Multicast totalmente e causalmente ordinato in Go

Cristiano Cuffaro

Università degli studi di Roma Tor Vergata

cristiano.cuffaro@outlook.com

18 Novembre 2021

Overview

Descrizione del progetto

Applicazione distribuita in Go

- Servizio di registrazione
- Supporto dei seguenti algoritmi:
  - Multicast totalmente ordinato implementato in modo centralizzato tramite un sequencer
  - Multicast totalmente ordinato implementato in modo decentralizzato tramite l'uso di clock logici scalari
  - Multicast causalmente ordinato implementato in modo decentralizzato tramite l'uso di clock logici vettoriali

Assunzioni di base

- Almeno due peer partecipano al gruppo di comunicazione multicast
- Membership statica durante l'esecuzione dell'applicazione
- Ritardo di trasmissione di un messaggio impredicibile, ma finito
- Comunicazione FIFO ordered e affidabile
  - I messaggi non vengono persi e/o duplicati
  - Non vi è la presenza di messaggi spuri

Architettura

Microservizi

- `peer_service`
  - riceve ed invia messaggi multicast
- `registration_service`
  - registrazione dei peer $\Rightarrow$ peer discovery
- `sequencer_service`
  - coordina la comunicazione tra i peer

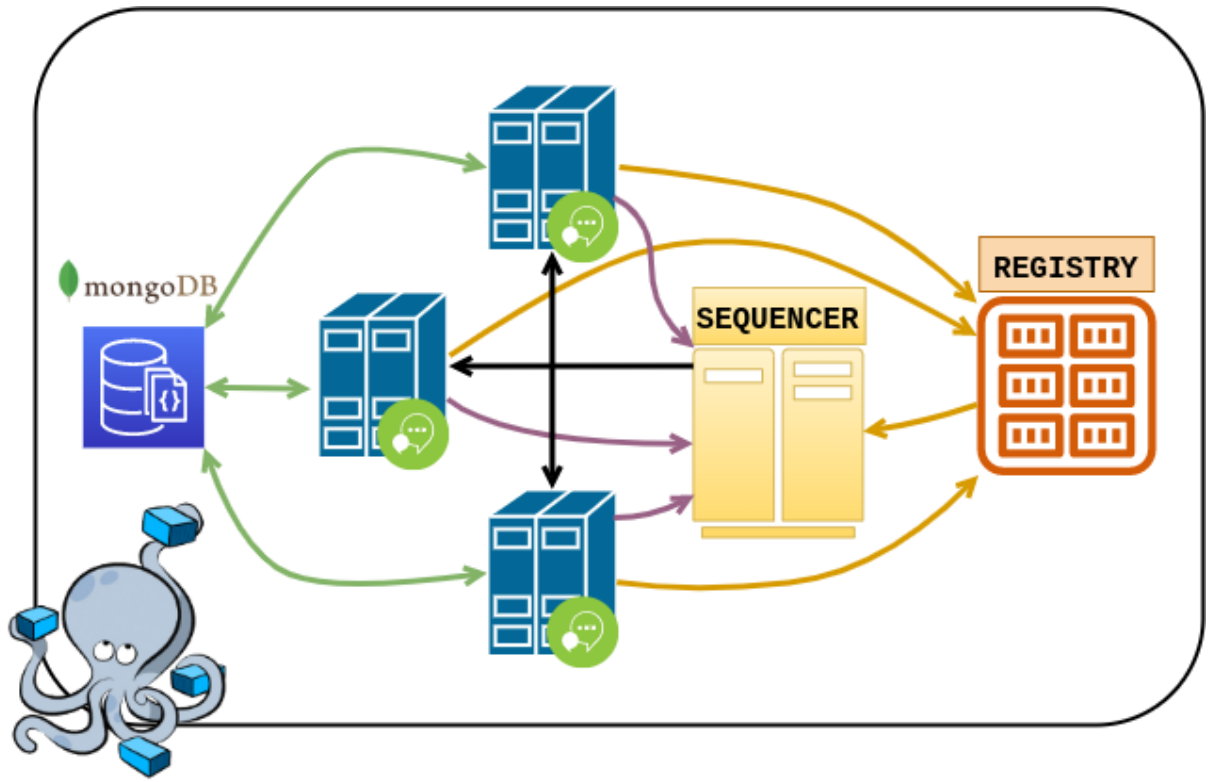
Datastore (Database NoSQL)

- `MongoDB`
  - orientato ai documenti
  - memorizza i messaggi consegnati a ciascun peer

