TÉCNICO LISBOA

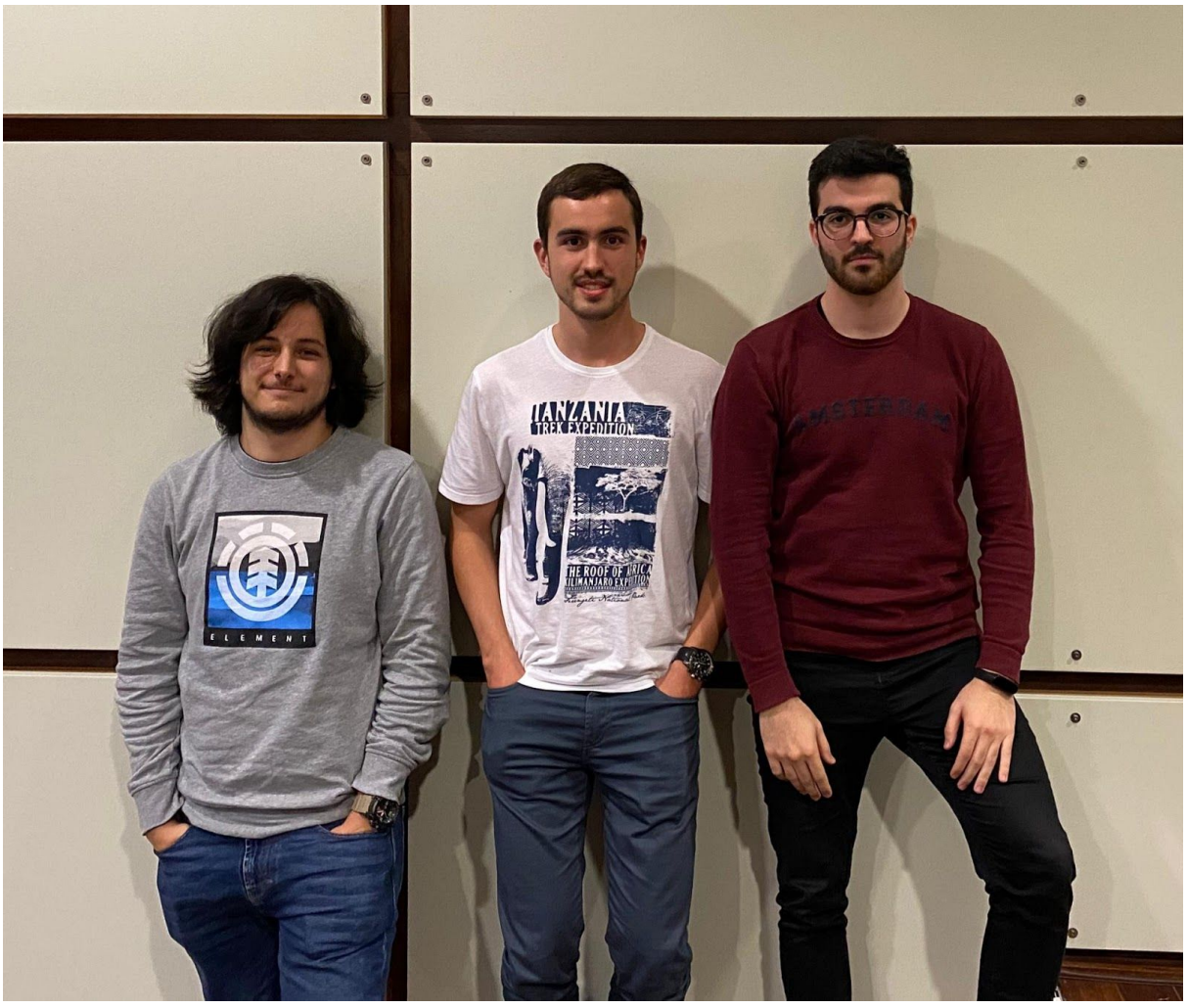
C01

Trabalho realizado por:

Alexandre Mota, 90585

Daniel Lopes, 90590

Duarte Matias, 90596



Definição do modelo de faltas:

No planeamento deste projeto inicialmente analisámos que características do gossip seriam convenientes ao projeto e quais é que não seriam, isto é, que faltas podemos tolerar no projeto e por consequência quais não poderíamos tolerar:

- **Sincronização de informação entre Depots:**
    - Neste projeto aceitamos que num intervalo de tempo razoavelmente pequeno, seria aceitável que houvesse discrepâncias entre a informação presente em Depots diferentes enquanto o protocolo Gossip Architecture resolve esta inconsistência. Aceitamos isto pois não é possível garantirmos que para um determinado instante o Depot contém a informação atualizada de outro.

