

Mestrado em Engenharia Informática e Computadores

**Computação Distribuída**

# Projecto Final

JANUARY 15, 2020

GRUPO 12  
ALEXANDRE SANTOS - 47926  
ANA PEREIRA – 47930  
RÚBEN SANTOS - 47766

# ***1 Índice***

2	Introdução .....	3
3	Enquadramento .....	4
4	Arquitetura .....	5
5	Mecanismos.....	5
5.1	Cardinalidade de Servidores.....	5
5.2	Cliente e serviço de configuração.....	5
5.3	Mecanismo de eleição .....	7
5.4	Mensagens Spread.....	7
5.5	Operação de Escrita.....	8
5.5.1	Cenário normal (happy path).....	9
5.5.2	Cenário timeout .....	9
5.5.3	Cenário conflito de escrita .....	10
5.6	Operação de Leitura .....	11
5.7	Startup .....	12
6	Demonstrações .....	12
6.1	Demonstração escrita.....	12
6.2	Demonstração leitura com replica local .....	14
6.3	Demonstração leitura sem replica local.....	14
6.4	Demonstração de timeout escrita.....	16
6.5	Demonstração da resolução de um conflito de escrita. ....	17
6.6	Demonstração do startup .....	20
7	Conclusões .....	22

## 2 Introdução

O projeto no âmbito da disciplina de computação distribuída tem como propósito desenvolver um sistema distribuído de armazenamento de dados num cluster de servidores com replicação de dados, com um modelo de consistência garantido por consenso entre todos os servidores usando comunicação por grupos.

Como representa a figura abaixo é pretendido que se desenvolva um sistema distribuído de armazenamento de dados (chave e valor) através da comunicação por grupos. A comunicação entre servidores pertencentes ao *Storage Cluster* é feita pelo *middleware* de comunicação por grupos *Spread Toolkit* e a comunicação entre Clientes e Configuration Service, e Clientes e Servidores, é feita por gRPC (Google Remote Procedure Call).

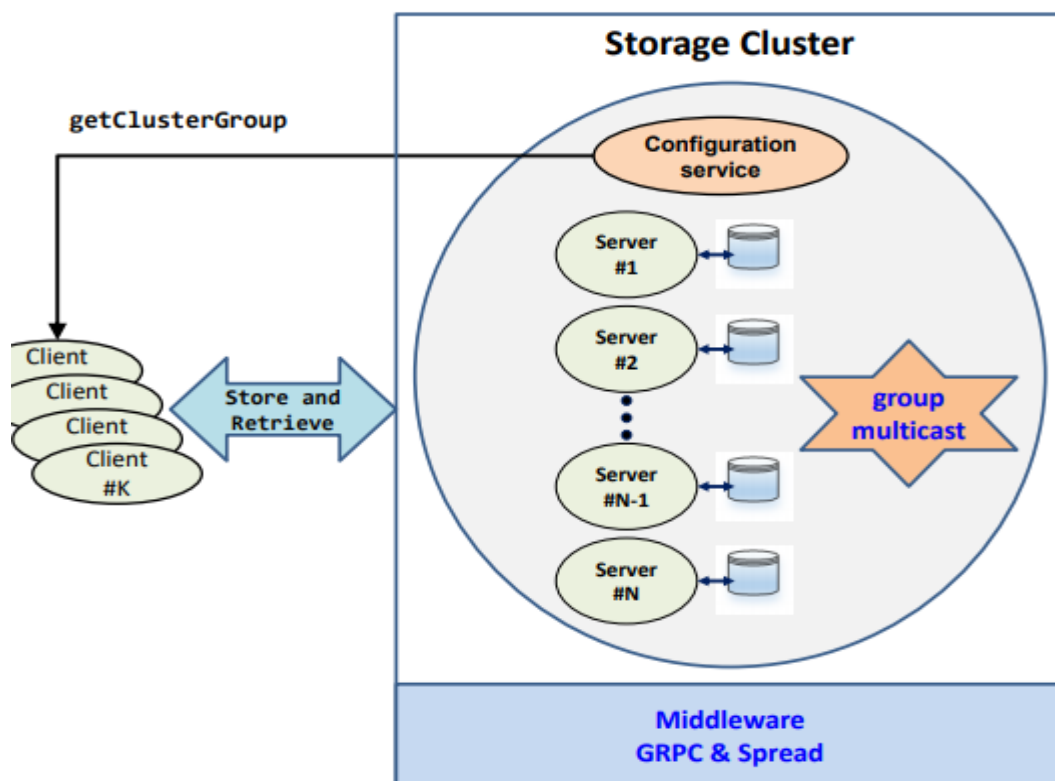


Figura 1 - Diagrama das partes envolvidas

### 3 *Enquadramento*

No funcionamento normal, cada elemento do Storage Cluster corre uma instância do servidor Spread e utiliza esse meio para comunicação com os outros elementos. O *Spread Toolkit* fornece um serviço de mensagens resistente a falhas, tanto em redes locais como em redes extensas. Este serviço funciona como um transportador de mensagens unificado e fornece *multicast*, comunicação em grupo e suporte de comunicação ponto a ponto. Existem dois principais tipos de mensagens que o *Spread* oferece, as mensagens “*regular*”, que consistem nas mensagens criadas pelos próprios membros dos grupos para troca de informação e as mensagens “*membership*”, que são geradas pelo próprio *Spread* que permite manter sempre os elementos do grupo informados dos que saem (quer por desconexão ou falha de ligação) e dos que entram.

Ainda dentro do mesmo cluster e também ele pertencente ao mesmo grupo Spread, vai estar presente um servidor “Configuration Service”. Este tem a função de manter sempre informados os clientes, através de gRPC, dos servidores que se encontram ativos e disponíveis para troca de operações de escrita e leitura.

Por sua vez os clientes, sabendo dos servidores que se encontram ativos e disponíveis, irão escolher um e fazer a comunicação também eles através de gRPC. Caso haja uma falha nessa conexão, o cliente automaticamente entra no processo de escolha de um outro servidor disponível e estabelece essa nova ligação.

O ponto chave deste projeto consiste na implementação de um algoritmo de consenso entre servidores de maneira a que seja evitado conflitos no momento de escrita por parte dos clientes.

## 4 Arquitetura

Aqui é descrita a arquitetura utilizada no projeto, assim como as tecnologias. Ambos componentes podem estar ou não a correr na mesma máquina, no entanto a base de dados *postgres*, normalmente está sempre na mesma máquina que a aplicação *springboot* do servidor.

A interação entre os componentes decorre exatamente como referido no enunciado.

