Progetto B3: Distributed PageRank

Sistemi Distribuiti e Cloud Computing

Alessandro Lioi, 0333693

Università di Roma Tor Vergata

A.A. 2022/2023

Indice

Contesto

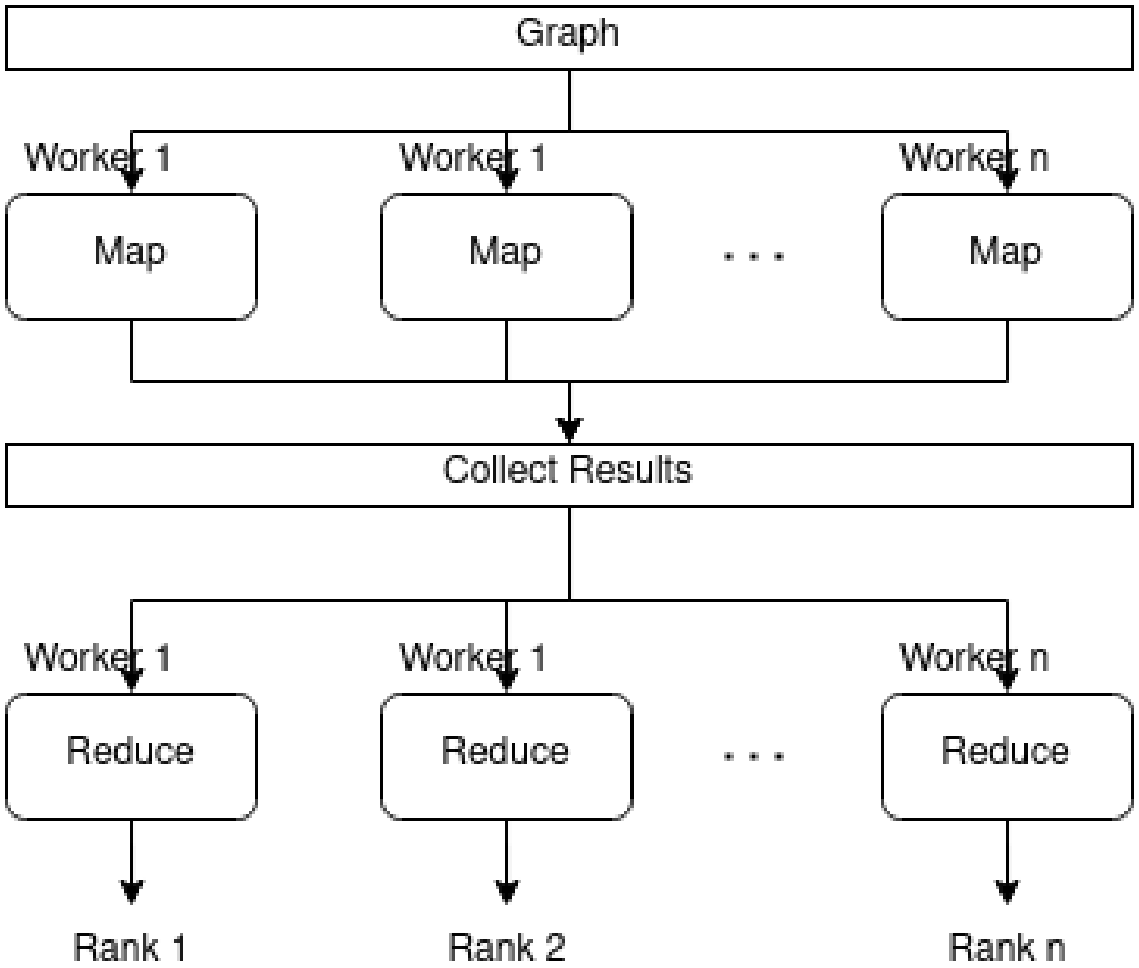
Lo scopo del progetto è implementare l'algoritmo del PageRank in maniera distribuita

