Sistemi Distribuiti e cloud Computing

Algoritmi per l’elezione di un leader nei sistemi distribuiti

Introduzione

Elezione nei sistemi distribuiti:

- Un sistema distribuito è un insieme di computer indipendente che appaiono agli utenti del sistema come un singolo sistema coerente. Questi comunicano tra loro tramite la rete e coordinano le loro azioni scambiandosi messaggi.
- Nei sistemi distribuiti, molto spesso, è necessario che, all'interno dei processi che costituiscono il sistema ci sia un processo che svolga attività di leader. Per tale motivo, è essenziale avere degli algoritmi che garantiscano la presenza costante di un leader all'interno del sistema
- Ci concentreremo su due algoritmi di elezione ditribuita:
  - Algoritmo di Bully;
  - Algoritmo di Dolev-Klawe-Rodeh(DKR).

Assunzioni per il funzionamento:

- Ci sono n processi che comunicano tra loro;
- La comunicazione è affidabile;
- Processi soggetti a fallimenti;
- Ogni processo ha un Id unico e il processo non guasto con l'ID più alto è il processo che dovrà essere eletto come leader;

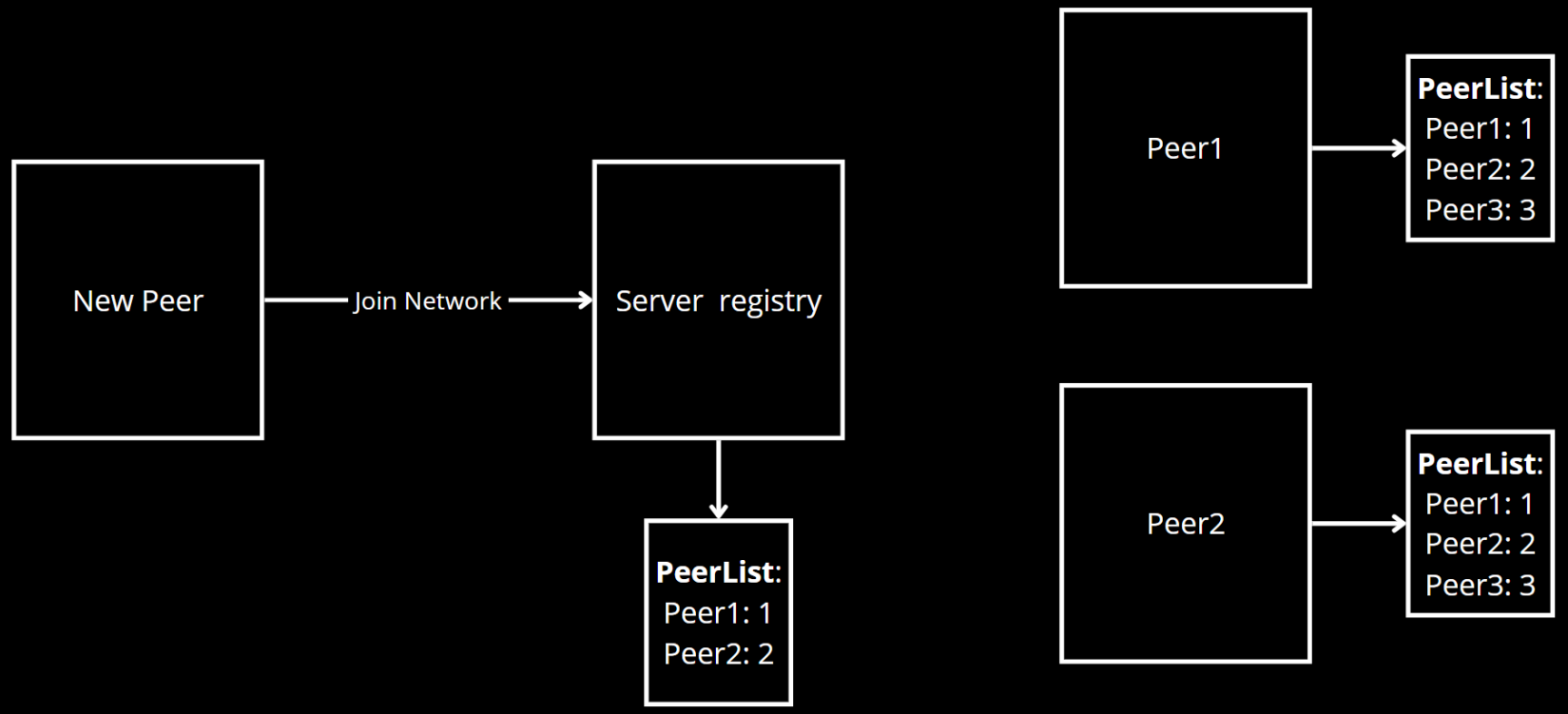
Server Registry

Server registry

- Affinché ogni processo sia a conoscenza degli altri nodi nel sistema distribuito è stato realizzato un meccanismo di server discovery.
- Quando un nuovo processo desidera entrare a far parte del sistema, questo contatterà il Server registry ad un indirizzo noto, invocando il servizio **JoinNetwork**, in modo da ottenere:
  - Il proprio Id;
  - La lista dei processi nel sistema e il relativo ID.
- Il Server registry si occupa anche di aggiornare tutti i processi in rete in caso di una nuova entry