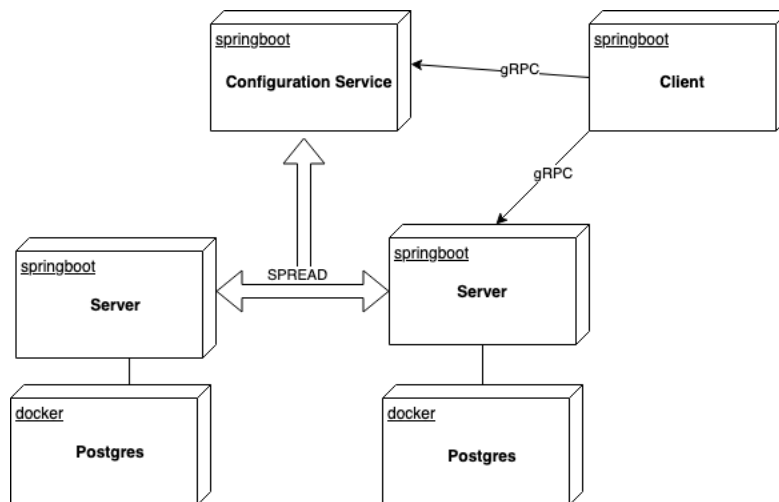


Figura 2 - Arquitetura geral

## 5 Mecanismos

Nesta secção serão explicados os mecanismos utilizados para o desenvolvimento do sistema distribuído, representado ao detalhe a nossa implementação do algoritmo de *consensus* aplicado por cada chave, da par chave valor.

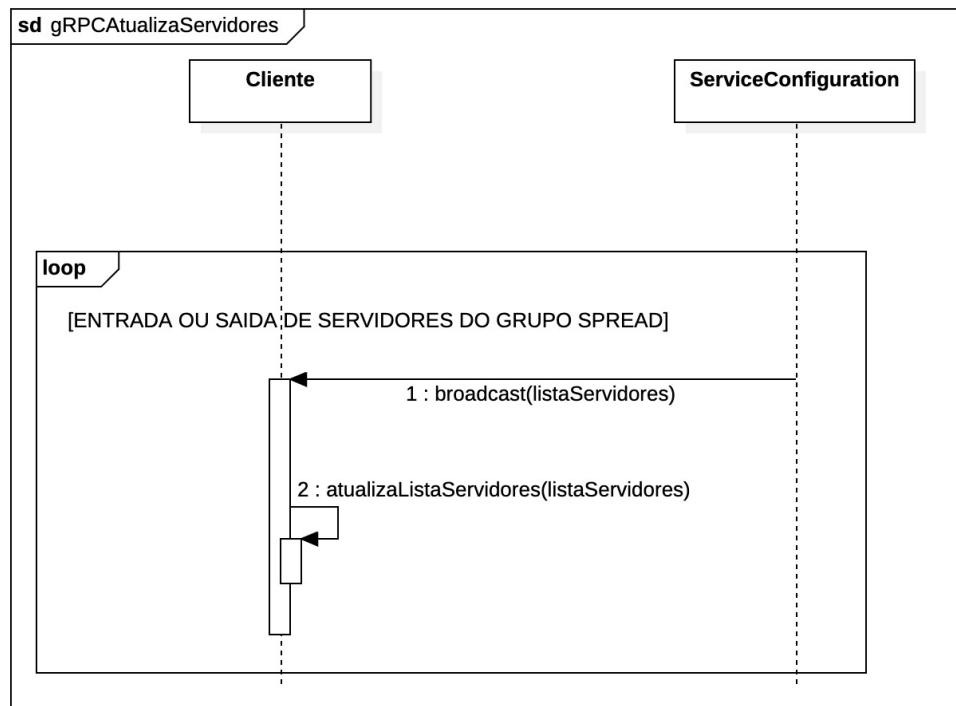
### 5.1 Cardinalidade de Servidores

O Mecanismo de eleição implementado utiliza uma nomenclatura de servidores, esta nomenclatura foi denominada de cardinalidade de um servidor, sendo composta pelo último carácter de um valor único numérico e incremental, do nome do servidor.

Como exemplo, SERVERXX onde XX é o número do servidor, (server00, server01, etc.)

### 5.2 Cliente e serviço de configuração

O serviço de configuração é o ponto comum entre o *cluster* de servidores e o cliente. Este é responsável por informar os clientes com uma lista de servidores disponíveis (servidores conectados ao grupo *Spread*) e também de fazer a tradução do *hostname* para o IP publico enviado para o cliente.



**Figura 3 - Atualização da Lista de Servidores**

### 5.3 *Mecanismo de eleição*

O mecanismo de eleição, consiste na comparação da cardinalidade de um servidor, é realizado por *Spread*, uma vez que através do *Spread* temos o conhecimento de todos os servidores presentes no grupo.

Este processo de eleição pode obter-se por diferentes regras, contudo, no projeto realizado a opção escolhida foi através da análise do servidor com de maior cardinalidade entre todos os servidores existentes.

Assim podemos escolher o servidor com maior cardinalidade para um processo de eleição, aplicando-se tanto num cenário de *startup* para eleger um coordenado, como um cenário de conflito de escrita para eleger um servidor.

A grande vantagem deste mecanismo de eleição é que se aproveita da tecnologia do *Spread*, para permitir uma eleição de servidores sem comunicação de mensagens!

### 5.4 *Mensagens Spread*

De seguida, serão enumerados os tipos de mensagens usados para diferenciação/caracterização das mensagens transmitidas pelo grupo *Spread*, assim como uma breve descrição. Estas mensagens foram criadas e não são fornecidas pela Framework do *Spread*.

1. **AskData:** Mensagem de pedido de valor da chave ao grupo.
2. **AskDataResponse:** Mensagem de resposta com informação pedida por **AskData**;
3. **DataWritten:** Mensagem enviada pelo servidor que realizou a escrita, que informa o servidor recetor que pode proceder com a completa invalidação/ remoção da par chave valor localmente.
4. **RevalidateData:** Mensagem de revalidação da par chave valor;
5. **StartupRequestUpdate:** Mensagem de pedido de auxílio no *startup*;
6. **StartupResponseUpdate:** Mensagem de resposta da **StartupRequestUpdate** com par chave valor para serem atualizados.
7. **WantToWrite:** Mensagem enviado pelo servidor que quer realização operação de escrita, é também a mensagem que invalida a par chave valor no recetor.
8. **WantToWriteResponse:** Mensagem de resposta da **WantToWrite**, o envio desta mensagem representa o sucesso da invalidação da par chave valor do servidor que envia.
9. **WriteConflict:** Mensagem que informa sobre a ocorrência de um conflito durante a operação de escrita de um valor para uma chave.

É ainda de importância referir que, as mensagens foram configuradas com a opção “*setReliable()*”. Esta opção permite ter a garantia de que as mensagens são sempre entregues aos membros do grupo, mesmo em caso de problemas de rede. Isto é possível graças ao mecanismo de recuperação de mensagens da ferramenta *Spread*.