Regola di aggiornamento dei rank:

$$R_{i+1}(u) = c \sum_{v \in B_u} \frac{R_i(v)}{N_v} + (1 - c)E(u)$$

Può essere implementato con il pattern Map-Reduce

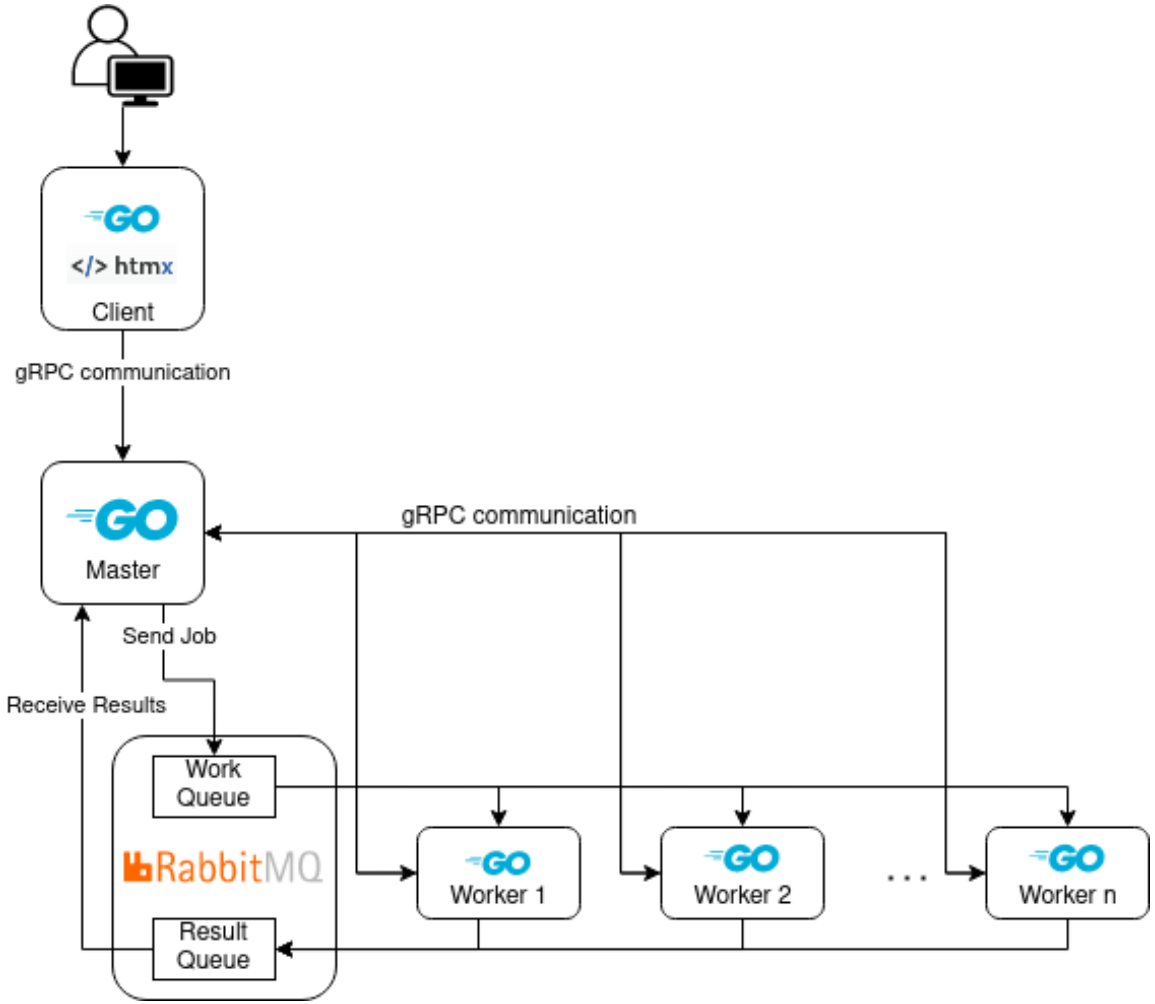
Map-Reduce



Si divide il grafo in sottografi, passati ognuno ad un worker che esegue la fase di Map