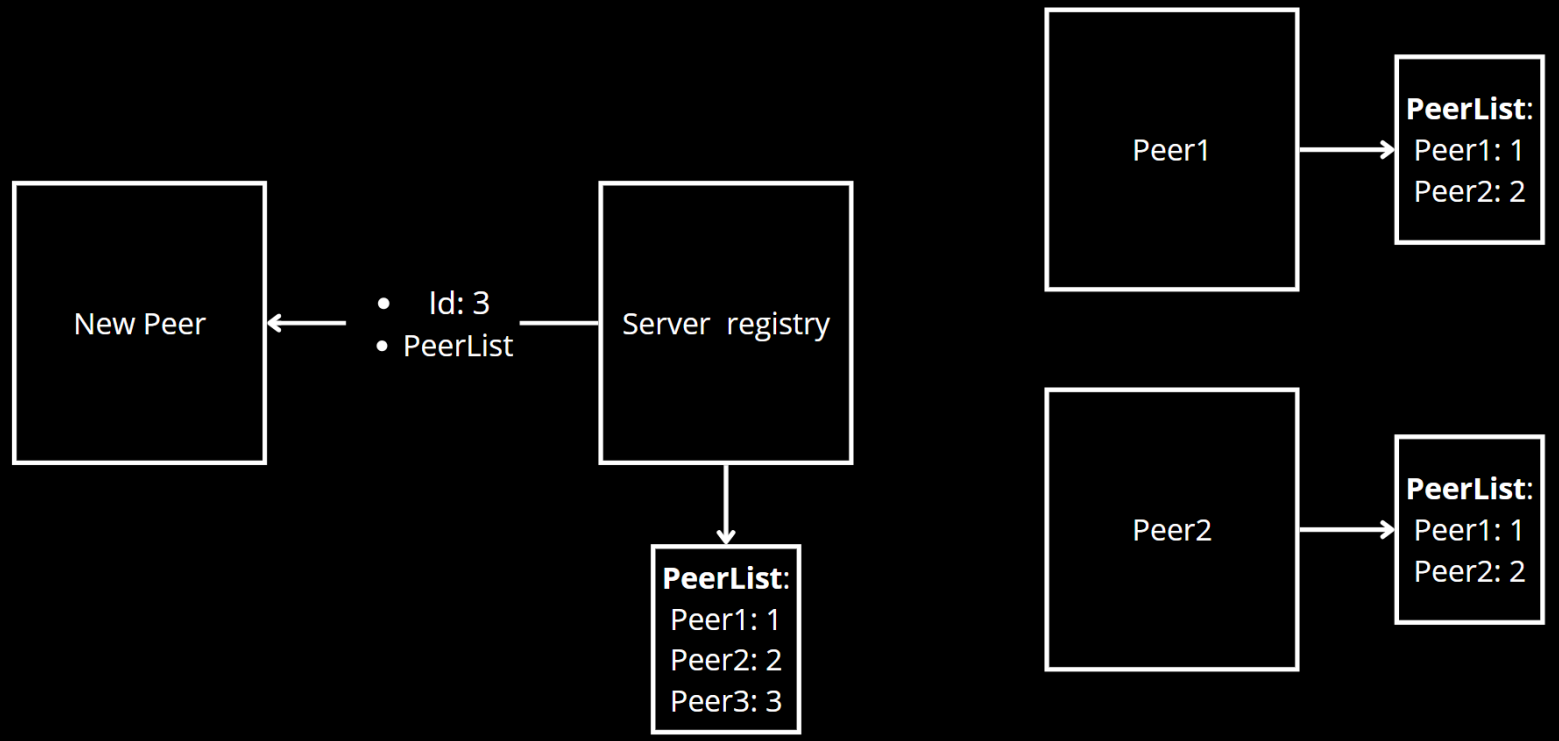
Server registry: Funzionamento

- Un nuovo nodo entra nella rete.



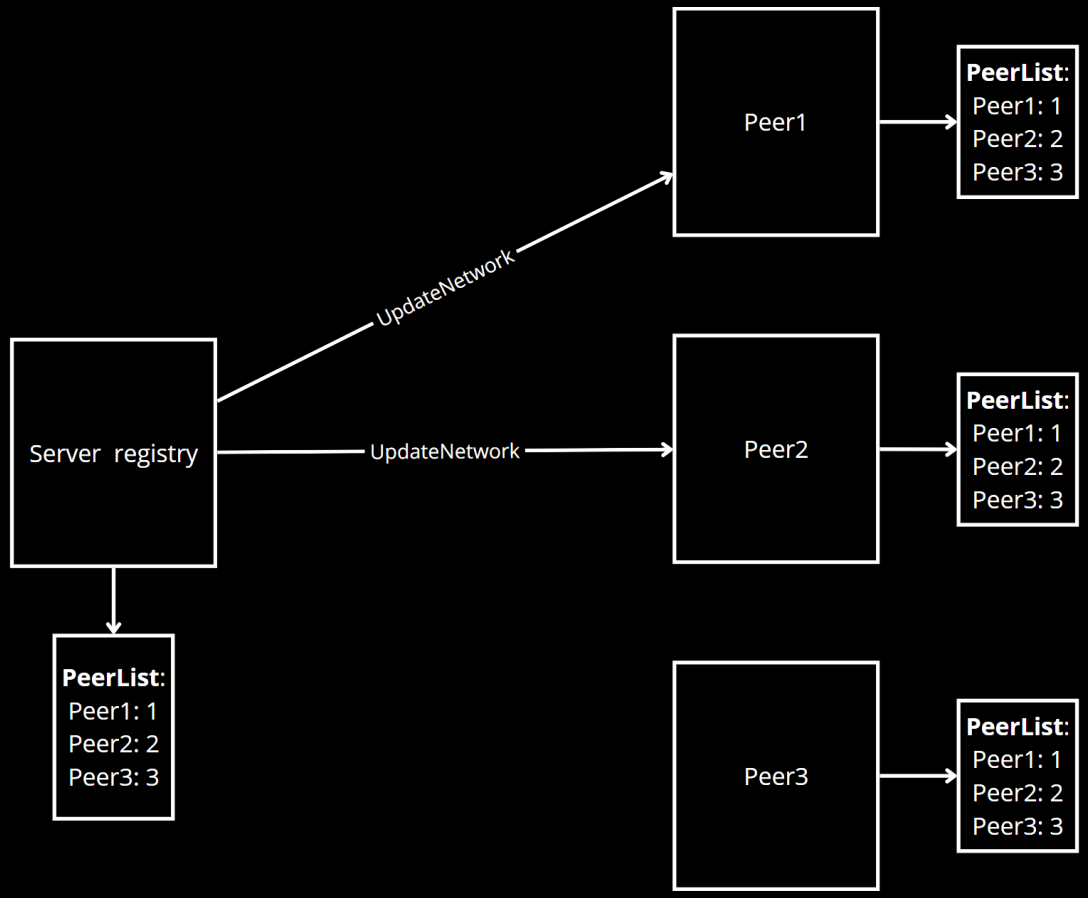
Server registry: Funzionamento

- Il server registry aggiorna la propria PeerList;
- Il server registry risponde inviando un Id e la PeerList.