### 5.5 Operação de Escrita

Esta operação consiste no envio, por parte do cliente, de um par de atributos “chave”, “valor” a um servidor com o objetivo do mesmo ser armazenado no cluster e mais tarde poder ser acedido ou atualizado. O diagrama de sequência representativo dos mecanismos deste processo pode ser visualizados na **Error! Reference source not found.**

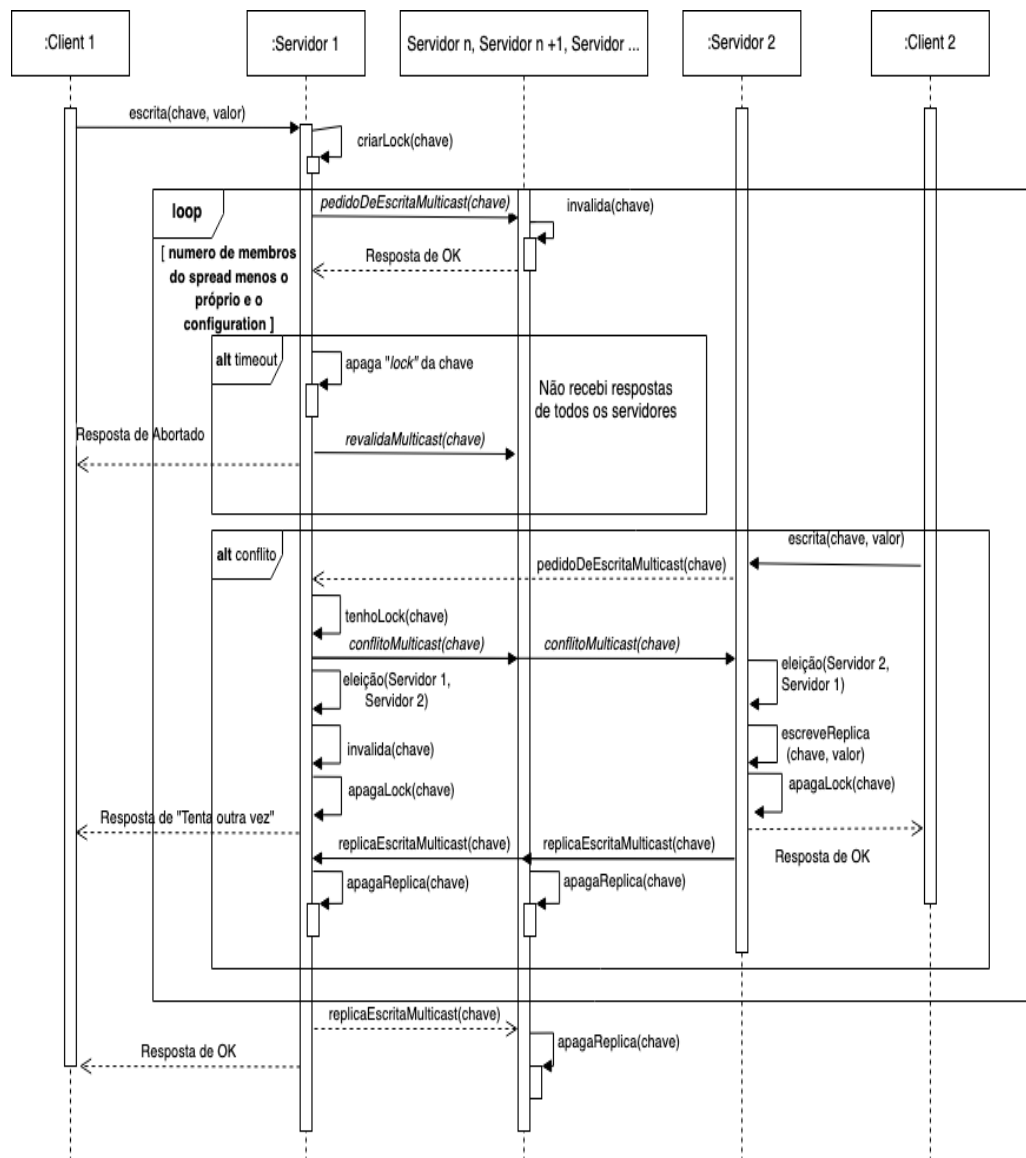


Figura 4 - Diagrama de sequência da operação de escrita



### 5.5.1 *Cenário normal (happy path)*

1. Servidor recebe um pedido de escrita de um cliente (Grpc);
2. Servidor criar um **WriteLock** referente à chave que quer escrever, localmente.
3. Servidor envia mensagem (**WantToWrite**) para todos os servidores do grupo, com o intuito de querer escrever;
4. Os restantes servidores do grupo ao receber a mensagem (**WantToWrite**), caso não estejam no processo de escrita para essa chave.
5. Invalidam a replica local correspondente a par chave valor;
6. Retornam a resposta ao Servidor que quer escrever (**WantToWriteResponse**) com a indicação que a invalidação foi realizada com sucesso;
7. Ao receber todas as respostas do grupo *Spread* exceto a sua e a do serviço de configuração.
8. O Servidor de escrita envia uma última mensagem ao grupo a dizer que a escrita foi realizada (**DataWritten**).
9. Os restantes servidores do grupo ao receber a mensagem (**DataWritten**), efetivam a invalidação anteriormente feita, removendo a replica da chave valor da memória local.
10. O Servidor apaga o **WriteLock** referente à chave que estava a escrever.
11. Por fim, o servidor retorna uma mensagem de OK para o cliente (mensagem sem erro).

### 5.5.2 *Cenário timeout*

Num cenário em que nem todos os servidores respondam ao pedido do servidor querer escrever, o servidor responsável pelo início da operação de escrita, deverá disparar um *timeout* que pode sua vez emitirá uma mensagem de revalidação (**RevalidateData**) da replica anteriormente invalidada para os restantes servidores do grupo.

1. Após execuções dos passos até ao ponto 7 do Cenário normal (happy path);
2. Caso o *timeout* de recebimento de todas as respostas (**WantToWriteResponse**) dispare antes;
3. O Servidor apaga o **WriteLock** referente à chave que estava a escrever.
4. O servidor envia uma mensagem de revalidação para os restantes participantes (**RevalidateData**);
5. Os servidores ao receberem a mensagem (**RevalidateData**) voltam a validar a replica anteriormente invalidada, tornando esta novamente disponível para leitura;
6. O Servidor responsável pela escrita, envia uma mensagem de erro par ao cliente a dizer que a escrita foi abortada e para voltar a tentar.

### 5.5.3 *Cenário conflito de escrita*

Se durante o processo de escrita, o servidor receber a informação de que outro servidor está também ele a entrar em processo de escrita para um outro objeto com o valor do atributo “chave” idêntico, é iniciado o mecanismo de conflito.

O mecanismo de conflito consiste na realização de uma eleição (Mecanismo de eleição) entre os servidores que entraram em conflito, onde é comparando a cardinalidade dos mesmos (Cardinalidade de Servidores).