Si aggregano i vari risultati e si passano ai worker della fase di Reduce che computano il nuovo rank

Visione Generale



- Utente contatta il Client con un grafo o i parametri di generazione
- Client contatta il Master e richiede l'esecuzione del PageRank
- Master inizia la computazione e, una volta finita, invia i risultati al client

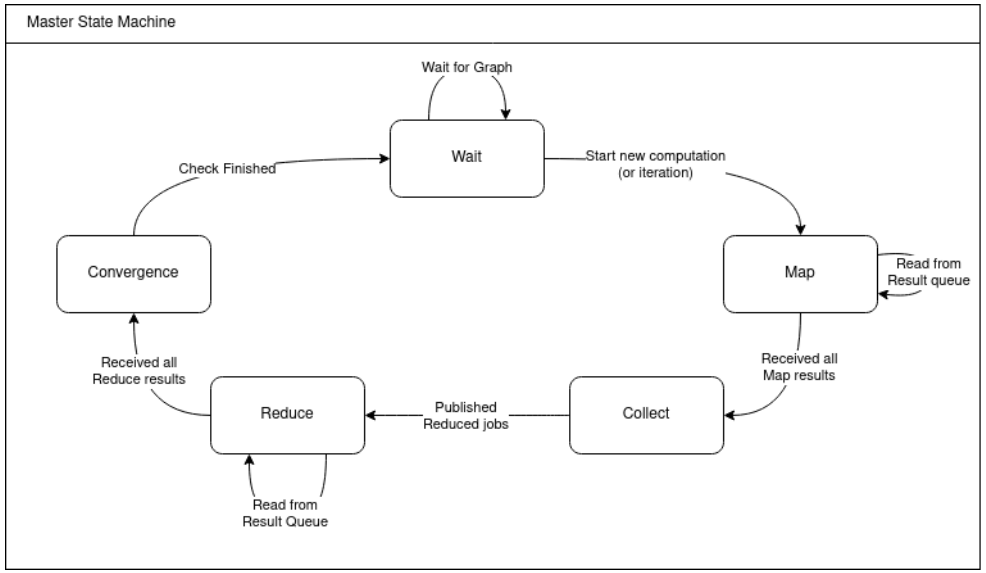
Client

Formato da due server:

- Echo: gestione richieste utente
- HTMX: richieste client-side al server Echo

- richiede una computazione al master
- resta in attesa dei risultati dal master

Master



- Wait: attesa computazione/pubblicazione Map Job su *Work Queue*
- Map: lettura somme parziali da *Result Queue*
- Collect: aggregazione risultati parziali e pubblicazione Reduce Job sulla *Work Queue*
- Reduce: lettura nuovi rank da *Result Queue*
- Convergence: aggiornamento rank e decisione se iterare o terminare

Senza worker nella rete si esegue una versione dell'algoritmo su singolo nodo

Invia il suo stato ai vari worker nella fase di Wait, a seguito di una nuova iterazione

Worker e Message Queue

Worker:

- leggono job dalla *Work Queue*
- eseguono computazione in base al tipo di job
- pubblicano risultati sulla *Result Queue*
- periodicamente eseguono *Health Check* con il master

Message Queue:

- uso di RabbitMQ come message broker
- *Work Queue*: messaggi scritti dal master e letti dai worker
- *Result Queue*: messaggi scritti dai worker e letti dal master

