Progettazione di un sistema MapReduce per il WordCount in C

Lorenzo Pio Virecci Fano Gabriele Franchina

Sommario

Obiettivo del progetto

Descrizione della traccia

- Il progetto richiede la realizzazione di un sistema di rete in linguaggio C che implementi l'algoritmo di WordCount utilizzando il paradigma MapReduce, applicato al testo *"Il Signore degli Anelli"*.

Scopo

- Implementare un sistema MapReduce in C.
- Calcolare il conteggio delle parole (WordCount) su un testo di grandi dimensioni quale *"Il Signore degli Anelli"*.

Tecnologie utilizzate

Strumenti e librerie

- Linguaggio C
- Programmazione modulare: sistema basato su più moduli autonomi
- Processi concorrenti
- Thread POSIX e mutex per sincronizzazione nel Reducer
- Comunicazione inter-process tramite FIFO (named pipes)

Cos'è MapReduce

Definizione

MapReduce è un paradigma di programmazione distribuita, introdotta nel 2004 da Google per l'elaborazione di grandi dataset in maniera scalabile e fault-tolerant.

Fasi principali

- **Map**: elaborazione parallela dei dati.
- **ShuffleSort**: aggregazione parziale dei dati per chiave.
- **Reduce**: aggregazione dei risultati parziali.

Perché utilizzare MapReduce

Vantaggi

- Parallelismo naturale e scalabilità.
- Gestione efficiente di dataset di grandi dimensioni.
- Modularità e replicabilità dei task.

Applicazione in contesti reali

- MapReduce è alla base di framework come Hadoop.
- Ideale per sistemi distribuiti o cluster di macchine.
- Permette di astrarre la complessità della sincronizzazione e comunicazione.