- **Leituras inconsistentes pelo mesmo cliente:**
    - Dado o domínio da aplicação não é aceitável que o cliente perca observações durante a utilização do programa, isto é quando o pedido é efetuado a um servidor Depot por parte de um cliente seeker, a resposta deverá sempre conter as observações previamente enviadas.

- **Perda de mensagens:**
    - No caso de ocorrerem perdas de mensagens na rede por qualquer que seja o motivo, tal causaria com que o período de tempo no qual dois depots têm informação diferente seja maior, no entanto como referido no primeiro ponto caso isto ocorra num intervalo de tempo suficientemente pequeno isto é aceitável.

- **Depot desatualizado relativamente ao seeker:**
    - É possível que durante a execução, que o seeker altere o depot a quem dirige as queries, nesta situação levanta-se a possibilidade que a resposta à última query tenha um timestamp mais recente que o do Depot novo. Como consequência disso surge a necessidade de garantir que o seeker não perde informação entre respostas; a solução para isto é o Depot reter a resposta até obter uma versão igual ou superior à da última resposta que o seeker recebeu.

Implementação do protocolo:

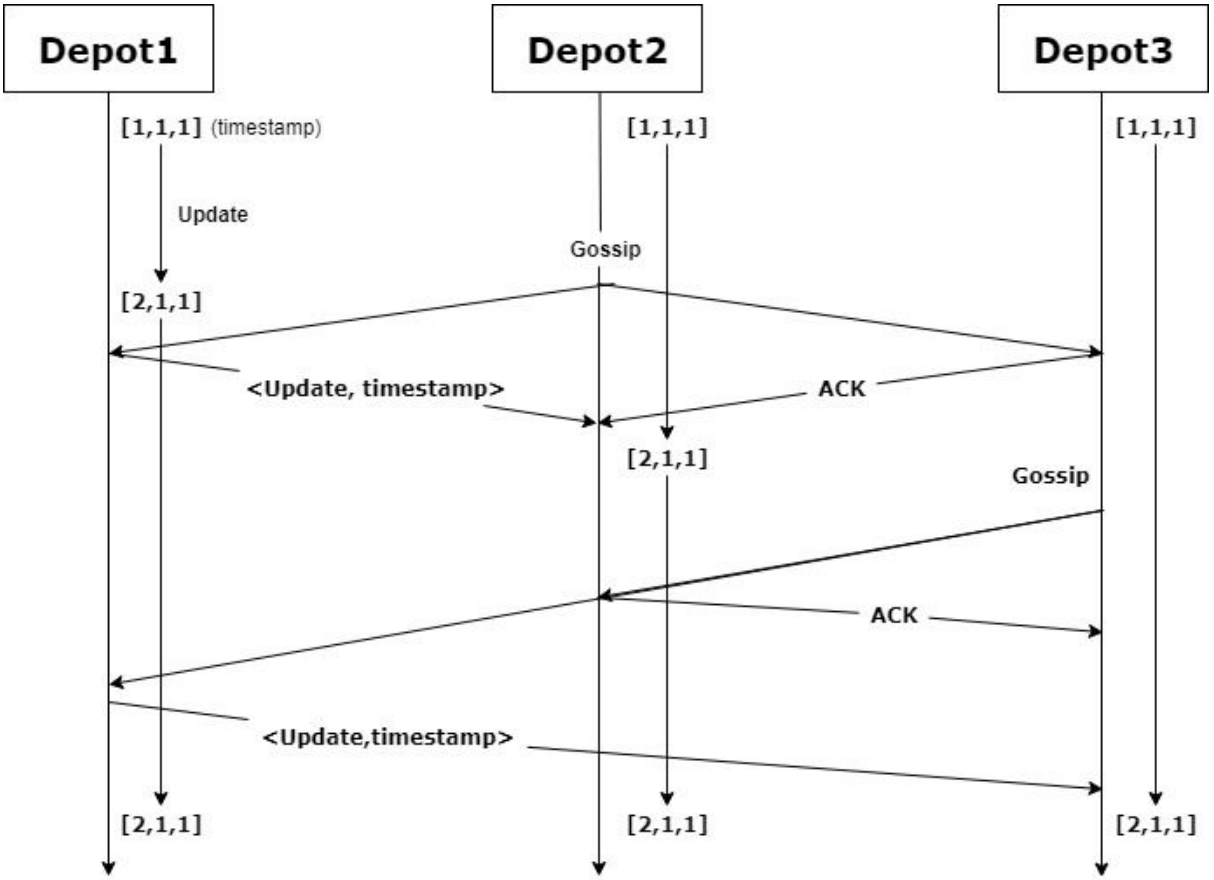


Fig. 1 - Implementação do protocolo Gossip Architecture

Através do protocolo Gossip architecture (exemplificado na figura), cada Depot utiliza um timer próprio para periodicamente enviar uma mensagem a todos os outros depots onde os informa da versão da informação que tem de todos os depots, quem recebe esta mensagem caso repare que a versão da sua respetiva informação está desatualizada envia a sua informação. É importante referir que os Depots nunca enviam informação sobre os dados que têm de um terceiro Depot, isto acontece pois eles não podem garantir para esse instante que a informação que têm de outro Depot é está atualizada. Nesta implementação do gossip cada depot quando vai iniciar o seu ciclo de gossip pede updates em vez de enviar o seu conteúdo para todos os outros para os atualizar

Detalhes do Protocolo:

- Inicialmente ao criar um novo Depot i (sendo i um número inteiro que identifica o depot) é necessário este inicializar um timestamp (array de números inteiros), com a posição i-1 do array inicializada a 1 e o resto a 0. Os números no timestamp são representativos do número de updates efetuados sobre esse Depot (indicado pelo valor num índice do array), quando o valor está a 0 significa que não se tem qualquer informação sobre esse depot.

