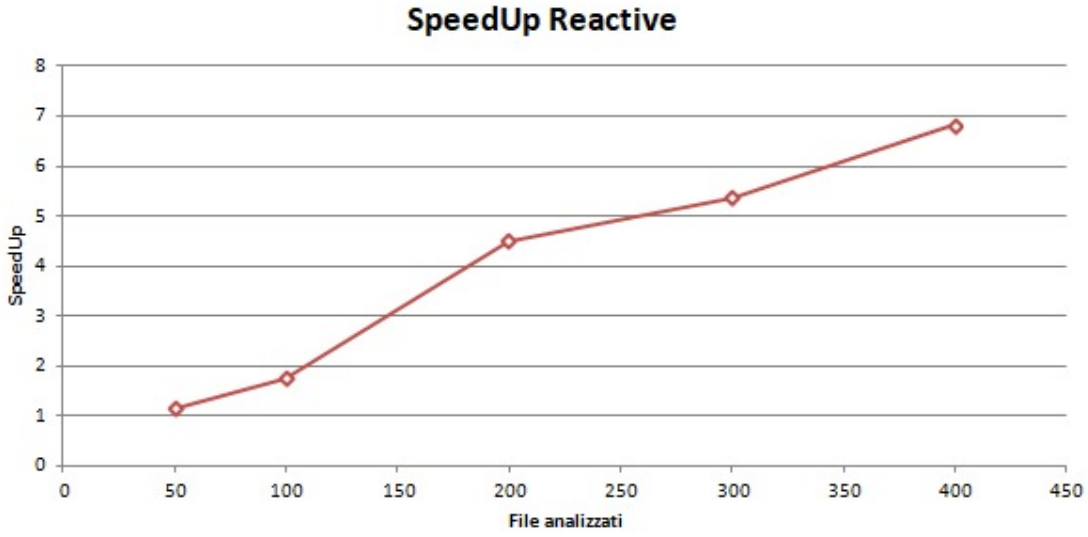
Numero di file	SpeedUp ForkJoin	SpeedUp Executors	SpeedUp Vertx	SpeedUp Reactivex
50	1,0178	0,9378	1,0962	1,1338
100	1,2587	1,2590	1,4796	1,7431
200	3,6167	2,8555	3,7321	4,5021
300	4,5771	4,0083	4,7051	5,3565
400	6,2161	5,5757	6,0317	6,8111

Di seguito si mostrano i grafici relativi allo speedup:




SpeedUp Executors
SpeedUp
File analizzati
0
1
2
3
4
5
6
0
50
100
150
200
250
300
350
400
450


SpeedUp Vertx
SpeedUp
File analizzati
0
1
2
3
4
5
6
7
0
50
100
150
200
250
300
350
400
450


SpeedUp Reactive
SpeedUp
File analizzati
0
1
2
3
4
5
6
7
8
0
50
100
150
200
250
300
350
400
450