

Cluster Store e Tolerância a Falhas

Felipe Braz Marques - 22.1.4030

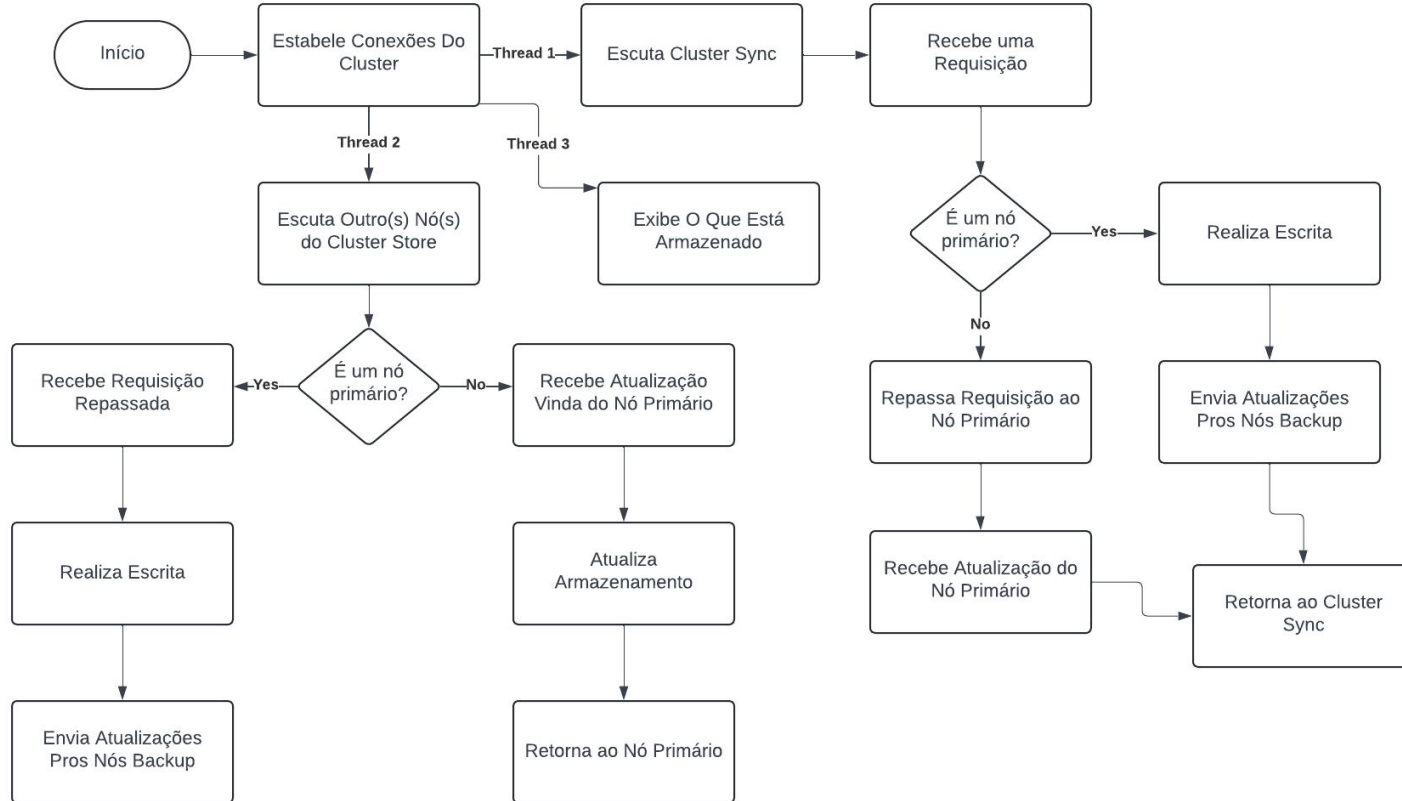
Matheus Peixoto Ribeiro Vieira - 22.1.4104

Pedro Henrique Rabelo Leão de Oliveira - 22.1.4022

Cluster Store

```
1 class noClusterStore:
2     def __init__(self, id, host, portaRequisicao, porta1 = None, porta2 = None, porta_no_primario = None, host_no_primario = None):
3         self.id = id
4         self.host = host
5         self.portaRequisicao = portaRequisicao # porta para receber requisicoes do cluster sync
6         self.primario = True if id == 0 else False # representa se o no eh um no primario do cluster store ou um no de backup
7         self.armazenamento = ""
8
9         if self.primario:
10             self.porta1 = porta1 # porta com qual um dos nos de backup do cluster store fara conexao
11             self.porta2 = porta2 # porta com qual um dos nos de backup do cluster store fara conexao
12             self.conn_backup1 = None
13             self.conn_backup2 = None
14
15             self.no_backup1_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
16             self.no_backup2_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
17
18         else:
19             self.porta_no_primario = porta_no_primario # porta para estabelecer conexao com o no primario
20             self.host_no_primario = host_no_primario # host do no primario para estabelecer conexao com ele
21
22             self.no_primario_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
23
24         self.no_clusterSync_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
25
26         self.mutex = threading.Lock()
27         self.esperar_resposta = threading.Condition()
```