Architettura del progetto

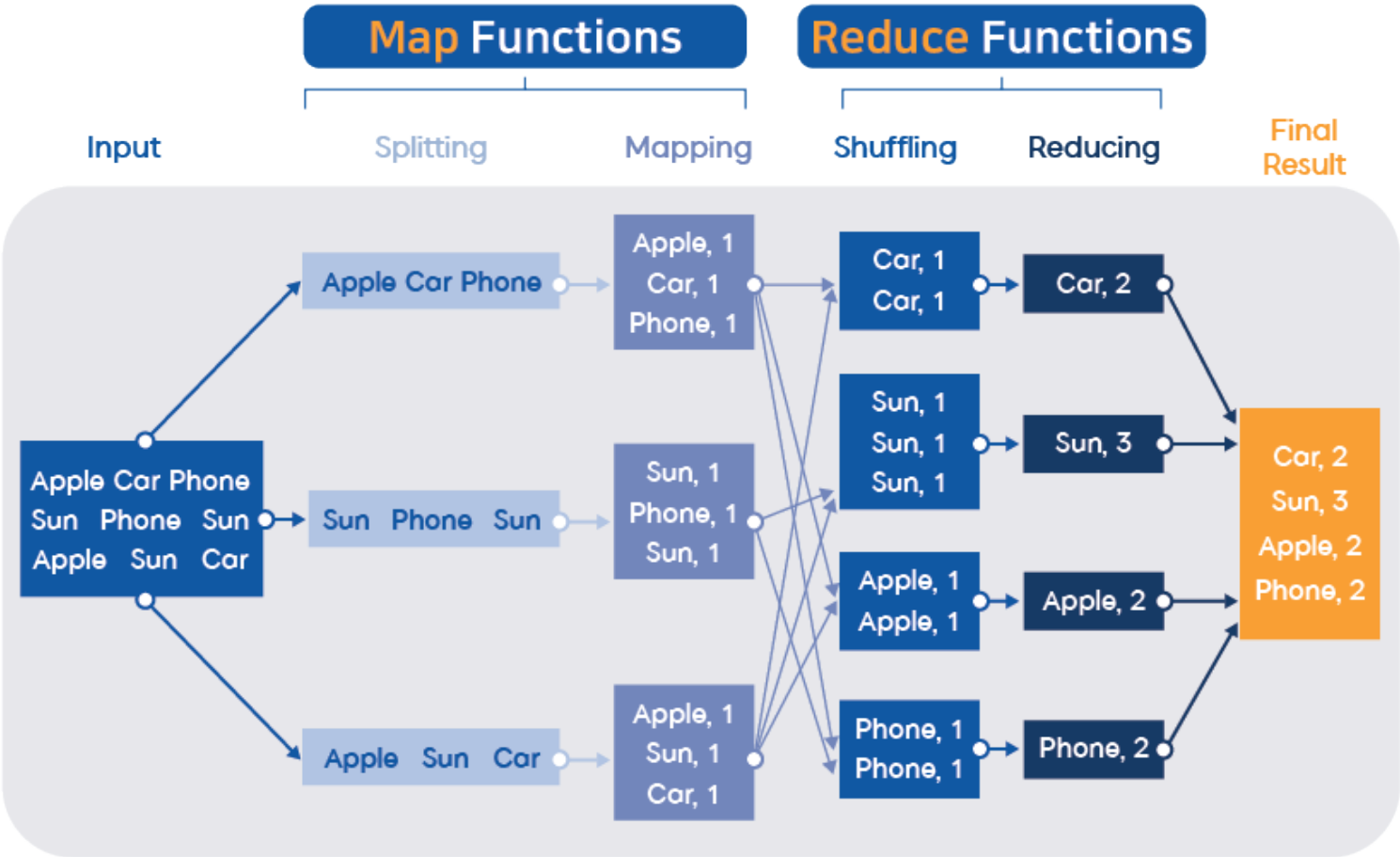
Struttura generale

- Processo Master: coordina l'intero flusso e si occupa della suddivisione dell'input in chunk.
- Mapper: worker paralleli che elaborano i chunk.
- Reducer: aggrega i risultati dei Mapper in output finale.

Comunicazione: inter-process

- Comunicazione tramite FIFO (named pipes) su singola macchina.
- Garantisce ordine e sincronizzazione dei messaggi.
- Evita duplicazioni: i Mapper leggono direttamente dal file i chunk assegnati.

Architettura del sistema



File di Configurazione del sistema

common.h

Le definizioni condivise tra Master, Mapper e Reducer sono contenute nel file *common.h*. Queste definizioni garantiscono coerenza tra tutti i processi e semplificano la manutenzione del codice.

Codice

```c
#ifndef COMMON_H
#define COMMON_H

#define NUM_MAPPERS 4

/* Path per le FIFO */
#define MASTER_TO_MAPPER_FIFO "/tmp/master_to_mapper_%d"
#define MAPPER_TO_REDUCER_FIFO "/tmp/mapper_%d_to_reducer"

#define MAX_WORD_LEN 64
#define MAX_LINE_LEN 1024

/* Struttura per i dati del chunk */
typedef struct {
    long offset;
    long size;
} ChunkData;

#endif
```

Ruolo del Master

Definizione

Il Master è il coordinatore principale del framework: si occupa della fase di splitting dell'input, della distribuzione dei chunk e della gestione dei processi.

Funzioni principali

- Suddivide il file in chunk.
- Avvia Mapper e Reducer.
- Invia informazioni sui chunk ai Mapper.
- Gestisce sincronizzazione e attesa dei processi.

Struttura Master

Suddivisione Input in chunk

```
if (stat("data/lotr.txt", &st) < 0) {
    perror("stat");
    exit(EXIT_FAILURE);
}

file_size = st.st_size;
chunk_size = file_size / NUM_MAPPERS;

printf("[Master] File size: %ld bytes, chunk size: %ld\n\n", file_size, chunk_size);
```

Struttura Master

Invio chunk ai mapper

```
    for (int i = 0; i < NUM_MAPPERS; i++) {
        snprintf(fifo_path, sizeof(fifo_path), MASTER_TO_MAPPER_FIFO, i);

        int fifo_fd = open(fifo_path, O_WRONLY);
        if (fifo_fd < 0) {
            perror("open master_to_mapper FIFO");
            continue;
        }

        ChunkData chunk;
        chunk.offset = i * chunk_size;
        chunk.size = (i == NUM_MAPPERS - 1)
                     ? (file_size - chunk.offset)
                     : chunk_size;

        ssize_t bytes_written = write(fifo_fd, &chunk, sizeof(ChunkData));
        if (bytes_written != sizeof(ChunkData)) {
            perror("write chunk to FIFO");
        }

        close(fifo_fd);

        if (i == 0) printf("\n");
        printf("[Master] Inviato chunk %d: offset= %ld, size= %ld\n", i, chunk.offset,
chunk.size);
    }

    printf("\n");
```

Ruolo del Mapper

Definizione

Il Mapper rappresenta il componente di elaborazione parallela del framework MapReduce. Il suo compito è leggere una porzione del file di input assegnata dal Master, eseguire il calcolo locale del WordCount e inviare i risultati parziali al Reducer. Ogni Mapper opera in maniera indipendente, consentendo di sfruttare il parallelismo a livello di processo.

Funzioni principali

- Ricezione informazioni sui chunk dal Master.
- Elaborazione parallela della propria porzione di file.
- Invio dei conteggi parziali al Reducer tramite FIFO.

Struttura Mapper

Ricezione informazioni chunk

```
snprintf(fifo_path, sizeof(fifo_path), MASTER_TO_MAPPER_FIFO, mapper_id);

int fifo_fd = open(fifo_path, O_RDONLY);
if (fifo_fd < 0) {
    perror("open master_to_mapper FIFO");
    exit(EXIT_FAILURE);
}

ChunkData chunk;
if (read(fifo_fd, &chunk, sizeof(ChunkData)) != sizeof(ChunkData)) {
    perror("read chunk");
    close(fifo_fd);
    exit(EXIT_FAILURE);
}
close(fifo_fd);
```