Server registry: Funzionamento

- Il server registry invia l'aggiornamento a tutti i nodi (tranne all'ultimo entrato)

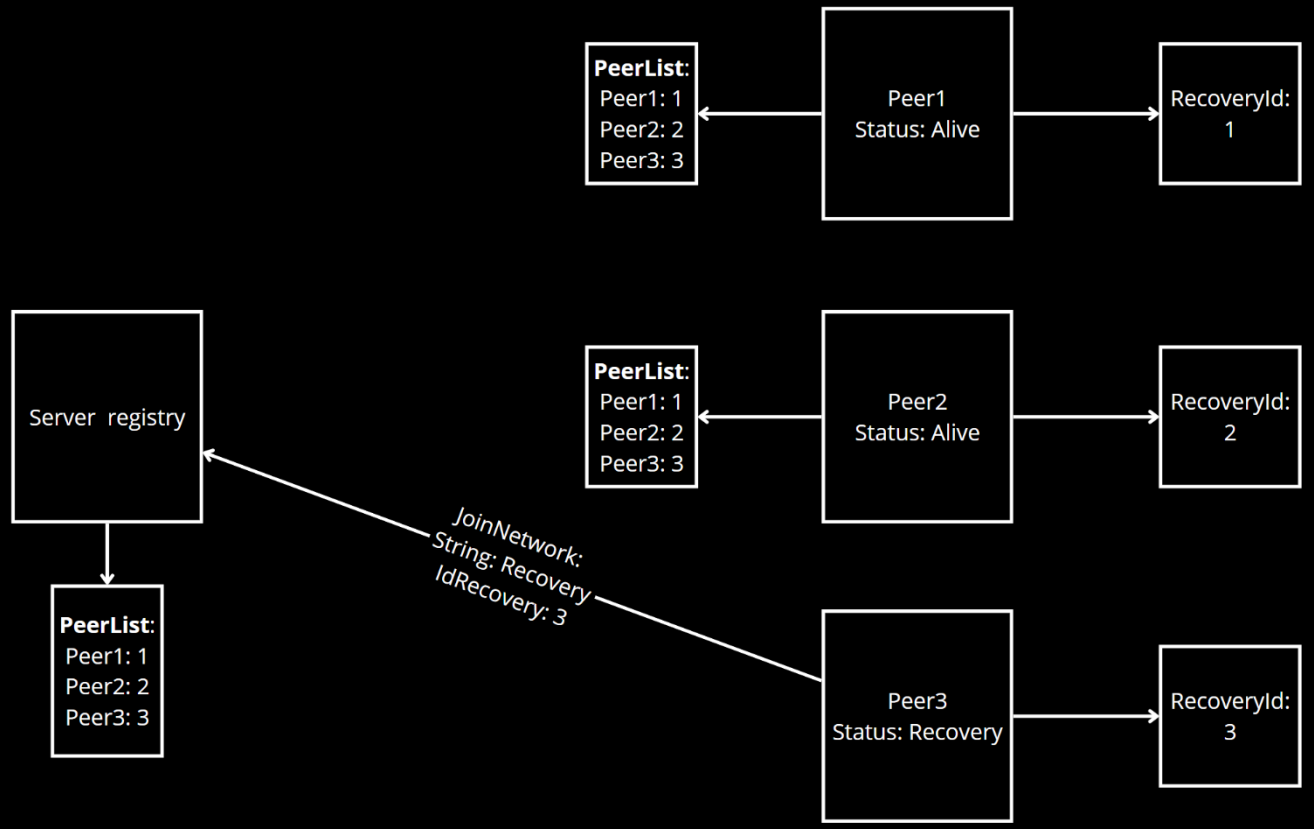


Server registry e crash-recovery

Come già annunciato, potrebbe accadere che a un certo istante un nodo fallisca, il server registry deve gestire l'eventualità che un peer sia in fase di recupero.
Supponiamo che un peer sia in fase di crash recovery:

Server registry e crash-recovery

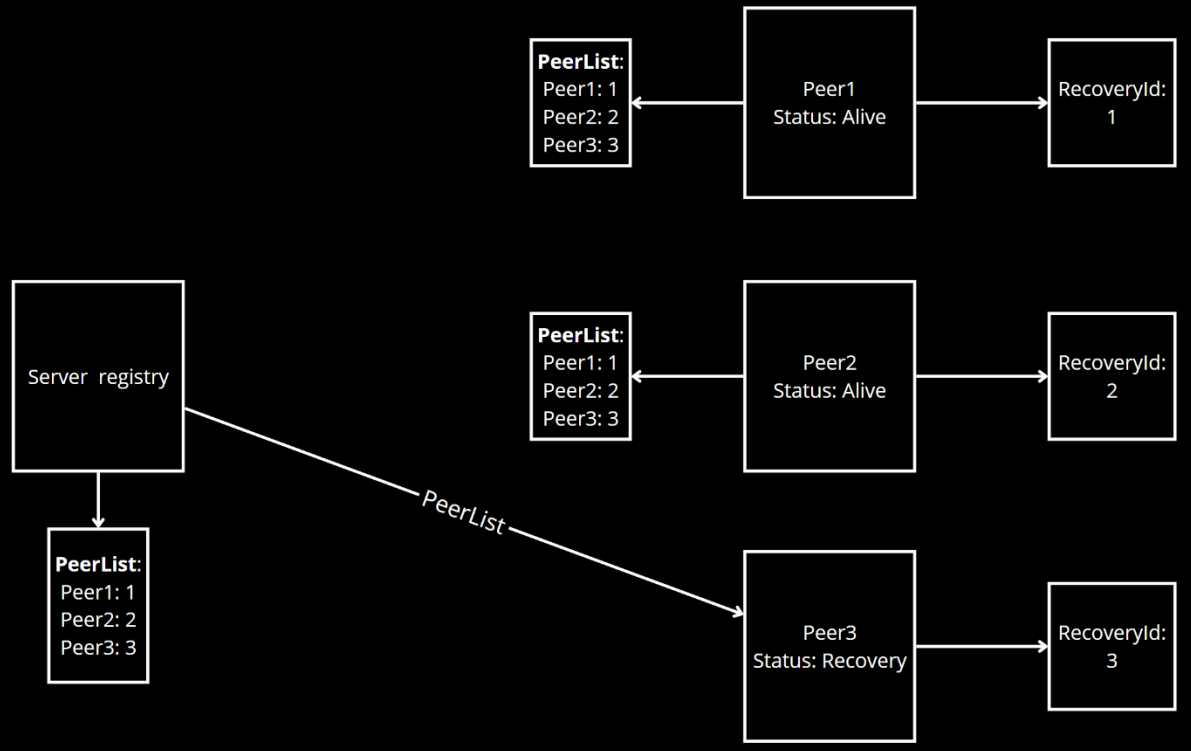
- Il peer si rende conto che è in fase di recupero dalla presenza del file Id.yaml
- Il peer invia una richiesta di JoinNetwork con i parametri:
  - String: «Recovery»;
  - IdRecovery.