Cluster Store



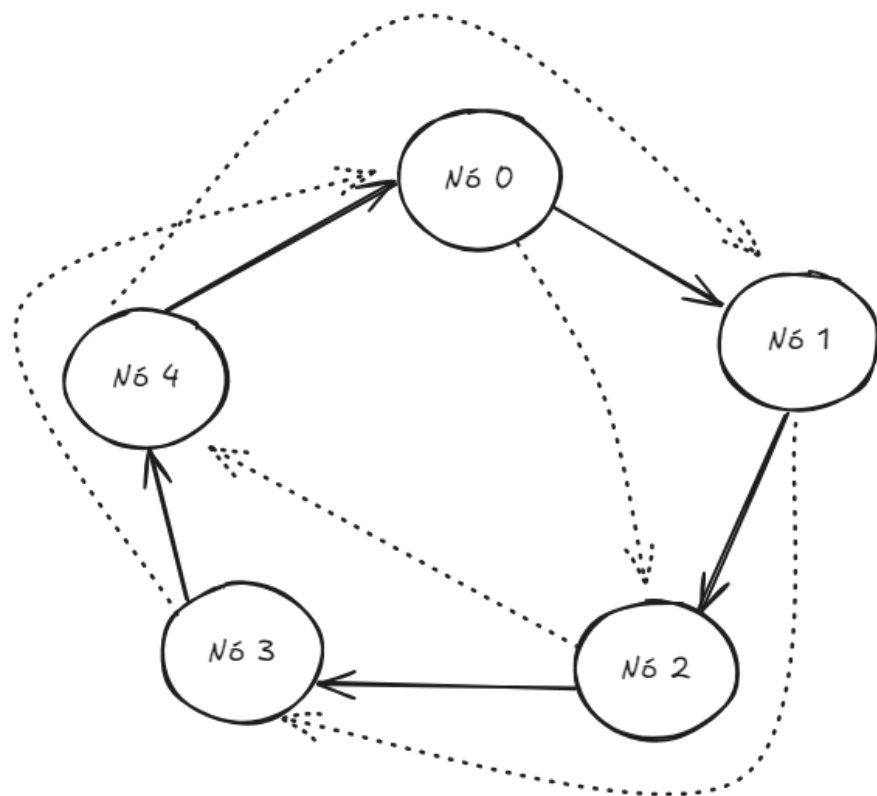
Conexão do Cluster Sync com o Cluster Store



```
1 class ClusterStore:
2     def __init__(self, lista_endreco):
3         self.cluster_store = [ClusterSync_ClusterStore(host, porta) for (host, porta) in lista_endreco]
4
5     def enviar_mensagem(self, mensagem):
6         selecionado = random.randint(0, len(self.cluster_store) - 1)
7         cluster = self.cluster_store[selecionado]
8
9         cluster.iniciar_conexao()
10        cluster.enviar_mensagem(mensagem)
11        cluster.esperar_retorno()
12        cluster.finalizar_conexao()
```

Conexão do Cluster Sync com o Cluster Store

```
1  # faz a comunicacao de um no do clusterSync com um no do clusterStore
2  class ClusterSync_ClusterStore:
3      def __init__(self, host, porta):
4          self.host = host # host do no do cluster store
5          self.porta = porta # porta do no do cluster store
6          self.socket_cSync_cStore = None
7
8      def iniciar_conexao(self):
9          # Verifica se o socket é None, recria-o se necessário
10         if self.socket_cSync_cStore is None:
11             self.socket_cSync_cStore = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
12             self.socket_cSync_cStore.connect((self.host, self.porta))
13
14         def enviar_mensagem(self, mensagem):
15             self.socket_cSync_cStore.sendall(json.dumps(mensagem).encode())
16
17         def esperar_retorno(self):
18             dados = self.socket_cSync_cStore.recv(BUFFER_SIZE)
19             mensagem = json.loads(dados.decode())
20             #print(mensagem["status"])
21
22         def finalizar_conexao(self) :
23             self.socket_cSync_cStore.close()
24             self.socket_cSync_cStore = None
```