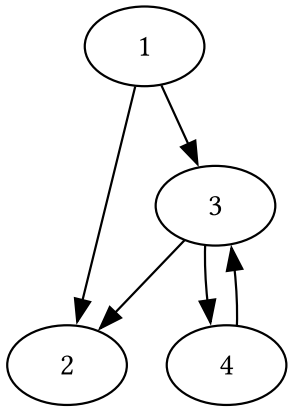
Grafo: caricamento da file

È possibile caricare un file nel formato: `FromNode ToNode`

Esempio di file

```
# From To
  1    2
  1    3
  3    2
  3    4
  4    3
```

Grafo associato



Grafo: generazione casuale

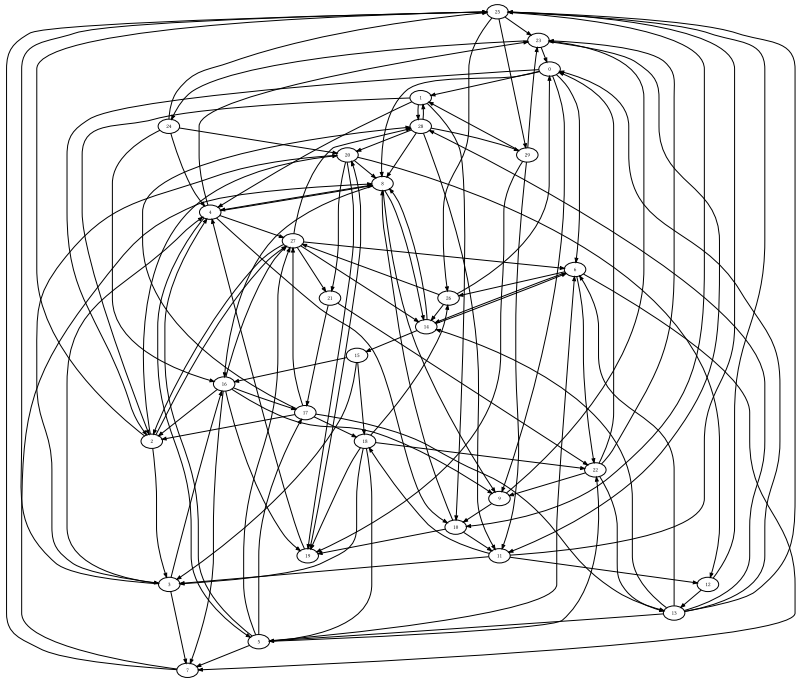
Basata su due parametri:

- numero di nodi
- massimo numero di archi di un nodo

Generazione:

- per ogni nodo, si scelgono `maxNumberOfEdges` nodi con cui il nodo deve essere collegato, escludendo connessioni con sè stessi
- per assicurare la connessione del grafo, si collega ogni nodo con il suo successivo

Esempio di grafo con 30 nodi e massimo 5 archi per nodo:



Grafo: struttura dati

- Grafo associa ID nodo (intero) a `GraphNode`
- `GraphNode` ha le informazioni per il nodo (rank, probabilità `e`) e gli archi entranti
- `GraphNodeInfo` ha le informazioni necessarie per la fase di Map

È richiesto pre-calcolare il valore di `outlinks` e aggiornare il `rank` in `GraphNodeInfo` ad ogni nuova assegnazione del rank

Non è necessario fare ulteriori operazioni nella fase di *Map*

```
// Graph
message State {
  // ...
  map<int32, GraphNode> graph = 2;
  // ...
}

message GraphNode {
  double rank = 1;
  double e = 2;
  map<int32, GraphNodeInfo> inLinks = 3;
}

message GraphNodeInfo {
  int32 outlinks = 1;
  double rank = 2;
}
```

Message Queue e Job

- Utilizzo di Protocol Buffers (riuso del messaggio `GraphNodeInfo`)
- `type` specifica il tipo di messaggio

```
message Job {
  int32 type = 1;
  map<int32, Map> mapData = 2;
  map<int32, Reduce> reduceData = 3;
}
message Map {
  map<int32, GraphNodeInfo> inLinks = 1;
}
message Reduce {
  double sum = 1;
  double e = 2;
}
message Result {
  map<int32, double> values = 1;
}
```