Architettura centralizzata

Schema di interazione nel caso di:

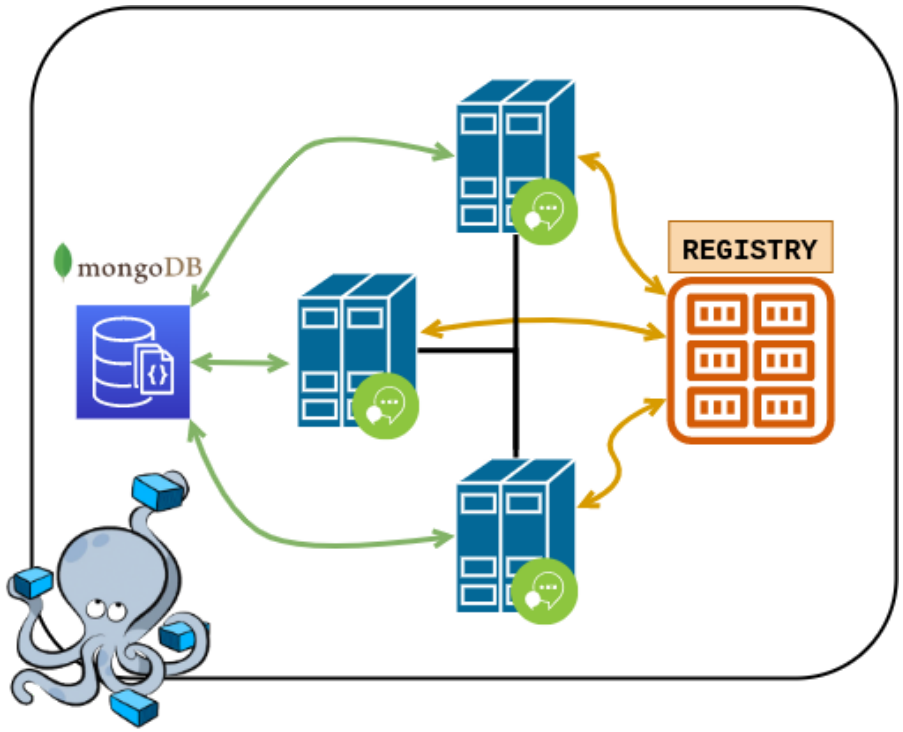
- Algoritmo di multicast totalmente ordinato implementato in modo centralizzato



Architettura decentralizzata

Schema di interazione nel caso di:

- Algoritmo di multicast totalmente ordinato implementato in modo decentralizzato
- Algoritmo di multicast causalmente ordinato implementato in modo decentralizzato



Deployment e configurazione del sistema

Compose file: `yml`

- Attributi particolari:
  - `profiles`
  - `scale`
  - `depends_on`

    \+

    `wait-for-it.sh`

Environment: `.env`

- Parametri di rete
  - `REGISTRATION_PORT`
  - `SEQUENCER_PORT`
  - `MULTICAST_PORT`
  - `CONTROL_PORT`
- Parametri applicativi
  - `MEMBERS_NUM`
  - `DELAY`
  - `MULTI_ALGO`

Ottimizzazione dei Dockerfile

Multi-stage build

È un approccio che consente di ottimizzare i Dockerfile, aumentandone la leggibilità e la manutenibilità, per ottenere immagini di piccola taglia.

```
▶ docker images
REPOSITORY              TAG       IMAGE ID       CREATED         SIZE
sequencer_service       latest    29c6a79e1c2b   2 days ago      15.6MB
peer_service            latest    f668a0718702   2 days ago      24.8MB
registration_service    latest    680c895cb86f   2 days ago      11.2MB
```

Comunicazione tra i servizi: RPC

Metodi esposti:

- `RegisterMember`
  - esposto da: `registration_service`
  - registra al gruppo di comunicazione e ne restituisce la definizione
- `RetrieveMembership`
  - esposto da: `registration_service`
  - restituisce la definizione del gruppo di comunicazione
- `SendInMulticast`
  - esposto da: `sequencer_service`
  - inoltra un messaggio di multicast a tutti i peer
- `ReceiveMessage`
  - esposto da: `peer_service`
  - permette al peer di ricevere un messaggio di multicast

Caratteristiche della comunicazione

- Connessioni aperte una sola volta
- Gestione connessioni con `RpcHandler`
- Utilizzo dei canali di Go
- Chiusura del servizio di registrazione