O servidor que tiver a maior cardinalidade é eleito líder e, conseqüentemente, fica com o poder de escrita, enquanto que o outro servidor retorna a informação ao cliente de que por motivos de conflito a operação não foi realizada.

1. Após execuções dos passos até ao ponto 7 do Cenário normal (happy path);
2. Um outro servidor entra no processo de escrita para um objeto com a mesma chave;
3. O Servidor que inicialmente começou o processo de escrita, recebe uma mensagem de pedido de escrita (**WantToWrite**), neste momento este servidor deteta o conflito e envia uma mensagem de conflito de escrita (**WriteConflict**) para os restantes participantes do grupo.
  - a. Por sua vez, o próprio servidor, executa o próprio *callback* que corresponde a notificação de um conflito de escrita, e executa o processo de eleição entre ele próprio e o servidor que enviou a mensagem de pedido de escrita (**WantToWrite**).
  - b. Dependente do resultado da eleição executa o respetivo comportamento desejado, mais a frente indicada, ponto 5.
4. Servidor remoto em conflito recebe a mensagem de conflito (**WriteConflict**) e executa o Mecanismo de eleição entre ele próprio e o servidor que enviou a mensagem.
5. Caso perca a eleição:
  - a. Invalida a par chave valor referente ao pedido.
  - b. Apaga o **WriteLock** referente à chave que estava a escrever.
  - c. Envia mensagem ao cliente a dizer para tentar novamente.
6. Caso ganhe a eleição:
  - a. Escreve uma replica local;
  - b. Envia uma mensagem ao grupo a dizer que a escrita foi realizada (**DataWritten**).
  - c. Apaga o **WriteLock** referente à chave que estava a escrever.
  - d. Informa o cliente que a escrita foi realizada.

## 5.6 Operação de Leitura

Na operação de leitura, inicialmente o cliente envia esse pedido para o servidor com um atributo “chave”. O servidor procura localmente na sua base de dados por um objeto com uma “chave” idêntica e, caso encontre, devolve de imediato ao cliente o seu valor.

Caso o servidor não encontre localmente, irá entrar no processo de procura no grupo. Este processo consiste em perguntar, através de *multicast* (**AskData**), se algum elemento do grupo contém um objeto com a “chave” pretendida pelo cliente. Caso algum servidor responder, o servidor que recebeu o pedido do cliente escreve esse valor localmente e responde ao cliente. Caso nenhum servidor responder, o servidor que recebeu o pedido informa o cliente que não existe nenhum objeto com o atributo “chave” pretendido.

Caso a informação pedida esteja invalidada, ou seja, durante um processo de escrita de alguém, o servidor não retorna a informação pretendida.

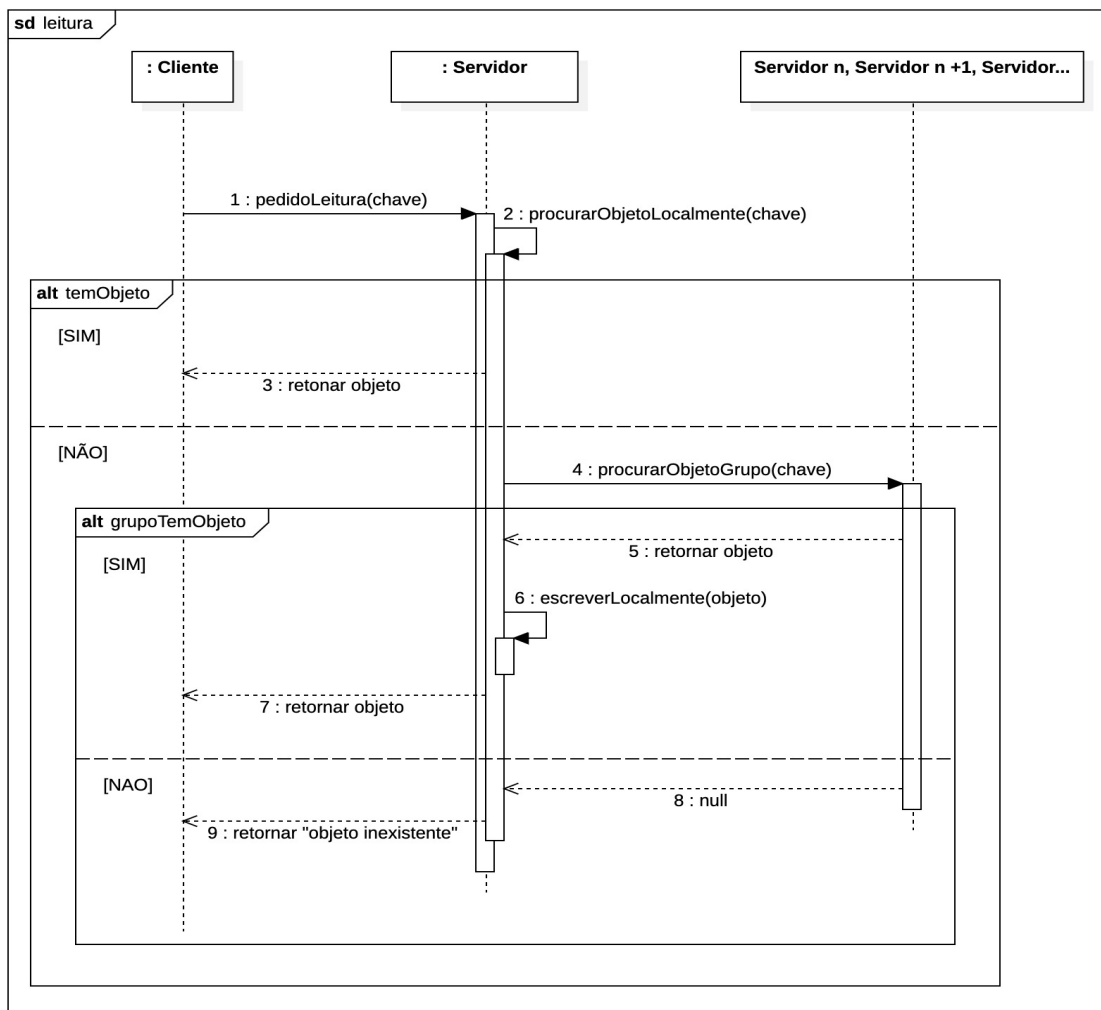


Figura 5 - Diagrama de Sequência – Operação de Leitura

## 5.7 Startup

Sempre que um servidor se junta ao grupo Spread, é iniciado o processo de *startup*. Neste processo, o novo servidor eleger um coordenador com base na maior cardinalidade entre os servidores já presentes no grupo, Mecanismo de eleição.