- Durante o funcionamento normal da aplicação periodicamente (30s contados internamente por cada Depot) são realizados ciclos de gossip, e paralelamente a isto updates vão sendo realizados. Como consequência de termos a informação distribuída, existe a possibilidade de, imediatamente após ser realizado um ciclo do gossip, outro Depot receber um update tornando a informação trocada via timestamps sobre este, desatualizada; no entanto para uma janela de tempo de 30s considerou-se que isto é aceitável.

- As mensagens gossip request que têm como objetivo minimizar este problema e permitir manter armazenamento de informação distribuída, consistem apenas de um timestamp e são enviadas para todos os Depots. Após receber uma mensagem existem duas respostas possíveis por parte do Depot: primeiro caso, o caso em que após notar que no timestamp o outro Depot tem a informação desatualizada, ele procede a enviar o seu timestamp e o log como resposta; no segundo caso não são necessárias quaisquer atualizações e portanto apenas um ACK é enviado como resposta.

- Uma situação de particular importância é o caso em que durante a execução um sentry altera o Depot ao qual comunica a informação que recebe do feeder, caso o novo Depot tenha um timestamp anterior ao Sentry neste caso o update fica em espera até se completarem ciclos suficientes de gossip para corrigir isto. O tratamento desta falha foi realizado através do uso de variáveis de log local a cada Depot e de alterações pendentes que esperam uma dada atualização/versão.

Possíveis otimizações:

- No caso dos pedidos de leitura, supondo que durante a execução um seeker envia um pedido de leitura a um Depot cujo timestamp prévio ao timestamp que o seeker possui do seu último pedido de leitura, a situação atual resulta numa espera pela atualização do Depot, no entanto como não sabemos da existência de updates poderia-se guardar a última leitura e reproduzi-la nestas situações para evitar bloqueios.

- Caso um cliente, tanto seeker como sentry, perca ligação ao depot a que está ligado, o cliente poderia conectar a um outro depot disponível automaticamente, isto é, atualmente só tentam reconectar se tiverem um pedido para processar.

- O sistema atual não suporta reinício de depots após falhas sem que o sistema evolua de forma divergente, isto é, caso um depot termine execução abruptamente todos os updates no seu log local são perdidos, e quando reiniciar não lhe será possível obter estes updates.