Server registry e crash-recovery

Il server registry:

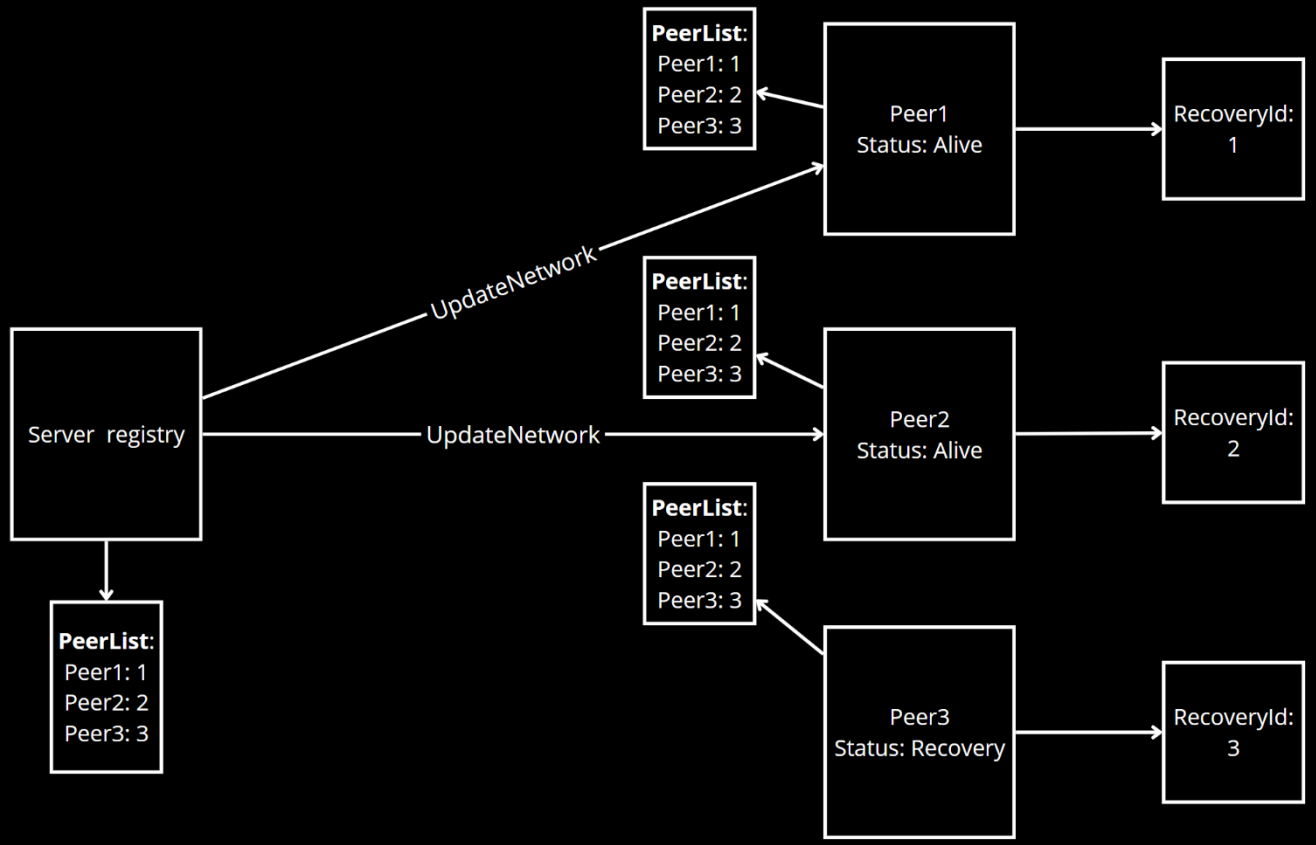
- Identificazione del processo in stato di crash recovery;
- Aggiorna la propria PeerList;
- Invia la PeerList al client.



Server registry e crash-recovery

Il server registry:

- Se le informazioni relative al nodo in ripristino sono cambiate, il server invoca il servizio di Update Network.



Server registry: duplicazione e guasti

Server registry: duplicazione e guasti

Come abbiamo analizzato fino ad ora, il server registry svolge un ruolo cruciale nella rete, senza il quale un nuovo processo non potrebbe ottenere informazioni per comunicare con gli altri nodi.

Avere un unico server registry rappresenterebbe quindi un unico punto di fallimento del sistema. Sarebbe quindi opportuno gestire i possibili guasti del server e replicare quest'ultimi per distribuire il carico di lavoro.

Nel progetto tali aspetti non sono stati trattati, ma cercheremo comunque di fornire una possibile soluzione a tale problema

Server registry: duplicazione e guasti

Soluzione proposta:

Una possibile soluzione sarebbe quella di replicare i vari server registry e utilizzare un load balancer per distribuire le richieste tra di essi.

Per non sovraccaricare il load balancer, quest'ultimo non dovrebbe gestire le risposte, ma semplicemente instradare le richieste ai server registry disponibili.

Problema: A causa della replicazione, la lista dei processi mantenuta da ciascun processo potrebbe non essere consistente.

Server registry: duplicazione e guasti

Soluzione proposta:

Per risolvere questa ulteriore problematica, si propone l'utilizzo di Apache Kafka. Ogni registry sarebbe registrato su un topic in cui vengono pubblicati messaggi contenenti la lista aggiornata dei peer.