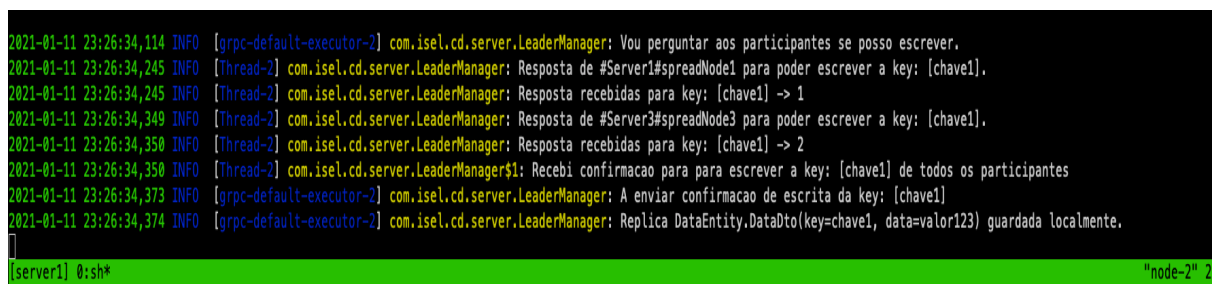
Ao eleger o coordenador, é iniciado o processo de atualização de informação, que consiste em começar por invalidar todas as replicas locais e de seguida enviar a chave dessas replicas ao coordenador elegido (**StartupRequestUpdate**). Caso o coordenador elegido tenha essas replicas, o coordenador retorna as replicas para o servidor que executa o *startup* (**StartupResponseUpdate**). Quando o servidor recebe essas replicas, as replicas encontradas são atualizadas e persistidas localmente (novamente validadas), as não encontradas ficam invalidadas, mas não removidas até que haja uma outra escrita para aquela chave.

## 6 Demonstrações

### 6.1 Demonstração escrita

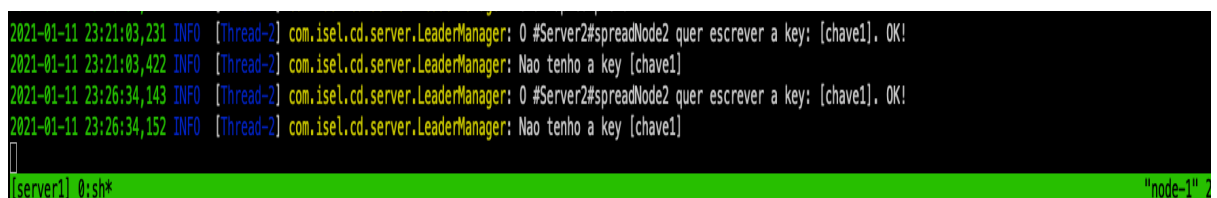
```
Wait for user input!. [w <key> <value> , r <key>, exit]
w chave1 valor123
Write data: valor123 with key: chave1
```

Figura 6 - Input da operação de escrita de um cliente



```
2021-01-11 23:26:34,114 INFO [grpc-default-executor-2] com.isel.cd.server.LeaderManager: Vou perguntar aos participantes se posso escrever.
2021-01-11 23:26:34,245 INFO [Thread-2] com.isel.cd.server.LeaderManager: Resposta de #Server1#spreadNode1 para poder escrever a key: [chave1].
2021-01-11 23:26:34,245 INFO [Thread-2] com.isel.cd.server.LeaderManager: Resposta recebidas para key: [chave1] -> 1
2021-01-11 23:26:34,349 INFO [Thread-2] com.isel.cd.server.LeaderManager: Resposta de #Server3#spreadNode3 para poder escrever a key: [chave1].
2021-01-11 23:26:34,350 INFO [Thread-2] com.isel.cd.server.LeaderManager: Resposta recebidas para key: [chave1] -> 2
2021-01-11 23:26:34,350 INFO [Thread-2] com.isel.cd.server.LeaderManager$: Recebi confirmacao para para escrever a key: [chave1] de todos os participantes
2021-01-11 23:26:34,373 INFO [grpc-default-executor-2] com.isel.cd.server.LeaderManager: A enviar confirmacao de escrita da key: [chave1]
2021-01-11 23:26:34,374 INFO [grpc-default-executor-2] com.isel.cd.server.LeaderManager: Replica DataEntity.DataDto(key=chave1, data=valor123) guardada localmente.
[server1] 0:sh* "node-2" 2
```

Figura 7 - Servidor 2 recebeu pedido de escrita e vai perguntar aos outros servidores se pode escrever



```
2021-01-11 23:21:03,231 INFO [Thread-2] com.isel.cd.server.LeaderManager: O #Server2#spreadNode2 quer escrever a key: [chave1]. OK!
2021-01-11 23:21:03,422 INFO [Thread-2] com.isel.cd.server.LeaderManager: Nao tenho a key [chave1]
2021-01-11 23:26:34,143 INFO [Thread-2] com.isel.cd.server.LeaderManager: O #Server2#spreadNode2 quer escrever a key: [chave1]. OK!
2021-01-11 23:26:34,152 INFO [Thread-2] com.isel.cd.server.LeaderManager: Nao tenho a key [chave1]
[server1] 0:sh* "node-1" 2
```

Figura 8 - Servidor 1 confirma que não tem presente essa chave e dá o OK

```

2021-01-11 23:21:03,284 INFO [Thread-2] com.isel.cd.server.LeaderManager: O #Server2#spreadNode2 quer escrever a key: [chave1]. OK!
2021-01-11 23:21:03,309 INFO [Thread-2] com.isel.cd.server.LeaderManager: Nao tenho a key [chave1]
2021-01-11 23:26:34,193 INFO [Thread-2] com.isel.cd.server.LeaderManager: O #Server2#spreadNode2 quer escrever a key: [chave1]. OK!
2021-01-11 23:26:34,198 INFO [Thread-2] com.isel.cd.server.LeaderManager: Nao tenho a key [chave1]
[server1] 0:sh* "node=3" 2

```

Figura 9 - Servidor 3 confirma que não tem presente essa chave e dá o OK

```

database1=# select * from server_data;
 key | invalidate | data
-----+-----+-----
(0 rows)

database1=# select * from server_data;
 key | invalidate | data
-----+-----+-----
(0 rows)

database1=# select * from server_data;
 key | invalidate | data
-----+-----+-----
(0 rows)

database1=#

```

---

```

database1=# select * from server_data;
 key | invalidate | data
-----+-----+-----
(0 rows)

database1=# select * from server_data;
 key | invalidate | data
-----+-----+-----
 chave1 |          | {"data": "valor123"}
(1 row)

database1=# select * from server_data;
 key | invalidate | data
-----+-----+-----
 chave1 |          | {"data": "valor123"}
(1 row)

database1=#

```

---

```

database1=# delete from server_data;
DELETE 1
database1=# select * from server_data;
 key | invalidate | data
-----+-----+-----
(0 rows)

database1=# select * from server_data;
 key | invalidate | data
-----+-----+-----
(0 rows)

database1=# select * from server_data;
 key | invalidate | data
-----+-----+-----
(0 rows)

database1=#