Accept VS acceptn

Official `Accept`

https://cs.opensource.google/go/go/+/refs/tags/go1.17.3:src/net/rpc/server.go;l=623

```
func (server *Server) Accept(lis net.Listener) {
        for {
                conn, err := lis.Accept()
                if err != nil {
                        log.Print("rpc.Serve: accept:", err.Error())
                        return
                }
                go server.ServeConn(conn)
        }
}
```

Reimplementazione per accettare fino a *n* richieste

```
func acceptn(server *rpc.Server, lis net.Listener, n int) {
        var wg sync.WaitGroup

        wg.Add(n)
        for i := 0; i < n; i++ {
                go func() {
                        conn, err := lis.Accept()
                        if err != nil {
                                utils.ErrorHandler("Accept", err)
                        }
                        server.ServeConn(conn)
                        wg.Done()
                }()
        }
        wg.Wait()
}
```

Gestione messaggi di multicast (1)

Strutture dati core

- Messaggio di multicast

```
type Message struct {
    ID          uint64
    Host        string
    Content     string
    Timestamp   []uint64
    Type        string
}
```

- Tempo logico

```
type Time struct {
    Clock   []uint64
    Lock    sync.Mutex
}
```

Strutture per la gestione dei messaggi in coda

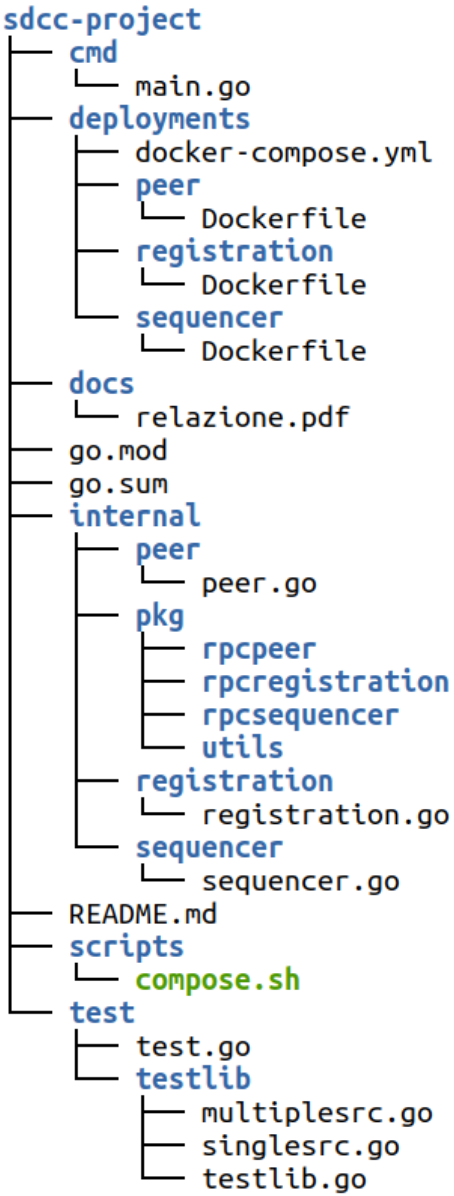
- queue: coda di messaggi
- `ackForMessages`: #ack ricevuti per ciascun messaggio
  - `"hostname:timestamp"` identifica il messaggio
- `messagesPerPeer`: #messaggi in coda ricevuti da ciascun peer

Gestione messaggi di multicast (2)

Utilizzo del datastore

- Formato BSON (Binary JSON)
- Marshaling e unmarshaling dei dati
- Struttura `datastoreHandler`
- Volume *Docker-managed*

Layout di progetto



- Standard non ufficiale

  `https://github.com/golang-standards/project-layout`

- Espandibilità del codice
- Riusabilità del codice
- Utilizzo di *Go modules*
  - `go.mod`
  - `go.sum`

Descrizione dei casi di test (1)

Sorgente singola

- Il peer che invia i messaggi è soltanto uno
- Vengono inviati dieci messaggi corrispondenti ai numeri da 0 a 9
- `Test PASSED` $\Leftrightarrow$ tutti i partecipanti ricevono i messaggi nello stesso ordine

Sorgente multipla - multicast totalmente ordinato

- In maniera ciclica, ciascun messaggio viene fatto inviare ad un peer differente
- Vengono inviati dieci messaggi corrispondenti ai numeri da 0 a 9
- `Test PASSED` $\Leftrightarrow$ tutti i partecipanti ricevono i messaggi nello stesso ordine

Descrizione dei casi di test (2)