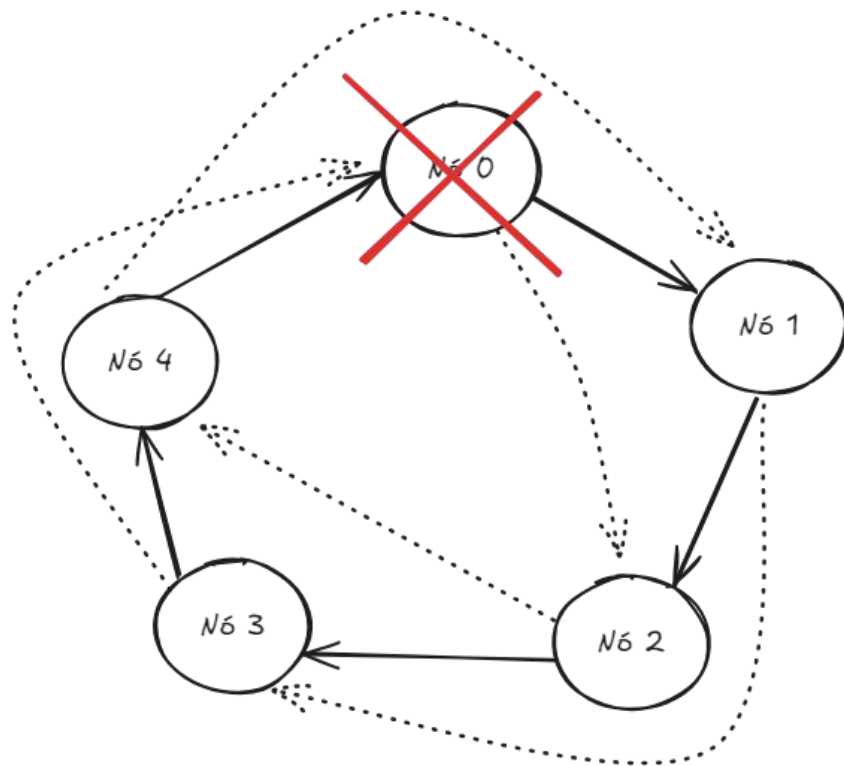
N6 0: 1 e 2

N6 1: 2 e 3

N6 3: 4 e 5

N6 4: 5 e 0

N6 5: 0 e 1



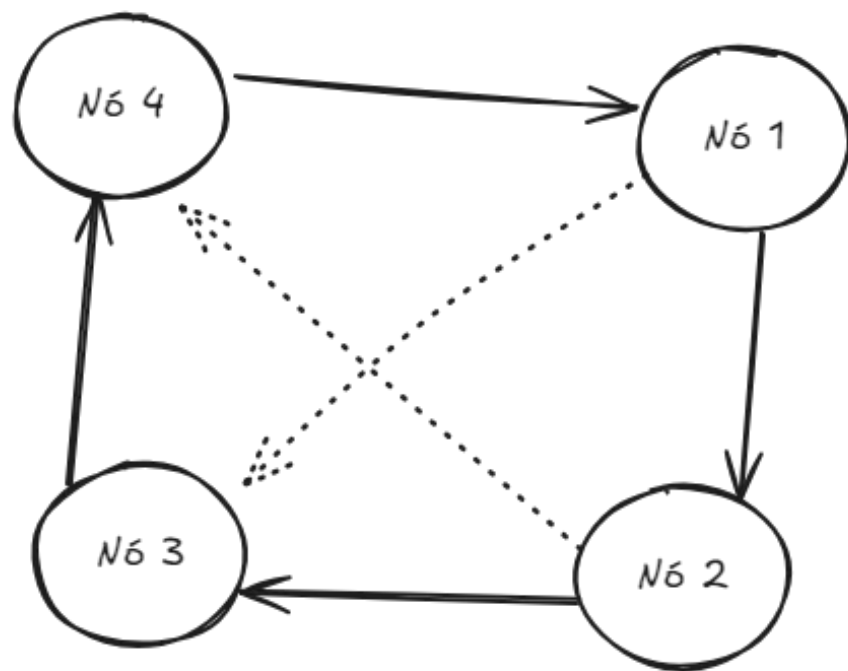
N6 0: 1 e 2

N6 1: 2 e 3

N6 3: 4 e 5

N6 4: 5 e 0

N6 5: 0 e 1



1.1 Cluster Sync sem pedido do cliente

```
1 def escutar_cliente(self):
2     # Fazendo o binding
3     self.cliente_socket.bind((self.host, self.porta_cliente))
4     self.cliente_socket.listen()
5
6     conn, addr = self.cliente_socket.accept()
7
8     with conn:
9         with self.mutex:
10             self.clienteConectado = True
11             print(f"Nó {self.id_no} conectado com o cliente {addr}")
12
13         while True:
14             dados = conn.recv(BUFFER_SIZE)
15             if not dados:
16                 break
17             mensagem = json.loads(dados.decode())
18             timestamp = mensagem.get('timestamp')
19             self.timestamp_cliente = timestamp
20
21             with self.esperar_resposta:
22                 self.esperar_resposta.wait() # Bloqueando a thread, esperando o nó sair da região crítica
23
24             conn.sendall(json.dumps({"status": "committed"}).encode())
25
26             if self.queda == 1:
27                 self.cair()
```

