

Relatório da 2ª Entrega

Projeto *ForkExec*

Grupo - T08

GitHub:

https://github.com/tecnico-distsys/T08-ForkExec

|  |  |  |
| --- | --- | --- |
| /var/folders/jg/x_kymnbj70d3_6mmfczc24940000gn/T/com.microsoft.Word/WebArchiveCopyPasteTempFiles/0?e=1562198400&v=beta&t=W9Kkgm6368FIFBlHd2EXMNlrcH58PdpczXCXHN-zPN0 | 87641 | Carolina Carreira |
|  |  |  |
|  |  |  |
| /var/folders/jg/x_kymnbj70d3_6mmfczc24940000gn/T/com.microsoft.Word/WebArchiveCopyPasteTempFiles/0?e=1562198400&v=beta&t=cA-a_dt3ZPlm-wzFk1Kow3kwfheWtgZwZJ2NbJlmha0 | 87691 | Miguel Barros |

Sistemas Distribuídos

2º Semestre

2019

1. Modelo de Faltas

– O sistema é assíncrono e a comunicação pode omitir mensagens.

– Não há garantia de receção FIFO.

– Existem N gestores de réplica e N é constante e igual a 3 (para a demonstração).

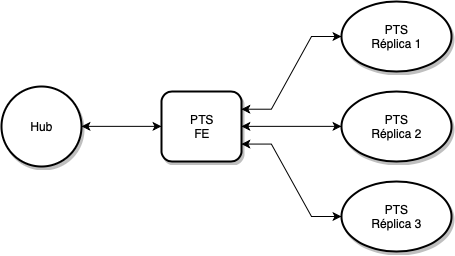
– Para o nosso modelo são necessárias 2\*f+1 réplicas para tolerar f falhas de réplicas. Como nós temos 3 réplicas, vamos tolerar 1 falha de réplica (na demonstração).

– No máximo, assumimos uma minoria de gestores de réplica em falha em simultâneo.

– Os gestores de réplica podem falhar silenciosamente, mas não arbitrariamente.

– Qualquer servidor pode ter uma falta por paragem, enquanto que a rede omitir ou atrasar mensagens.

1. Figura da solução de tolerância a faltas



1. Descrição da figura e breve explicação da solução

O Hub em vez de comunicar diretamente com o PTS comunica com um Front End (FE) e este sim faz a gestão das réplicas do PTS.

Réplicas aceitam apenas operações de leitura ou escrita, por isso PTS FE faz a 'tradução' de operações complexas (exem. incremento) do Hub em operações de leitura e escrita para as réplicas.

Também é no Front End que a maioria das verificações são efetuadas, dado que, na nossa solução, as operações de *read* e *write* assumem que os argumentos estão corretos. Tomamos esta decisão para simplificar nosso código.

Cada réplica, para cada registo guarda o valor e uma tag.

A tag corresponde ao número de sequência de escrita que deu origem ao registo. Quanto maior a tag mais recente o registo.

A cada leitura:

FE

- envia read() para todos

- espera que *Q(quorum)*respondam

- escolhe resposta com maior tag

A cada escrita:

FE

- executa uma leitura para obter a maxTag e para obter valor atual da instância

- envia write com newTag = maxTag + 1

- espera *Q(quorum)* ACKS

- retorna valor escrito ao cliente

Para todas estas operações o:

Q(*quorum*)> |N/2|

sendo:

• Q um inteiro;

• N o número de réplicas;

Dado que temos 3 réplicas o nosso Q(*quorum*) é de 2.

O Hub suporta as operações:

* **activateAccount** - ativa uma conta de email
* **loadAccount** - carregar a conta com pontos
* **pointsBalance** - obter pontos de uma conta

Operação não é critica e se der resultado desatualizado pode-se sempre voltar a chamar sem consequências negativas (idempotente).

1. Descrição de otimizações/simplificações

No protocolo QC original a tag tinha um Cid.

Na nossa solução a tag não necessita de Client ID pois só há um cliente, o FE.

No protocolo QC todas as escritas são sincronizadas.

Na nossa solução apenas as escritas na mesma 'conta' são sincronizadas.

Verificações são efetuadas no Front End na nossa solução, as operações de *read* e *write* assumem que os argumentos estão corretos. Tomamos esta decisão para simplificar nosso código.

Removemos a excepçõe EmailAlreadyExistsFault pois na nossa implementação, só com *read* e *write* não temos maneira de saber se o email já está registado. Esta impossibilidade deve-se ao facto de que se o *write* for chamado com um email que já existe este não falha, nem lança excepção, apenas adiciona a conta. O mesmo acontece com o *read*.