```

Figura 10 - Base de Dados dos três servidores após escrita

## 6.2 Demonstração leitura com replica local

```
Wait for user input!. [w <key> <value> , r <key>, exit]
r chave1
Read data with key: chave1
Data: key: "chave1"
data: "valor123"
```

Figura 11 - Input de operação de leitura de um cliente e resposta do servidor

```
2021-01-11 23:24:44,023 INFO [grpc-default-executor-1] com.isel.cd.server.ClientService: Cliente pediu informacao com [chave1]
2021-01-11 23:24:44,096 INFO [grpc-default-executor-1] com.isel.cd.server.ClientService: Tenho replica local DataEntity.DataDto(key=chave1, data=valor123)
2021-01-11 23:24:44,097 INFO [grpc-default-executor-1] com.isel.cd.server.ClientService: A enviar resposta para o cliente: DataEntity.DataDto(key=chave1, data=valor123)
[server1] 0:sh*
```

Figura 12 - Servidor recebeu pedido de leitura e devolveu valor ao cliente

## 6.3 Demonstração leitura sem replica local

```
2021-01-11 23:31:47,966 INFO [grpc-default-executor-0] com.isel.cd.server.ClientService: Cliente pediu informacao com [chave1]
2021-01-11 23:31:47,974 INFO [grpc-default-executor-0] com.isel.cd.server.ClientService: Não tenho a [chave1], vou pedir...
2021-01-11 23:31:47,978 INFO [grpc-default-executor-0] com.isel.cd.server.LeaderManager: Vou pedir a todos, se alguém tem: [chave1]
2021-01-11 23:31:48,145 INFO [Thread-2] com.isel.cd.server.LeaderManager: Recebi resposta da informacao que pedi: DataEntity.DataDto(key=chave1, data=valor123)
2021-01-11 23:31:48,187 INFO [Thread-2] com.isel.cd.server.LeaderManager: Informacao pedida recebida : DataEntity.DataDto(key=chave1, data=valor123)
2021-01-11 23:31:48,985 INFO [grpc-default-executor-0] com.isel.cd.server.LeaderManager: Guardei uma copia local de: DataEntity(key=chave1, invalidate=null, data=DataEntity.Data(data=valor123))
2021-01-11 23:31:49,007 INFO [grpc-default-executor-0] com.isel.cd.server.ClientService: A enviar resposta para o cliente: DataEntity.DataDto(key=chave1, data=valor123)
[server1] 0:sh* "node-1" 23:31 11-Jan-21
```

Figura 13 - Servidor 1 recebeu o pedido de leitura mas não tem essa chave, procede a pedir ao grupo

```
2021-01-11 23:31:47,985 INFO [Thread-2] com.isel.cd.server.LeaderManager: #Server1#spreadNode1 est? a procura da informacao [chave1]
2021-01-11 23:31:47,991 INFO [Thread-2] com.isel.cd.server.LeaderManager: Eu tenho a informacao: DataEntity(key=chave1, invalidate=null, data=DataEntity.Data(data=valor123)), vou enviar para : #Server1#spreadNode1
[server1] 0:sh* "node-2" 23:32 11-Jan-21
```

Figura 14 - Servidor 2 recebeu pedido do Servidor 1, confirmou que tem essa chave e respondeu

```

database1=# select * from server_data;
 key | invalidate | data
-----+-----+-----
(0 rows)

database1=# select * from server_data;
 key | invalidate | data
-----+-----+-----
(0 rows)

database1=# select * from server_data;
 key | invalidate | data
-----+-----+-----
(0 rows)

database1=# 

```

```

key | invalidate | data
-----+-----+-----
chave1 |          | {"data": "valor123"}
(1 row)

database1=# select * from server_data;
 key | invalidate | data
-----+-----+-----
chave1 |          | {"data": "valor123"}
(1 row)

database1=# select * from server_data;
 key | invalidate | data
-----+-----+-----
chave1 |          | {"data": "valor123"}
(1 row)

database1=# 

```

```

(0 rows)

database1=# select * from server_data;
 key | invalidate | data
-----+-----+-----
(0 rows)

database1=# select * from server_data;
 key | invalidate | data
-----+-----+-----
(0 rows)

database1=# select * from server_data;
 key | invalidate | data
-----+-----+-----
chave1 |          | {"data": "valor123"}
(1 row)

database1=# 

```

Figura 15 - Replicação da replica na bases de dados

## 6.4 Demonstração de timeout escrita.

Para simular o *timeout*, iremos forçar que um dos servidores do grupo não responda ao pedido de escrita do servidor que quer escrever.

```
Wait for user input!. [w <key> <value> , r <key>, exit]
w chave1 ola_mundo
Write data: ola_mundo with key: chave1
Tenta outra vez.
```

Figura 16 - Mensagem de erro para o cliente

```
database1=# select * from server_data;
 key | invalidate | data
-----+-----+-----
 chave1 | t          | {"data": "ola_mundo"}
(1 row)

database1=#
```

Figura 17 - Invalidação durante a escrita

```
2021-01-11 23:36:54,081 INFO [grpc-default-executor-3] com.isel.cd.server.ClientService: Um cliente pediu para escrever: key: "chave1"
data: "ola_mundo"
2021-01-11 23:36:54,082 INFO [grpc-default-executor-3] com.isel.cd.server.LeaderManager: Vou perguntar aos participantes se posso escrever.
2021-01-11 23:36:54,349 INFO [Thread-2] com.isel.cd.server.LeaderManager: Resposta de #Server3#spreadNode3 para poder escrever a key: [chave1].
2021-01-11 23:36:54,350 INFO [Thread-2] com.isel.cd.server.LeaderManager: Resposta recebidas para key: [chave1] -> 1
2021-01-11 23:37:04,122 WARN [grpc-default-executor-3] com.isel.cd.server.LeaderManager: Timeout reached!
2021-01-11 23:37:04,123 INFO [grpc-default-executor-3] com.isel.cd.server.LeaderManager: Nem todos os participantes responderam, abortar escrita.
2021-01-11 23:37:04,123 INFO [grpc-default-executor-3] com.isel.cd.server.LeaderManager: A enviar mensagem para revalidacao da key: [chave1]
[server1] 0:shh "node-2" 23:37 11-Jan-21
```

Figura 18 - Timeout do servidor



```

key | invalidate | data
-----+-----+-----
chave1 | f | {"data": "ola_mundo"}
(1 row)

database=# select * from server_data;
key | invalidate | data
-----+-----+-----
chave1 | f | {"data": "ola_mundo"}
(1 row)

database=# select * from server_data;
key | invalidate | data
-----+-----+-----
chave1 | f | {"data": "ola_mundo"}
(1 row)

database=#

database=# select * from server_data;
key | invalidate | data
-----+-----+-----
chave1 | t | {"data": "ola_mundo"}
(1 row)

database=# select * from server_data;
key | invalidate | data
-----+-----+-----
chave1 | f | {"data": "ola_mundo"}
(1 row)

database=#

```

Figura 19 - Revalidação após *timeout*

### 6.5 Demonstração da resolução de um conflito de escrita.

Para simular o conflito, iremos forçar que um dos servidores do grupo não responda ao pedido de escrita, e durante a espera da resposta (aumentamos o *timeout*) executamos o pedido de escrita para outro servidor.

```

Wait for user input!. [w <key> <value> , r <key>, exit]
w chave1 valor1
Write data: valor1 with key: chave1
Tenta outra vez.