Sorgente multipla - multicast causalmente ordinato

- Due peer riproducono il seguente dialogo:
  1. `catOwner:` "Oh no! My cat just jumped out the window."
  2. `catOwner:` "Whew, the catnip plant broke her fall."
  3. `friend:` "I love when that happens to cats!"
- Tutti gli altri peer inviano messaggi *dummy*
- `Test PASSED` $\Leftrightarrow$ tutti i partecipanti ricevono i messaggi relativi al dialogo in maniera ordinata

Esecuzione dei test (1)

Example (Multicast totalmente ordinato centralizzato)

- MEMBERS_NUM=4
- DELAY=5
- Sorgente singola

```
          Test started!
          All messages have been sent, wait for propagation...

          Test results:  Test PASSED!
************ user0 *************        ************ user1 *************
<0>          [user0] 0                  <0>          [user0] 0
<1>          [user0] 1                  <1>          [user0] 1
<2>          [user0] 2                  <2>          [user0] 2
<3>          [user0] 3                  <3>          [user0] 3
<4>          [user0] 4                  <4>          [user0] 4
<5>          [user0] 5                  <5>          [user0] 5
<6>          [user0] 6                  <6>          [user0] 6
<7>          [user0] 7                  <7>          [user0] 7
<8>          [user0] 8                  <8>          [user0] 8
<9>          [user0] 9                  <9>          [user0] 9
************ user2 *************        ************ user3 *************
<0>          [user0] 0                  <0>          [user0] 0
<1>          [user0] 1                  <1>          [user0] 1
<2>          [user0] 2                  <2>          [user0] 2
<3>          [user0] 3                  <3>          [user0] 3
<4>          [user0] 4                  <4>          [user0] 4
<5>          [user0] 5                  <5>          [user0] 5
<6>          [user0] 6                  <6>          [user0] 6
<7>          [user0] 7                  <7>          [user0] 7
<8>          [user0] 8                  <8>          [user0] 8
<9>          [user0] 9                  <9>          [user0] 9
```

Esecuzione dei test (2)

Example (Multicast totalmente ordinato centralizzato)

- MEMBERS_NUM=4
- DELAY=5
- Sorgente multipla

```
Test started!
All messages have been sent, wait for propagation...

Test results:  Test PASSED!
************* user0 *************          ************* user1 *************
<0>           [user1] 1                    <0>           [user1] 1
<1>           [user3] 3                    <1>           [user3] 3
<2>           [user2] 2                    <2>           [user2] 2
<3>           [user0] 0                    <3>           [user0] 0
<4>           [user2] 6                    <4>           [user2] 6
<5>           [user1] 5                    <5>           [user1] 5
<6>           [user0] 4                    <6>           [user0] 4
<7>           [user1] 9                    <7>           [user1] 9
<8>           [user0] 8                    <8>           [user0] 8
<9>           [user3] 7                    <9>           [user3] 7
************* user2 *************          ************* user3 *************
<0>           [user1] 1                    <0>           [user1] 1
<1>           [user3] 3                    <1>           [user3] 3
<2>           [user2] 2                    <2>           [user2] 2
<3>           [user0] 0                    <3>           [user0] 0
<4>           [user2] 6                    <4>           [user2] 6
<5>           [user1] 5                    <5>           [user1] 5
<6>           [user0] 4                    <6>           [user0] 4
<7>           [user1] 9                    <7>           [user1] 9
<8>           [user0] 8                    <8>           [user0] 8
<9>           [user3] 7                    <9>           [user3] 7
```

Esecuzione dei test (3)

Example (Multicast totalmente ordinato decentralizzato)

- MEMBERS_NUM=4
- DELAY=5
- Sorgente singola

```
Test started!
All messages have been sent, wait for propagation...

Test results:  Test PASSED!

        ************* user0 *************

        ************* user1 *************

        ************* user2 *************

        ************* user3 *************
```

Esecuzione dei test (4)

Example (Multicast totalmente ordinato decentralizzato)

- MEMBERS_NUM=4
- DELAY=5
- Sorgente multipla

```
Test started!
All messages have been sent, wait for propagation...

Test results:  Test PASSED!
************* user0 *************          ************* user1 *************
<1>          [user3] 3                     <1>          [user3] 3
<1>          [user2] 2                     <1>          [user2] 2
<1>          [user1] 1                     <1>          [user1] 1
<1>          [user0] 0                     <1>          [user0] 0
<3>          [user3] 7                     <3>          [user3] 7

************* user2 *************          ************* user3 *************
<1>          [user3] 3                     <1>          [user3] 3
<1>          [user2] 2                     <1>          [user2] 2
<1>          [user1] 1                     <1>          [user1] 1
<1>          [user0] 0                     <1>          [user0] 0
<3>          [user3] 7                     <3>          [user3] 7
```

