WORD COUNTER MAP REDUCE

ANDREA DI IORIO

In questo progetto ho realizzato un app in Go per contare le occorrenze di ogni parola in maniera concorrente ed efficientemente, mediate la divisione del file in blocchi e assegnazione dei lavori in uno schema master-worker mediante chiamate RPC.

La computazione è divisa in 3 fasi: Map, Shuffle&Sort, Reduce.

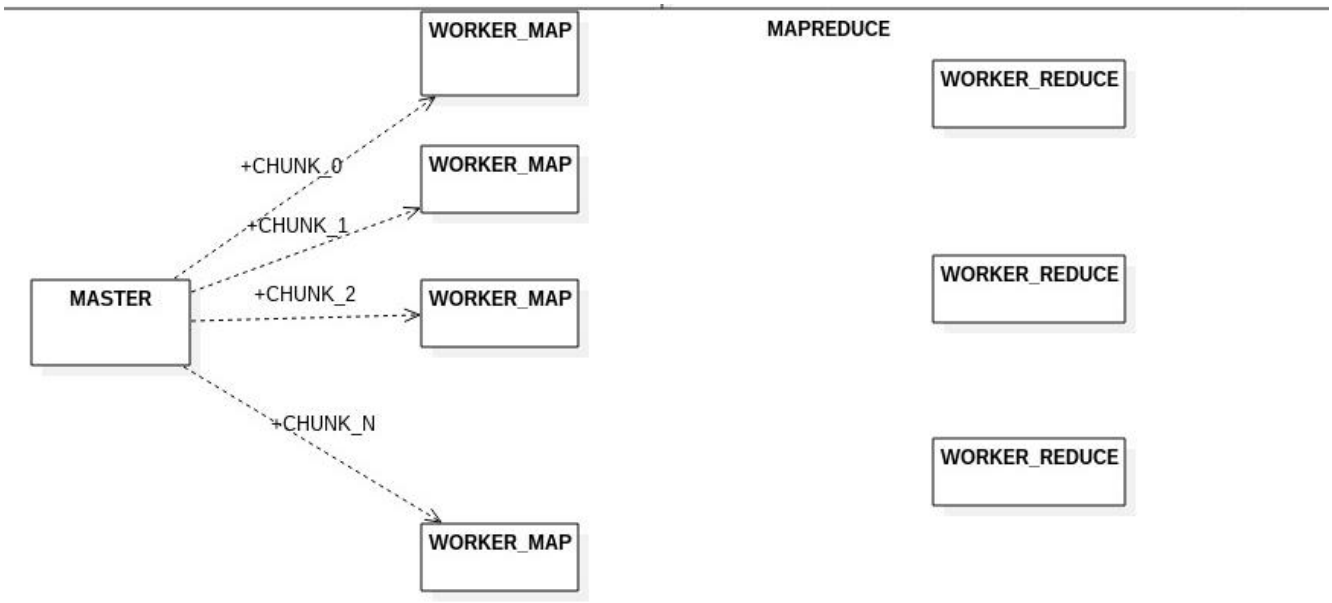
ARCHITETTURA

L'architettura si ispira al framework brevettato di Google MapReduce ed è strutturata su più thread che simulano nodi di rete di un cluster.

I nodi possono essere Worker di tipo Map, Worker di tipo Reduce o Master

1. **MAP**

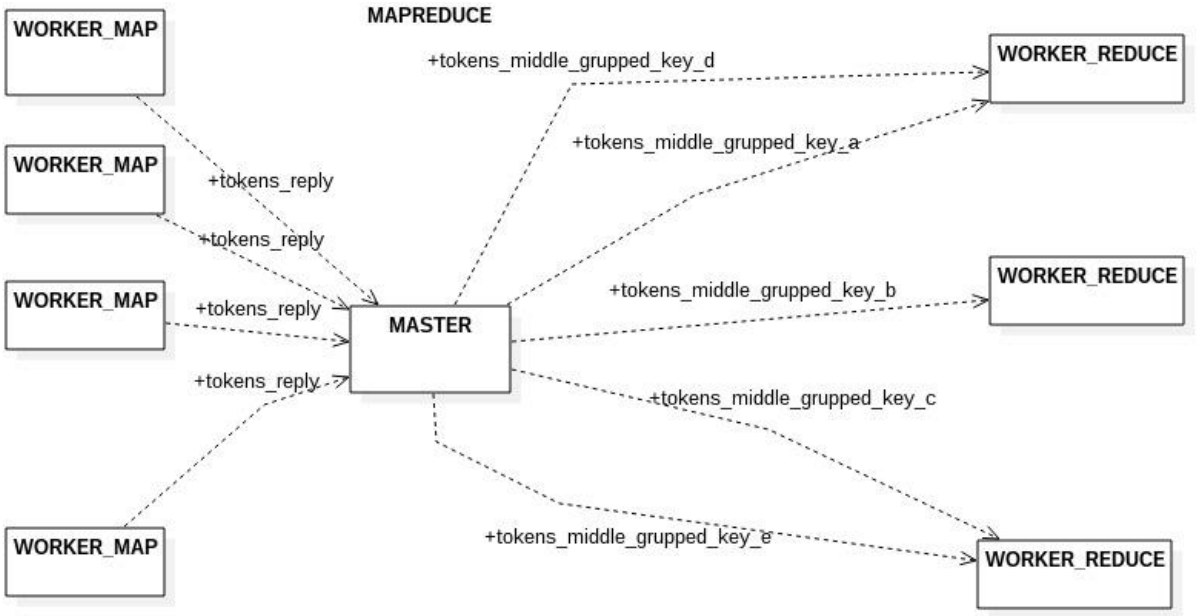
Il Master, dopo aver diviso il file in blocchi (leggendo ogni file con un thread), assegna un chunk ad ogni worker per l'operazione Map, il quale effettuerà il conteggio delle parole all'interno del suo blocco assegnato mediate l'uso di una *Hashmap.*