State Update e Worker Join

State Update

- nella fase di Wait ad ogni iterazione o inizio nuova computazione
- si inviano:
  - grafo
  - parametri `c` e `threshold` (impostati dal client)
  - informazioni di connessione con il client
  - numero di iterazione
  - lista di worker

Join

- controlla se esiste un master
- se non ottiene rispota, si identifica come master
- altrimenti, riceve dal master:
  - nomi delle code
  - stato del master
  - id univoco per il nodo

A seguito di un join di un worker, si invia un aggiornamento di stato ristretto alla lista dei worker

Health Check e Elezione Master

Health Check

- ogni *health_check* (parametro di configurazione), i worker inviano un messaggio al master
  - se è ancora in vita, risponde con un messaggio vuoto o l'ultimo stato
  - altrimenti, è richiesta una nuova elezione

Elezione Master: algoritmo Bully

- worker invia la propria candidatura agli altri nodi
- se riceve un ACK negativo, abbandona la candidatura e aspetta di ricevere la candidatura del worker con ID maggiore del suo
- altrimenti, con tutti ACK positivi si elegge nuovo master

- In caso di crash del master prima di essere contattato dal client, è impossibile recuperare l'indirizzo IP da contattare
- Non è possibile renderizzare il grafo se il numero di nodi è > 50 (la fase di rendering richiede troppo tempo).

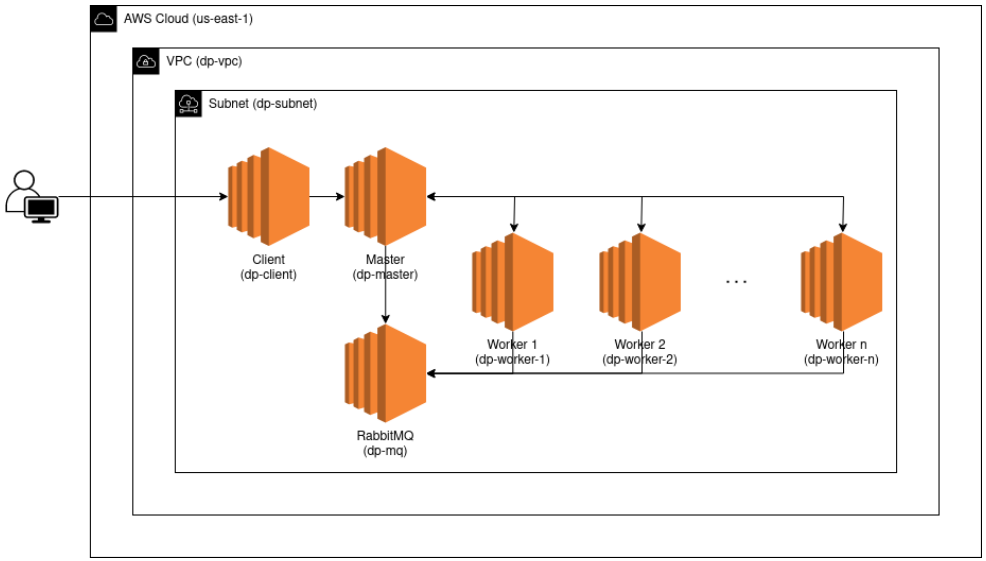
Esecuzione

Configurazione tramite `config.json`

Locale: tramite script in `deploy/local.py`

Docker Compose: tramite script in `deploy/compose.py` ed esecuzione con `docker compose up`

AWS: tramite script `deploy/aws.py`



- Client, Master, RabbitMQ e Workers su macchine EC2 (configurate tramite script bash), sulla stessa VPC e stessa subnet
- Comunicazione interna tramite indirizzi IP locali
- Client accessibile tramite indirizzo IP pubblico

Repository

Repository contenente il codice sorgente

https://github.com/lioia/distributed-pagerank