1.2 Cluster Sync com pedido do cliente

```
1 def executar_no(self):
2     self.rodadaPassandoToken = 1
3
4     self.bind_para_no_anterior()
5     time.sleep(1)
6     threading.Thread(target=self.aceitar_conexoes).start()
7
8     self.conectar_ao_no_proximo()
9     time.sleep(2)
10
11     # O primeiro no inicia o processo de escrever no vetor (token) vazio do anel e passar ao proximo no
12     if self.id_no == 0:
13         self.escrever_no_token() # Escreve no vetor
14         self.enviar_para_proximo()
15
16     while True:
17         self.esperar_token() # Espera o vetor (token) do no anterior
18
19         if self.verificar_regiao_critica():
20             self.entrar_regiao_critica()
21             self.sair_da_regiao_critica()
22
23         else:
24             self.escrever_no_token() # Escreve no vetor
25             if self.queda == 2:
26                 self.cair()
27
28             self.enviar_para_proximo() # Envia o vetor para o proximo no
29
30     self.rodadaPassandoToken += 1
```

1.1, 1.2 Cluster Sync sem/com pedido do cliente - Solução


Ao tentar enviar para o próximo, caso tenha um erro, então ele estará indisponível. Então, coloca sua posição como None.

Não se preocupa com o N+2 pois, eventualmente, ele será verificado.

Menor inundação da rede do que se fizesse um ping.

```
1 def enviar_para_proximo(self):
2     if self.no_proximo_conectado:
3         try:
4             self.no_proximo_socket.sendall(json.dumps(self.token).encode())
5         except:
6             self.no_proximo_conectado = False
7             self.token[(self.id_no + 1) % self.num_de_nos] = None
8
9     if self.no_proximo_proximo_conectado:
10        try:
11            self.no_proximo_proximo_socket.sendall(json.dumps(self.token).encode())
12        except:
13            self.no_proximo_proximo_conectado = False
14            self.token[(self.id_no + 2) % self.num_de_nos] = None
```

1.3 Cluster Sync na zona crítica



```
1 def entrar_regiao_critica(self):
2     print(f"Nó {self.id_no} entrando na seção crítica...")
3
4     mensagem = f"Cluster Sync id: {self.id_no} - Timestamp do cliente: {self.timestamp_cliente}"
5     self.cluster_store.enviar_mensagem(mensagem)
6
7     if self.queda == 3:
8         self.cair()
```

1.3 Cluster Sync na zona crítica - Solução

- Cada nó recebe o token de N-1 e N-2 de forma paralela por soquetes diferentes;
- Se recebe somente de N-1, utiliza ele e irá parar de esperar por N-2;
- Se recebe somente de N-2, utiliza ele e irá parar de esperar por N-1;
- Se recebeu de ambos, utiliza o token de N-1, pois é o mais atual

```
1 def esperar_token_anterior(self, espera: EsperaToken):
2     dados = ''
3     while espera.anterior_conectado:
4         try:
5             dados = espera.conexao.recv(BUFFER_SIZE)
6             if dados:
7                 espera.token = json.loads(dados.decode())
8                 espera.recebeu_resposta = True
9                 return
10        except:
11            espera.recebeu_resposta = False
12            espera.anterior_conectado = False
13            return
14
15 def esperar_token(self):
16     # Informações a serem passadas por referencia
17     anterior = EsperaToken(self.conn_anterior, self.no_anterior_conectado)
18     anterior_anterior = EsperaToken(self.conn_anterior_anterior, self.no_anterior_anterior_conectado)
19
20     t_anterior = threading.Thread(target=self.esperar_token_anterior, args=(anterior,))
21     t_anterior_anterior = threading.Thread(target=self.esperar_token_anterior, args=(anterior_anterior,))
22
23     t_anterior.start()
24     t_anterior_anterior.start()
25
26     t_anterior.join(5)
27     t_anterior_anterior.join(3)
28
29     if self.rodadaPassandoToken != 1 and (self.id_no == 1 or self.id_no == 0):
30         # Atualiza as flags de conexão
31         self.no_anterior_conectado = anterior.anterior_conectado
32         self.no_anterior_anterior_conectado = anterior_anterior.anterior_conectado
33
34     # Verifica se recebeu o token do nó anterior
35     if anterior.recebeu_resposta:
36         self.token = anterior.token
37
38     elif anterior_anterior.recebeu_resposta:
39         self.token = anterior_anterior.token
40
41     # Remove o timestamp do token anterior, pois era o menor e bloqueia o acesso a região crítica
42     self.token[(self.id_no - 1) % self.num_de_nos] = None
```