Il Master al termine della fase di assegnazione mediante RPC asincrona, attenderà il risultato delle operazioni Map.

2. SHUFFLE & SORT

Questi risultati parziali dalla fase di Map vengono riorganizzati nella fase di **Shuffle&Sort**, raggruppando le chiavi delle Map e concatenando i valori relativi mediate le built-in di Go sulle *map*.



3. REDUCE

Infine nella fase di Reduce, viene assegnata ogni coppia chiave-[valore$_1$,…,valore$_n$] (dove la chiave è una parola, e i valori sono i risultati delle Map relativi alla chiave, nello schema rappresentato mediante *tokens_middle_grupped_key* ..) ad un Worker assegnato all'operazione di Reduce mediate RPC asincrona.

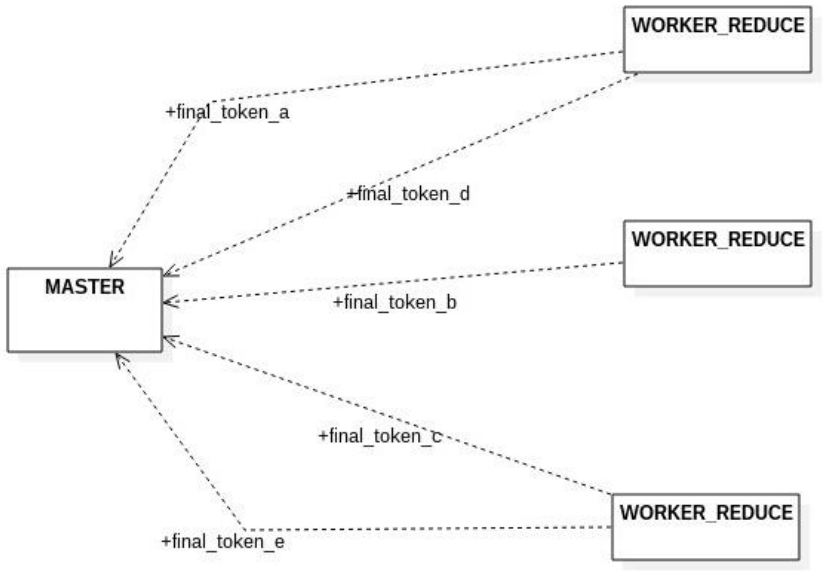
Tutti i worker RPC vengono avviati su thread in una fase iniziale, in particolare vengono avviati il massimo tra il numero di Worker per la Map e per la Reduce.

Al termine della fase di Map, vengono terminati i Worker superflui per liberare risorse.

Grazie alle RPC asincrone il lavoro viene prima assegnato a tutti e dopo viene raccolto il risultato da ogni worker, in questa maniera la computazione è effettivamente concorrente sui worker.

I risultati dell'operazione di Reduce saranno coppie chiave-valore, dove la chiave è una parola e il valore è il conteggio finale di tutte le occorenze di tale parola

(nello schema rappresentati mediante *final_token_...).*



UTILIZZO

Il codice è compilabile semplicemente tramite il comando go build nella directory mapReduce settando la variabile d'ambiente GOPATH a tale cartella.

Alternativamente mediate il Makefile, presente in tale cartella, è possibile accorpare queste operazioni semplicemente mediate il comando make build

Passando da riga di comando all'eseguibile dei filenames di file di tipo *plain-text* verrà eseguito il conteggio delle parole contenute nei file indicati.

all'interno della cartella txtSrc sono presenti alcuni file di tale tipo, presi dal sito *https://www.gutenberg.org/*

Il risultato finale verrà salvato in file denominato finalTokens.txt

E’ presente anche un semplice caso di test dove viene confrontato il risultato ottenuto dell’app con una versione del conteggio delle parole eseguita su un singolo thread.

A causa della divisione in Chunk alcune parole possono essere divise nell’assegnazione dei lavori di Map, quindi vengono tollerate un piccolo numero di discrepanze tra i 2 risultati (con N divisioni è possibile avere 2N + N parole mal conteggiate ed N parole conteggiate ma non presenti nel testo sorgente… dato che N è molto piccolo rispetto al numero delle parole complessive, statisticamente questo non inficia il risultato finale)

CONFIGURAZIONI

nel file config.json è possibile configurare

- il numero di worker da assegnare rispettivamente alla fase di Map e Reduce.
- la porta base, da cui vengono calcolate le porte da assegnare agl’altri worker, mediante offset progressivo, nella fase di inizializzazione
- un flag per indicare se è necessario ordinare il risultato finale in ordine decrescente per valore