Esecuzione dei test (5)

Example (Multicast causalmente ordinato)

- `MEMBERS_NUM=4`
- `DELAY=5`
- Sorgente singola

```
Test started!
All messages have been sent, wait for propagation...

Test results:  Test PASSED!
```

```
************* user0 *************
<0,0,0,1>    [user0] 0
<0,0,0,2>    [user0] 1
<0,0,0,3>    [user0] 2
<0,0,0,4>    [user0] 3
<0,0,0,5>    [user0] 4
<0,0,0,6>    [user0] 5
<0,0,0,7>    [user0] 6
<0,0,0,8>    [user0] 7
<0,0,0,9>    [user0] 8
<0,0,0,10>   [user0] 9
```

```
************* user1 *************
<0,0,0,1>    [user0] 0
<0,0,0,2>    [user0] 1
<0,0,0,3>    [user0] 2
<0,0,0,4>    [user0] 3
<0,0,0,5>    [user0] 4
<0,0,0,6>    [user0] 5
<0,0,0,7>    [user0] 6
<0,0,0,8>    [user0] 7
<0,0,0,9>    [user0] 8
<0,0,0,10>   [user0] 9
```

```
************* user2 *************
<0,0,0,1>    [user0] 0
<0,0,0,2>    [user0] 1
<0,0,0,3>    [user0] 2
<0,0,0,4>    [user0] 3
<0,0,0,5>    [user0] 4
<0,0,0,6>    [user0] 5
<0,0,0,7>    [user0] 6
<0,0,0,8>    [user0] 7
<0,0,0,9>    [user0] 8
<0,0,0,10>   [user0] 9
```

```
************* user3 *************
<0,0,0,1>    [user0] 0
<0,0,0,2>    [user0] 1
<0,0,0,3>    [user0] 2
<0,0,0,4>    [user0] 3
<0,0,0,5>    [user0] 4
<0,0,0,6>    [user0] 5
<0,0,0,7>    [user0] 6
<0,0,0,8>    [user0] 7
<0,0,0,9>    [user0] 8
<0,0,0,10>   [user0] 9
```

Esecuzione dei test (6)

Example (Multicast causalmente ordinato)

- MEMBERS_NUM=4
- DELAY=5
- Sorgente multipla

```
    Test started!
    All messages have been sent, wait for propagation...

    Test results:  Test PASSED!

************* user0 *************
<0,0,0,1>    [user0] Oh no! My cat just jumped out the window.
<0,0,0,2>    [user0] Whew, the catnip plant broke her fall.
<1,0,0,0>    [user3] Dummy message!
<1,1,0,2>    [user1] I love when that happens to cats!
<0,0,1,1>    [user2] Dummy message!
************* user1 *************
<0,0,0,1>    [user0] Oh no! My cat just jumped out the window.
<1,0,0,0>    [user3] Dummy message!
<0,0,0,2>    [user0] Whew, the catnip plant broke her fall.
<1,1,0,2>    [user1] I love when that happens to cats!
<0,0,1,1>    [user2] Dummy message!
************* user2 *************
<0,0,0,1>    [user0] Oh no! My cat just jumped out the window.
<0,0,1,1>    [user2] Dummy message!
<1,0,0,0>    [user3] Dummy message!
<0,0,0,2>    [user0] Whew, the catnip plant broke her fall.
<1,1,0,2>    [user1] I love when that happens to cats!
************* user3 *************
<1,0,0,0>    [user3] Dummy message!
<0,0,0,1>    [user0] Oh no! My cat just jumped out the window.
<0,0,0,2>    [user0] Whew, the catnip plant broke her fall.
<0,0,1,1>    [user2] Dummy message!
<1,1,0,2>    [user1] I love when that happens to cats!
```

Thin client per l'utilizzo dell'applicazione

Features

- Associa uno username a ciascun peer
- Manda ai peer il contenuto dei messaggi da inviare
- Visualizza i messaggi ricevuti da ciascun peer
  - Disponibile solo in modalità *verbose*

```
Now you can send messages from each peer you want

************* Allowed Actions *************
1) Send a message from a participant
2) View messages received from a participant
3) Quit

Select an option
>>
```

Fine