Struttura Mapper

Elaborazione parallela del proprio chunk

```c
typedef struct WordCount {
    char word[MAX_WORD_LEN];
    int count;
    struct WordCount *next;
} WordCount;

WordCount *head = NULL;

WordCount *find_word(const char *word) {
    WordCount *curr = head;
    while (curr) {
        if (strcmp(curr->word, word) == 0)
            return curr;
        curr = curr->next;
    }
    return NULL;
}

void add_word(const char *word) {
    WordCount *wc = find_word(word);
    if (wc) {
        wc->count++;
    } else {
        wc = malloc(sizeof(WordCount));
        strncpy(wc->word, word, MAX_WORD_LEN);
        wc->word[MAX_WORD_LEN - 1] = '\0';
        wc->count = 1;
        wc->next = head;
        head = wc;
    }
}
```

Struttura Mapper

Invio ai reducer

```
snprintf(fifo_path, sizeof(fifo_path), MAPPER_TO_REDUCER_FIFO, mapper_id);
    fifo_fd = open(fifo_path, O_WRONLY);
    if (fifo_fd < 0) {
        perror("open mapper_to_reducer FIFO");
        exit(EXIT_FAILURE);
    }

    FILE *stream = fdopen(fifo_fd, "w");
    if (!stream) {
        perror("fdopen");
        close(fifo_fd);
        exit(EXIT_FAILURE);
    }

    WordCount *curr = head;
    while (curr) {
        fprintf(stream, "%s %d\n", curr->word, curr->count);
        curr = curr->next;
    }
```

Ruolo del Reducer

Definizione

Il Reducer rappresenta la fase di aggregazione finale del paradigma MapReduce. Il suo compito è raccogliere i risultati parziali prodotti dai Mapper, sommare i conteggi relativi a ciascuna parola e produrre l'output finale del WordCount.

Funzioni principali

- Thread per ogni Mapper.
- Mutex per sincronizzazione accesso lista globale di WordCount.
- Produzione del file `report.txt` con conteggio aggregato.

Struttura Reducer

Thread per ogni mapper

```c
pthread_t threads[NUM_MAPPERS];

/* Crea thread per ogni mapper */
for (int i = 0; i < NUM_MAPPERS; i++) {
    ThreadArgs *args = malloc(sizeof(ThreadArgs));
    args->mapper_id = i;
    if (pthread_create(&threads[i], NULL, process_mapper, args) != 0) {
        perror("pthread_create");
        free(args);
        exit(EXIT_FAILURE);
    }
}
```

Struttura Reducer

Struttura dati e counter globale

```c
/* Linked list globale */
typedef struct WordCount {
    char word[MAX_WORD_LEN];
    int count;
    struct WordCount *next;
} WordCount;

WordCount *global_head = NULL;
pthread_mutex_t list_mutex = PTHREAD_MUTEX_INITIALIZER;
int total_words = 0;

typedef struct {
    int mapper_id;
} ThreadArgs;

/* Cerca parola nella lista globale */
WordCount *find_word_unsafe(const char *word) {
    WordCount *curr = global_head;
    while (curr) {
        if (strcmp(curr->word, word) == 0)
            return curr;
        curr = curr->next;
    }
    return NULL;
}
```

Struttura Reducer

Struttura dati e counter globale

```
void add_word_safe(const char *word, int count) {
    pthread_mutex_lock(&list_mutex);

    WordCount *wc = find_word_unsafe(word);
    if (wc) {
        wc->count += count;
    } else {
        wc = malloc(sizeof(WordCount));
        strncpy(wc->word, word, MAX_WORD_LEN);
        wc->word[MAX_WORD_LEN - 1] = '\0';
        wc->count = count;
        wc->next = global_head;
        global_head = wc;
    }

    total_words += count;
    pthread_mutex_unlock(&list_mutex);
}
```

Scenario di test

Configurazione

- Input: testo completo de *Il Signore degli Anelli*.
- Esecuzione su singola macchina (localhost).

Esecuzione in ambiente WSL

- make
- ./master

Output finale

File `report.txt` contenente tutti i conteggi aggregati.

Scenario di test

Visualizzazione output

```
[Master] File size: 3262595 bytes, chunk size: 815648

Avvio reducer...

Avvio mapper 0...
Avvio mapper 1...
Avvio mapper 2...
Avvio mapper 3...


[Master] Inviato chunk 0: offset= 0, size= 815648
[Master] Inviato chunk 1: offset= 815648, size= 815648
[Master] Inviato chunk 2: offset= 1631296, size= 815648
[Master] Inviato chunk 3: offset= 2446944, size= 815651

Numero parole contate in totale: 582115 parole
Numero parole uniche contate: 14085 parole

Report counter in report.txt

[Master] Esecuzione completata
```

Conclusioni

Risultati raggiunti

- Sistema MapReduce funzionante in C.
- Approfondimento su comunicazione IPC, processi concorrenti e sincronizzazione.
- Analisi delle prestazioni in ambiente locale.

Possibili evoluzioni

- Bilanciamento dinamico dei chunk.
- Esecuzione su macchine distribuite reali.
- Strutture dati più efficienti nel Reduce.