Quando un server deve registrare un nuovo nodo in ingresso:

- **Verifica se ci sono messaggi sul topic:** In caso affermativo, consuma i messaggi e aggiorna la propria lista dei processi
- **Risponde al nodo:** Invia la lista aggiornata al nuovo peer in entrata
- **Pubblica un nuovo messaggio:** Fornisce informazioni sul nuovo peer entrato

Server registry: duplicazione e guasti

Soluzione proposta:

Infine, sarebbe opportuno gestire i fallimenti dei server registry e in particolare la fase di crash-recovery. Per gestire tale situazione, si potrebbe definire un leader fra tutti i sever registry che ha il compito di fornire la lista aggiornata al server in fase di rcupero

Algoritmi di elezione distribuita

Algoritmo di Bully:

- L'idea principale dell'algoritmo di Bully è che il nodo con l'ID più alto si fa strada prepotentemente verso la leadership.

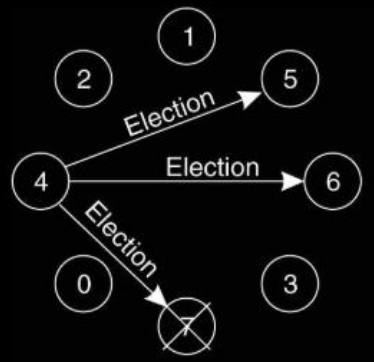
- Supponiamo di avere un sistema con N processi e che il processo p_i si accorga che il coordinatore non sia più attivo:

- p_i avvia l'elezione inviando un messaggio di **Election** a tutti i processi con ID più alto del suo.

- Se nessuno risponde, p_i vince l'elezione e invia a tutti i nodi un messaggio **Coordinator.**

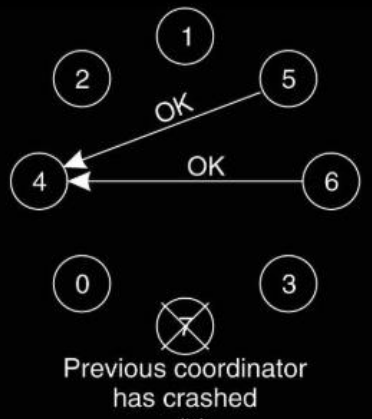
- Se un processo p_k con ID più alto riceve il messaggio **Election** da p_i, gli risponde con un **OK message** che blocca l'elezione precedente**.** Il processo p_k avvierà quindi una nuova elezione ripetendo gli stessi passaggi sopracitati.

Algoritmo di Bully:



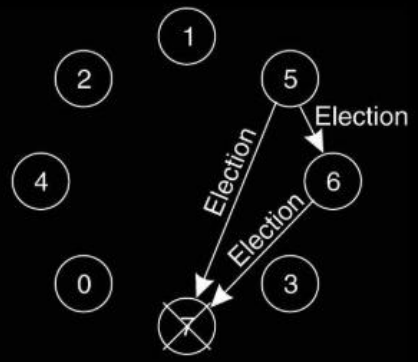
(a)

p_4 initiates the election



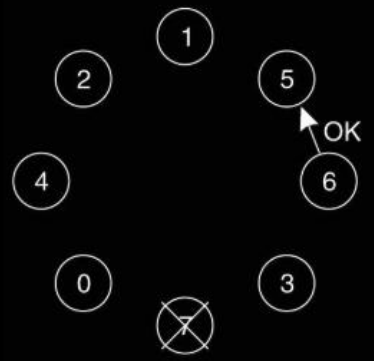
(b)

p_5 and p_6 stop p_4



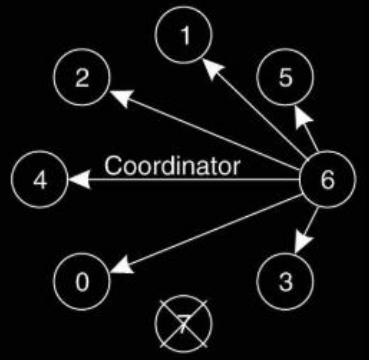
(c)

p_5 and p_6 initiate a new election



(d)

p_6 stops p_5



(e)

p_6 wins the election and becomes the new coordinator

Algoritmo di bully: Costo computazionale

Al fine di valutare il numero di valutare l'efficienza di un algoritmo, possiamo valutare il numero di messaggi scambiati, consideriamo due casi:

- **Best case:** Il processo con il secondo ID più alto si accorge che il leader è fallito
  Costo computazionale: $O(N)$
- **Worst case:** Il processo con l'ID più piccolo avvia l'elezione.
  Costo computazionale: $O(N^2)$

Algoritmo DKR:

L’algoritmo DKR si basa sull’idea di confrontare gli identificatori dei nodi con quelli dei loro vicini per individuare un massimo locale. Successivamente, i nodi condivideranno i loro massimi locali con l’obiettivo di determinare un massimo globale per l’intera rete.

Per il funzionamento dell’algoritmo è necessario introdurre un set di variabili che verranno utilizzate:

Algoritmo DKR: variabili utilizzate:

- **StimateLeader**: Rappresenta la conoscenza che un nodo ha della rete. Ad ogni istante di tempo rappresenta il massimo locale conosciuto dal processo. All’inizio ogni processo ha la variabile StimateLeader pari al proprio ID. Con lo scambio dei messaggi tra i vari processi, la variabile sarà aggiornata al valore max(StimateLeader,ReceivedId).

Algoritmo DKR: variabili utilizzate

- **State:** Indica lo stato del nodo, può essere:
  - **Active:** All’inizio di ogni elezione, ogni processo è in uno stato attivo. Indica che il processo non ha ancora informazioni sui propri vicini e rappresenta un possibile candidato per diventare leader.
  - **Waiting:** Indica che il processo ha ricevuto messaggi dai suoi vicini e, per il momento, il suo ID rappresenta un massimo locale. Il nodo rappresenta un possibile candidato per diventare leader
  - **Passive:** Indica che il processo ha ricevuto messaggi dai suoi vicini e il suo ID è minore del massimo locale. In questo stato, il nodo non rappresenta un candidato per diventare leader.

Algoritmo DKR: Funzionamento

- p_i invia la sua variabile StimateLeader al nodo che lo sussegue in senso orario. Se il nodo non risponde invio al successore del mio successore.

Algoritmo DKR: Funzionamento

- Un nodo p_j che riceve un messaggio dal proprio vicino, verifica il proprio stato e si comporta come segue:
  - **Il processo p_j è nello stato active**: Confronta il valore ricevuto (ReceivedStimateLeader) con la sua variabile stimateLeader:
    - **Se ReceivedStimateLeader > stimateLeader:** Pongo stimateLeader=ReceivedStimateLeader ed entro nello stato passive;
    - **Altrimenti,** entro nello stato waiting.