```

Figura 20 - Mensagem de erro para cliente

```

* chave1 ISEL_RULES
Write data: ISEL_RULES with key: chave1
Start operations!
Wait for user input!. [w <key> <value> , r <key>
|

```

Figura 21 - Mensagem sucesso cliente

```

2021-01-11 23:41:11,868 INFO [grpc-default-executor-4] com.isel.cd.server.ClientService: Um cliente pediu para escrever: key: "chave1"
data: "valor1"

2021-01-11 23:41:11,869 INFO [grpc-default-executor-4] com.isel.cd.server.LeaderManager: Vou perguntar aos participantes se posso escrever.
2021-01-11 23:41:12,107 INFO [Thread-2] com.isel.cd.server.LeaderManager: Resposta de #Server3#spreadNode3 para poder escrever a key: [chave1].
2021-01-11 23:41:12,108 INFO [Thread-2] com.isel.cd.server.LeaderManager: Resposta recebidas para key: [chave1] -> 1
2021-01-11 23:41:12,244 WARN [Thread-2] com.isel.cd.server.LeaderManager: Eu estou no processo de escrever para a key: [chave1], FIGHT!
2021-01-11 23:41:12,247 INFO [Thread-2] com.isel.cd.server.LeaderManager$1: Conflito com #Server3#spreadNode3!
2021-01-11 23:41:12,271 WARN [grpc-default-executor-4] com.isel.cd.server.LeaderManager: Conflict!
2021-01-11 23:41:12,272 INFO [grpc-default-executor-4] com.isel.cd.server.LeaderManager: My cardinality: 4 Conflict cardinality: 6
2021-01-11 23:41:12,273 INFO [grpc-default-executor-4] com.isel.cd.server.LeaderManager: Eu nao vou escrever, eu tenho a cardinalidade mais pequena

[server1] 0:sh*

```

Figura 22 - Servidor que perdeu eleição

```

2021-01-11 23:41:11,983 INFO [grpc-default-executor-0] com.isel.cd.server.ClientService: Um cliente pediu para escrever: key: "chave1"
data: "ISEL_RULES"

2021-01-11 23:41:12,092 INFO [grpc-default-executor-0] com.isel.cd.server.LeaderManager: Vou perguntar aos participantes se posso escrever.
2021-01-11 23:41:12,445 INFO [Thread-2] com.isel.cd.server.LeaderManager: Conflito recebido para a key: [chave1] do #Server2#spreadNode2
2021-01-11 23:41:12,445 INFO [Thread-2] com.isel.cd.server.LeaderManager$1: Conflito com #Server2#spreadNode2!
2021-01-11 23:41:12,448 WARN [grpc-default-executor-0] com.isel.cd.server.LeaderManager: Conflict!
2021-01-11 23:41:12,449 INFO [grpc-default-executor-0] com.isel.cd.server.LeaderManager: My cardinality: 6 Conflict cardinality: 4
2021-01-11 23:41:12,449 INFO [grpc-default-executor-0] com.isel.cd.server.LeaderManager: Eu tenho a cardinalidade maior eu vou escrever
2021-01-11 23:41:12,626 INFO [grpc-default-executor-0] com.isel.cd.server.LeaderManager: A enviar confirmacao de escrita da key: [chave1]
2021-01-11 23:41:12,627 INFO [grpc-default-executor-0] com.isel.cd.server.LeaderManager: Informa??o DataEntity.DataDto(key=chave1, data=ISEL_RULES) salva localmente, apos conflito com: #Server2#spreadNode2.

[server1] 0:sh*
"node-3" 23:42 11-Jan-2

```

Figura 23 - Servidor que ganhou eleição

```

key | invalidate | data
-----+-----+-----
(0 rows)

database1=# select * from server_data;
key | invalidate | data
-----+-----+-----
(0 rows)

database1=# select * from server_data;
key | invalidate | data
-----+-----+-----
chave1 |          | {"data": "ISEL_RULES"}
(1 row)

database1=# 

```

---

```

database1=# select * from server_data;
key | invalidate | data
-----+-----+-----
chave1 |          | {"data": "ola_mundo"}
(1 row)

database1=# select * from server_data;
key | invalidate | data
-----+-----+-----
chave1 |          | {"data": "ola_mundo"}
(1 row)

database1=# select * from server_data;
key | invalidate | data
-----+-----+-----
(0 rows)

database1=# 

```

---

```

database1=# select * from server_data;
key | invalidate | data
-----+-----+-----
chave1 | f          | {"data": "ola_mundo"}
(1 row)

database1=# select * from server_data;
key | invalidate | data
-----+-----+-----
chave1 | f          | {"data": "ola_mundo"}
(1 row)

database1=# select * from server_data;
key | invalidate | data
-----+-----+-----
(0 rows)

database1=# 

