Distributed Grep

Esercizio per il corso di Sistemi Distribuiti e Cloud Computing

Mirko Leandri 0299946 - Ilenia Rocca 0285151
Corso di studi in ingegneria informatica
Università degli studi di Roma "Tor Vergata"

INTRODUZIONE

L'esercizio assegnato durante il corso prevede l'implementazione di un "grep" distribuito nel linguaggio Go, utilizzando RPC e un paradigma map-reduce.

IMPLEMENTAZIONE

L'IDE utilizzato è Goland e la struttura del progetto è un tipico client server. Nella cartella server registriamo appunto quest'ultimo attraverso le chiamate Register, Listen e Accept come visto a lezione, in questo modo mettiamo in attesa il server sulla porta 4040. A questo punto possiamo creare i servizi offerti dal nostro server attraverso le interfacce dei metodi che andremo ad utilizzare, che poi dovremo passare alle chiamate "Call" o "Go" del client.
In particolare, andiamo a creare un'unica interfaccia che prende come input una stringa (la parola da cercare) e una struct che andrà a contenere la risposta. Questa mantiene a sua volta una stringa (la concatenazione delle righe contenenti la parola da cercare) e un intero (il numero di occorrenze della parola cercata).
Nel client viene chiamata l'API tramite la Call(), poiché è sincrona e bisogna aspettare un input dell'utente. La chiamata manda come input un inserimento dell'utente preso con scanf.
A questo punto viene implementato il vero e proprio grep distribuito, creando N goroutine che sezionano il file in cui vogliamo cercare la stringa in N batch e che comunicano con il thread master attraverso N channel contenuti in una slice.

MAP-REDUCE

Nel momento in cui viene passato il controllo ai worker, questi si occupano di eseguire l'implementazione del paradigma map-reduce utilizzato per il calcolo dei risultati.
Le fasi attraversate sono tre:

1. **Map**: per ciascuna riga del file si itera sulle singole parole ed appena viene rivelata la parola cercata, si crea una slice composta da due elementi, la parola stessa e la riga in cui è stata trovata.
2. **Shuffle-and-sort**: gli elementi vengono raggruppati per chiave all'interno di una struttura dati di Go, la *map*, in cui la key sarà appunto la parola cercata (l'elemento 0 della slice nella fase di map) ed il value sarà la lista delle righe associate a quella chiave.
3. **Reduce**: viene generata una nuova map che invece di avere come value una lista di stringhe, ha una stringa unica che è la concatenazione di tutte le righe restituite, separate da "\n".

Abbiamo anche aggiunto un'ulteriore funzionalità alla grep, che oltre a restituire le righe che contengono la parola cercata, indica anche il numero di occorrenze di quel termine all'interno del file.
Le tre fasi del paradigma sono molto simili a quelle precedentemente descritte, con la differenza che nella fase di map alla key viene associato il valore "1", nella fase di shuffle-amd-sort viene creato un array di "1" e nella fase di reduce questi vengono sommati, andando così a determinare il valore totale.

DEPLOYMENT

Abbiamo deciso, per arricchire la nostra esperienza, di caricare il server su una macchina ec2 mini idonea al piano AWS gratuito, in modo da poter chiamare l'API da qualunque macchina attraverso l'indirizzo IP pubblico passato nella funzione Dial().