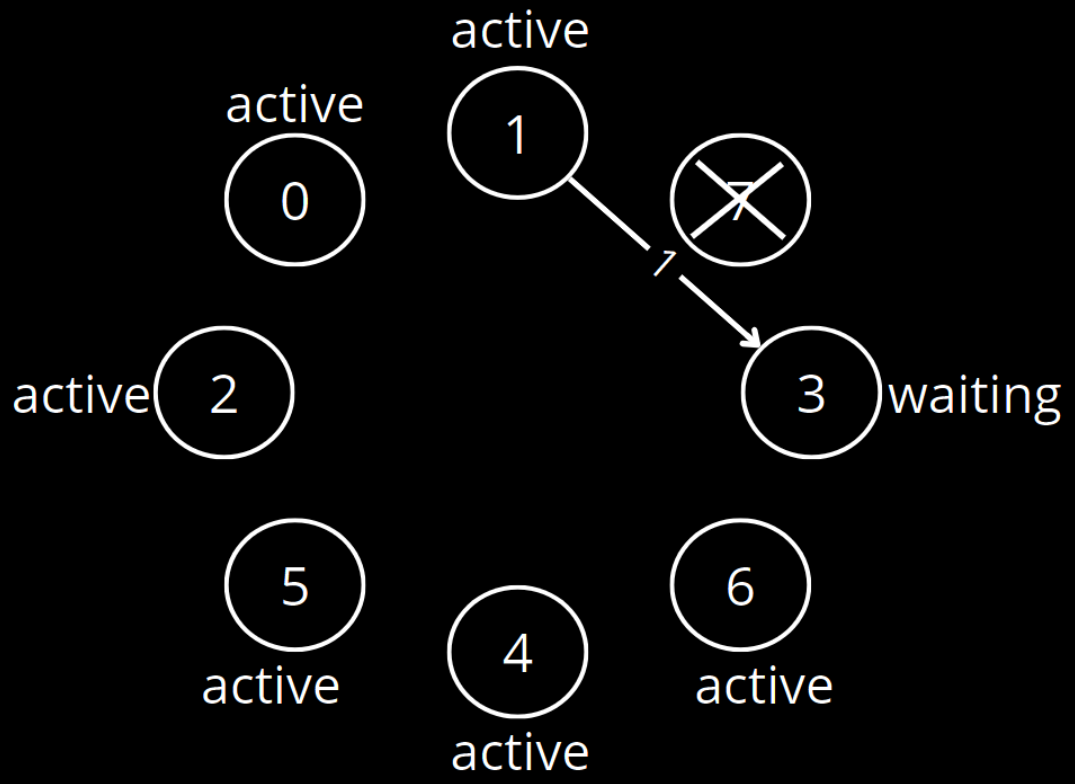
Algoritmo DKR: Funzionamento

- Un nodo p_j che riceve un messaggio dal proprio vicino, verifica il proprio stato e si comporta come segue:
  - **Il processo p_j è nello stato passive:** Confronto il valore ricevuto (ReceivedStimateLeader) con la sua variabile stimateLeader, aggiornandola se necessario e propagado l'informazione

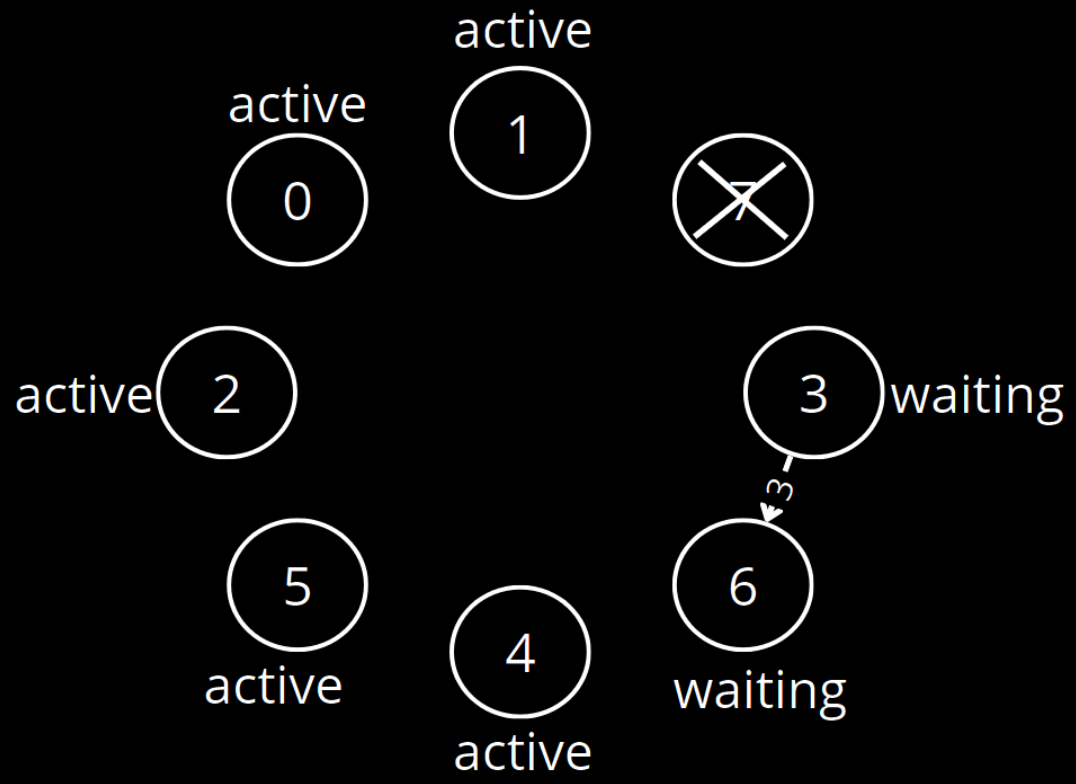
Algoritmo DKR: Funzionamento

- Un nodo p_j che riceve un messaggio dal proprio vicino, verifica il proprio stato e si comporta come segue:
  - **Il processo p_j è nello stato waiting:** Confronta il valore ricevuto (ReceivedStimateLeader) con la sua variabile stimateLeader:
    - **Se ReceivedStimateLeader = ID(p_j):** p_j diventa leader;
    - **Altrimenti,** il processo entra nello stato passive, aggiorna la propria variabile StimateLeader e propaga al vicino l'informazione