```

Figura 24 - Estado final da base de dados, apenas uma entrada para a chave1

## 6.6 Demonstração do startup

```

2021-01-11 23:46:26,971 INFO [Thread-2] com.isel.cd.server.LeaderManager: #Server1#spreadNode1 est? a procura da informacao [chave]
2021-01-11 23:46:26,970 INFO [Thread-2] com.isel.cd.server.LeaderManager: Nao tenho [chave]
2021-01-11 23:46:52,333 INFO [grpc-default-executor-5] com.isel.cd.server.ClientService: Cliente pediu informacao com [chave1]
2021-01-11 23:46:52,333 INFO [grpc-default-executor-5] com.isel.cd.server.ClientService: N7o tenho a [chave1], vou pedir...
2021-01-11 23:46:52,339 INFO [grpc-default-executor-5] com.isel.cd.server.LeaderManager: Vou pedir a todos, se algum tem: [chave1]
2021-01-11 23:46:52,600 INFO [Thread-2] com.isel.cd.server.LeaderManager: Recebi resposta da informacao que pedi: DataEntity.DataDto(key=chave1, data=ISEL_RULES)
2021-01-11 23:46:52,600 INFO [Thread-2] com.isel.cd.server.LeaderManager: Informacao pedida recebida : DataEntity.DataDto(key=chave1, data=ISEL_RULES)
2021-01-11 23:46:52,653 INFO [grpc-default-executor-5] com.isel.cd.server.LeaderManager: Guardei uma copia local de: DataEntity(key=chave1, invalidate=null, data=DataEntity.Data(data=ISEL_RULES))
2021-01-11 23:46:52,653 INFO [grpc-default-executor-5] com.isel.cd.server.ClientService: A enviar resposta para o cliente: DataEntity.DataDto(key=chave1, data=ISEL_RULES)
2021-01-11 23:47:12,744 INFO [Thread-2] com.isel.cd.server.SpreadMessageListener: #Server1#spreadNode1 DISCONNECTED. group size = 3
2021-01-11 23:47:15,244 INFO [Thread-2] com.isel.cd.server.SpreadMessageListener: #Server3#spreadNode3 DISCONNECTED. group size = 2
2021-01-11 23:47:39,115 INFO [grpc-default-executor-5] com.isel.cd.server.ClientService: Um cliente pediu para escrever: key: "chave1" data: "olamundo"
2021-01-11 23:47:39,116 INFO [grpc-default-executor-5] com.isel.cd.server.LeaderManager: Sou o unico no grupo, vou escrever.
[server1] 8ish*
"node-2" 23:47 11-Jan-21

```

Figura 25 - Escrita para um servidor, único no grupo

```

chave1 | | {"data": "ISEL_RULES"}
(1 row)

database1=# select * from server_data;
 key | invalidate | data
-----+-----+-----
chave1 | | {"data": "ISEL_RULES"}
(1 row)

database1=# select * from server_data;
 key | invalidate | data
-----+-----+-----
chave1 | | {"data": "ISEL_RULES"}
(1 row)

database1=# 

```

---

```

database1=# select * from server_data;
 key | invalidate | data
-----+-----+-----
(0 rows)

database1=# select * from server_data;
 key | invalidate | data
-----+-----+-----
chave1 | | {"data": "ISEL_RULES"}
(1 row)

database1=# select * from server_data;
 key | invalidate | data
-----+-----+-----
chave1 | | {"data": "olamundo"}
(1 row)

database1=# 

```

Figura 26 - Base de dados com registos diferentes para a mesma chave

```

2021-01-11 23:49:51,471 INFO [Thread-2] com.isel.cd.server.SpreadMessageListener: #Server3#spreadNode3 JOINED GROUP. group size = 3
2021-01-11 23:49:51,472 INFO [Thread-2] com.isel.cd.server.LeaderManager: A eleger coordenador para me ajudar no startup.
2021-01-11 23:49:51,472 INFO [Thread-2] com.isel.cd.server.LeaderManager: Elegi o coordenador #Server2#spreadNode2 com cardinalidade: 4
2021-01-11 23:49:52,291 INFO [main] com.isel.cd.server.ServerApplication: type exit to exit!
2021-01-11 23:49:52,485 INFO [Thread-2] com.isel.cd.server.LeaderManager: Toda a informacao local foi invalidada
2021-01-11 23:49:52,486 INFO [Thread-2] com.isel.cd.server.LeaderManager: Vou pedir startup para a informacao: [DataEntity.DataDto(key=chave1, data=ISEL_RULES)]
2021-01-11 23:49:52,946 INFO [Thread-2] com.isel.cd.server.LeaderManager: Recebi informacao de startup do leader: #Server2#spreadNode2 = StartupResponseUpdate(dataDtoList=[DataEntity.DataDto(key=chave1, data=olamundo)])
2021-01-11 23:49:52,989 INFO [Thread-2] com.isel.cd.server.LeaderManager: Validada: DataEntity(key=chave1, invalidate=true, data=DataEntity.Data(data=olamundo))
2021-01-11 23:49:53,010 INFO [Thread-2] com.isel.cd.server.LeaderManager: Startup completo.
[server1] 0:sh*
"node-3" 23:51 11-Jan-21

```

Figura 27 - Pedido de startup e eleição de coordenador

```

2021-01-11 23:49:52,744 INFO [Thread-2] com.isel.cd.server.LeaderManager: COORDENADOR==Server2#spreadNode2: Recebi o pedido de ajuda para o startup de: #Server3#spreadNode3
2021-01-11 23:49:52,753 INFO [Thread-2] com.isel.cd.server.LeaderManager: Vou devolver a seguinte informacao de startup: [DataEntity.Dto(key=chave1, data=olamundo)]
server1] 0:sh* "node-2" 23:51 11-Jan-21

```

Figura 28 – Coordenador

```

database1=# select * from server_data;
  key | invalidate |      data
-----+-----+-----
 chave1 | f          | {"data": "olamundo"}
(1 row)

database1=#

```



```

database1=# select * from server_data;
  key | invalidate |      data
-----+-----+-----
 chave1 |          | {"data": "ISEL_RULES"}
(1 row)

database1=# select * from server_data;
  key | invalidate |      data
-----+-----+-----
 chave1 |          | {"data": "olamundo"}
(1 row)

database1=# select * from server_data;
  key | invalidate |      data
-----+-----+-----
 chave1 |          | {"data": "olamundo"}
(1 row)

database1=#

```

Figura 29 - Estado final base de dados

## 7 Conclusões

O *Spread Toolkit*, demonstrou ser uma ferramenta útil no sentido em que forneceu, no processo de desenvolvimento de um sistema em computação distribuída, com os seguintes pontos fortes:

- Um “ponto central” que permite ter a percepção dos participantes do grupo (servidores/nós) presentes e ativos;
- Comunicação *multicast* com um alto nível de confiança no envio das mesmas;
- Eleição de um *leader* entre um ou mais servidores sem que haja partilha de mensagens.

Um ponto fraco notado, é relativamente a não ter presente um mecanismo de *response/reply* tornando um pouco mais difícil a implementação de algumas funcionalidades.

A distribuição de informação (replicação) ocorre naturalmente a pedido de cada cliente, sendo que a replica mais pedida é facilmente replicada por mais servidores e assim exigindo um menor custo computacional e de armazenamento, ao invés, no caso do armazenamento de uma replicação forçada de todas as replicas por todos os servidores do grupo.

Numa anterior iteração nossa, foi implementado uma distribuição “sequencial” e forçada das replicas por todos os participantes aquando da escrita de uma réplica, que inocentemente achamos ser uma abordagem correta, no entanto não tivemos em consideração as falácias da computação distribuída, e ao assumir um modelo de consensos simplista como esse não poderíamos garantir como toda a certeza de que a informação replicada era a mais recente/válida.

No entanto, através do *Spread* é possível garantir na escrita, uma fácil eleição de quem escreve a informação em caso de conflito, garantindo que apenas um participante do grupo escreve essa replica, ao contrário do que acontecia na implementação anteriormente referida, na qual podemos dizer, que não tínhamos 100% certeza do que estava a acontecer.