Algoritmo DKR: Funzionamento

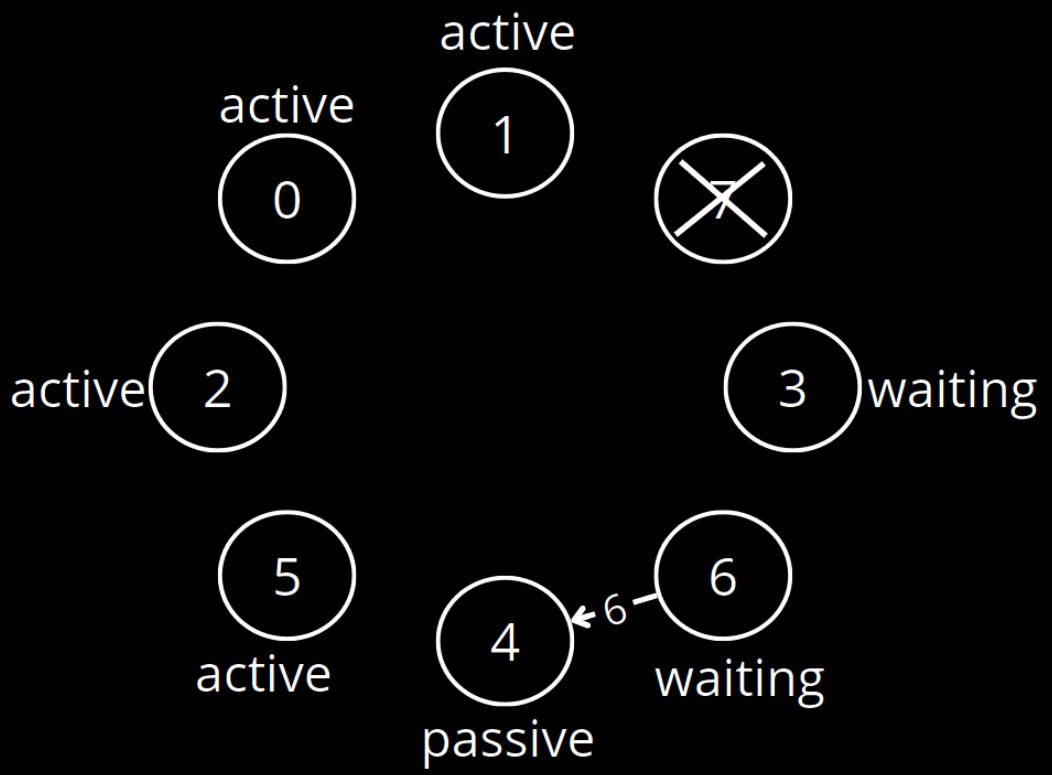


P1 initiates the election

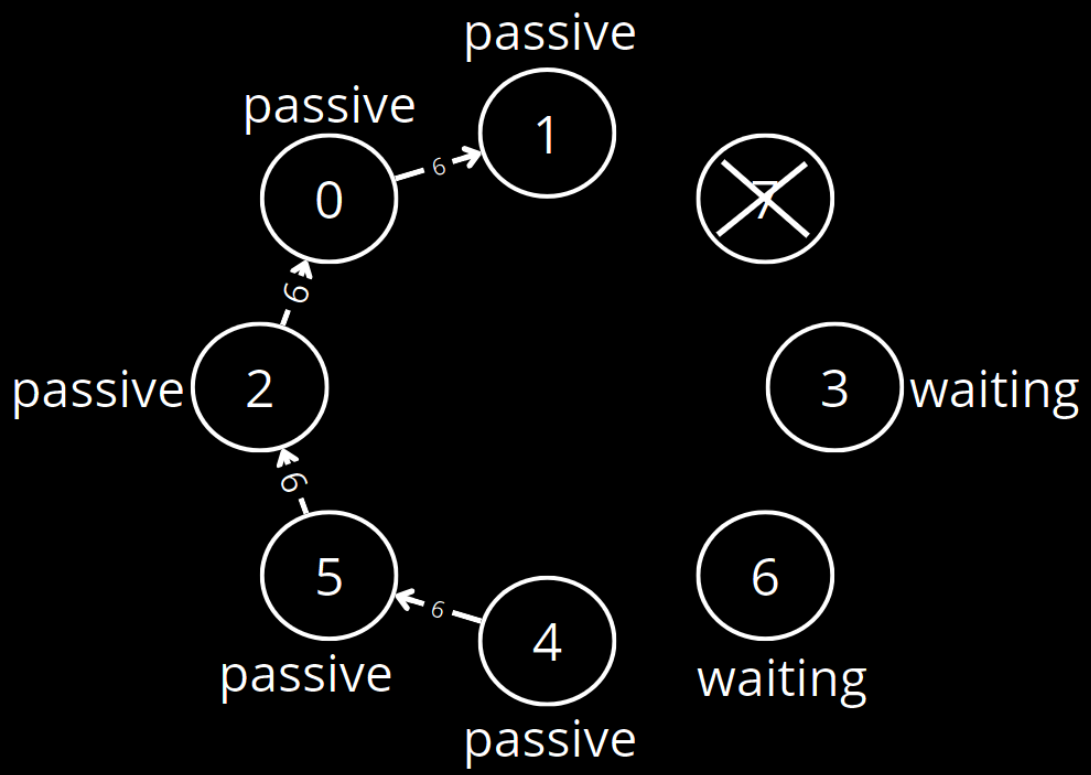


P3 propagates the election

Algoritmo DKR: Funzionamento

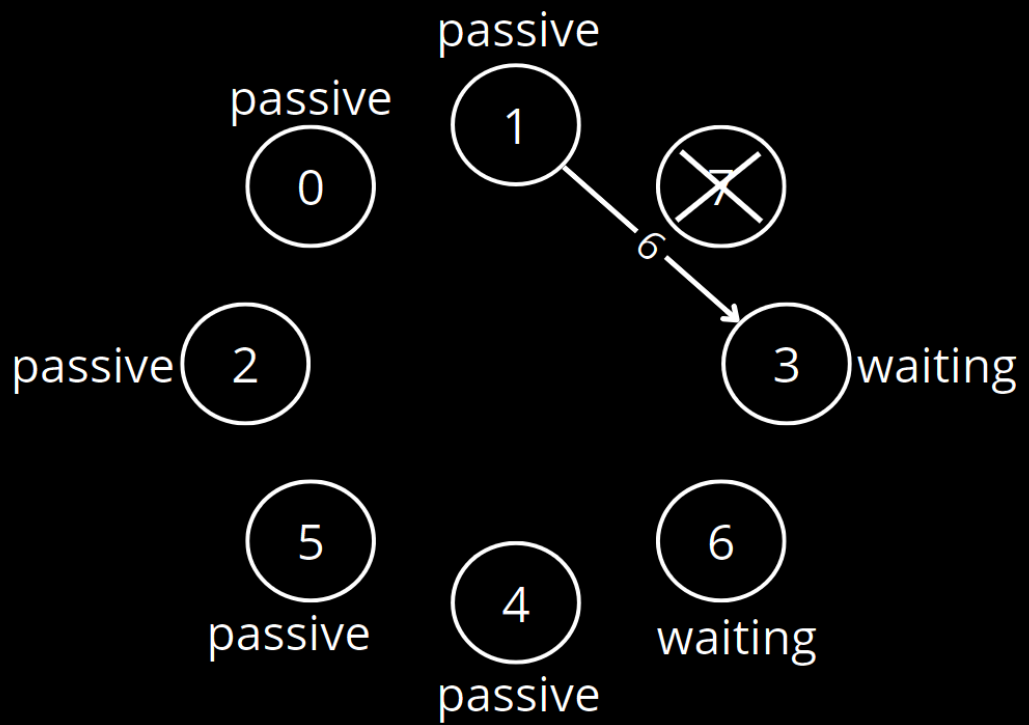


P6 propagates the election

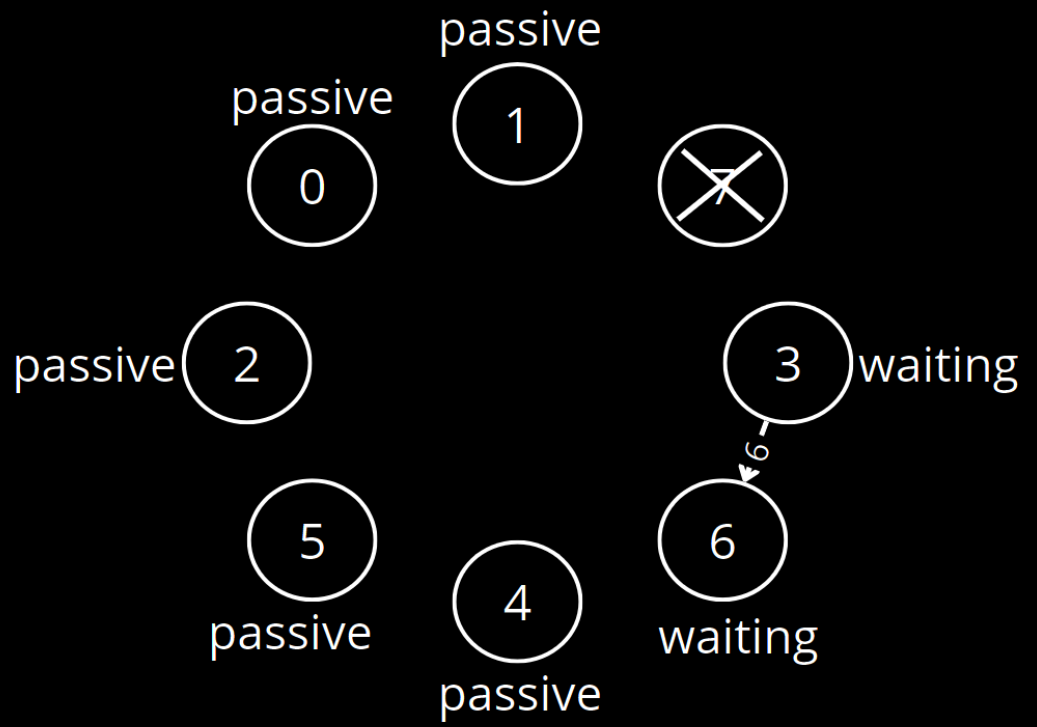


P6 is the process with the highest ID in the network, for this reason, P4, P5, P0, and P1 become passive and forward the value 6 to their neighbours

Algoritmo DKR: Funzionamento



P1 sends the value 6 to P3, which is waiting.
Upon receiving a value different from its ID,
P3 transitions into the passive state



P3 sends the value 6 to P6, which is waiting.
Upon receiving a value equal to its ID, P6
becomes the new leader.

Tecnologie Utilizzate

Linguaggio di programmazione: gRPC in go

gRPC è un framework di comunicazione remota sviluppato da Google. Le principali caratteristiche che presenta sono:

- **Supporto per applicazioni poliglotte:** Possibilità di scrivere i diversi programmi in differenti linguaggi di programmazione, mantenendo però la capacità di comunicazione fra essi
- **Comunicazione affidabile:** gRPC utilizza HTTP/2 per la comunicazione, offrendo così la possibilità di avere streaming bidirezionale.
- **Utilizzo di Protocol buffer:** gRPC usa Protocol Buffer sia come IDL che permette la genereazione del client/server stub, sia come formato di scambio messaggi.

Docker

Docker è una tecnologia di virtualizzazione basata su container. Un container Docker contiene un’applicazione e le sue dipendenze, isolandole dal resto del sistema e condividendo solo il kernel del sistema operativo con altri container.

L’utilizzo di docker ci permette di isolare le nostre applicazioni e le relative dipendenze garantendo che esse siano eseguite in ambienti containerizzati rendendo più agevole il processo di deployment su diversi ambienti.

Docker

Nel progetto sono stati implementati due DockerFile:

- Uno per il server registry
- Uno per i processi nel sistema

Per orchestrare e avviare l’insieme dei container, abbiamo utilizzato Docker compose. Abbiamo creato un file «compose.yaml» dove sono stati specificati i servizi necessari e le configurazioni per comporre e avviare i container