DAD Project Report

António Coelho 95535
Tomás Tavares 95680
João Ramalho 95599

Abstract

Neste relatório é descrito o BoneyBank, uma aplicação bancária confiável e com tolerância a faltas que consiste num sistem de três camadas. O BoneyBank pretende alcançar uma solução simples que garante consistência e log replication duradoura.

1. Introduction

O serviço é dividido em três camadas. A primeira composta por processos cliente que providenciam uma interface ao utilizador, que o permite alterar o estado da sua conta bancária através de comandos. A segunda camada, usando primary-backup replication garante que os comandos são executados em todas as replicas pela mesma ordem, garantindo então consistência entre replicas. Na terceira camada, processos Boney providenciam uma simples interface à segunda camada para eleição do líder, para isto o Boney implementa um algoritmo de consenso, mais especificamente o Paxos.

2. Client

O cliente executa comandos, enviando-os para todos os servidores do banco, sequencialmente. A forma mais fácil de o garantir é esperando por todas as respostas dos servidores, garantindo assim que as mudanças foram aplicadas em todas as réplicas. Esta solução não é a ideal, visto que o cliente só consegue fazer progresso se todos os servidores não se encontrarem frozen.
Assim, a solução a que chegamos foi esperar apenas por uma resposta recebendo as restantes em background. Os perfect channels e o algoritmo de replicação dos servidores do bank garantem que todas as mudanças do comando n - 1 são aplicadas antes das do comando n.

3. Bank

O banco usa replicação primary-backup, sendo o lider eleito usando o serviço boney em intervalos de tempos iguais, chamados slots. Para cada slot o bank propõe como lider ou o da slot anterior, se o failure detector indicar que ele está up, ou o bank com menor id que se pensa estar up. Qualquer que seja o lider eleito pelo boney será, então, o novo líder. O bank recebe do cliente os comandos que deve executar, guardando-os num dicionário com todos os existentes (allCommands) e num set dos ainda não ordenados (unordered). Após isso, se for o coordenador da slot atual, atribui o comando à próxima posição vazia do log e inicia a replicação assíncrona por two-phase commit. Quando o comando é finalmente executado, os banks são notificados e devolvem o resultado da operação ao cliente.

3.1. Two-Phase Commit

Os banks recebem chamadas Tentative para cada comando a ser proposto. No caso do bank ainda não conter esse comando(e.g. a chamada de outro bank chega antes do cliente enviar-lhe o comando), espera até o receber. Para além disso, apenas pode responder a uma chamada Tentative para uma posição x do log, quando já aceitou ou deu commit de comandos para todas as posições y do log, com y menor que x. Desta maneira, asseguramos que o log do primário cresce estritamente, ou seja, nunca precisará de reatribuir um comando a uma posicao do log mais antiga. Se se cumprirem estas restrições temos então de garantir que a chamada veio do lider da slot do assignment e esse lider não mudou entretanto. Finalmente, se no sequence number proposto já existir um comando apenas o substitui pelo novo, se o último tiver sido assigned por um lider mais recente. Caso contrário, simplesmente adiciona o comando ao log e remove-o do set unordered.
Quando o lider recebe uma maioria de acks para um comando, inicia uma chamada de commit. Os backups ao receberem commit de um comando, deixam-os pendentes ate receberem commit de todos os commits antes desse, e só aí aplicam todos os comandos committed que são sequenci-

ais desde o último que foi executado. Podemos ter a certeza que se recebermos um commit para um comando na posição x do log, então brevemente receberemos commits de todos os comandos em posições inferiores do log, uma vez que a maioria de banks a dar ack para esse comando tivera obrigatoriamente de dar ack a comandos para todas as posicoes anteriores do log.

3.2. Cleanup

Quando um bank descobre que se torna o novo lider, tem de realizar um cleanup para ter a certeza que conhece todos os comandos propostos pelo lider anterior e aceites por uma maioria dos backups. Desta maneira, evita inconsistencias tanto ao prevenir enviar comandos para posições do log já committed como não propondo comandos que já foram aceites por uma maioria.
Como tal, o procedimento inicia-se com o novo primário a enviar para todas os backups o valor do último comando que ele executou. Os backups, ao entrar na slot do cleanup, respondem devolvendo todos os comandos do que contêm no log com sequece number superior ao enviado pelo lider. Do lado do lider, ele volta a propor todos os comandos que recebe, assgurando-se que em caso de conflito(i.e. diferentes comandos para uma mesma posicao do log) propoem o comando da assignment slot mais recente. Por fim, altera o valor do seu highestSequenceNumber para o maior valor entre os comandos recebidos das replicas e procede a enviar os comandos que ainda contém no seu set unordered.

4. Boney

Pela sua interface, o Boney recebe compareAndSwap's com o id do lider que um cliente seu propõem, para uma dada posição do log. Se já tiver um valor guardado nessa posição do seu log devolve-a imediatamente ao cliente, senão inicia-se uma nova instância do Paxos com os restantes processos Boney. Para facilitar progreso, apenas o boney que pensa ser coordenador propõe novos valores. Isto descobre-se com ajuda do failure detector que indica que boneys estão suspeitos ou não de estar em baixo em cada slot de tempo e assumindo o boney com id mais baixo que se pensa estar disponível é o coordenador.

5. Perfect Channels

As conexões Bank-Bank, Boney-Boney and Bank-Boney são feitas através de perfect channels que garantem que os processos recebem e enviam todas as mensagens quando fica unfrozen. Esta funcionalidade foi implementada com o auxílio de grpc channel interceptors, que apenas permitem a passagem de mensagens do canal para a aplicação quando o processo se encontra unfrozen. Se por acaso se encontrar frozen, o interceptor bloqueia até receber um signal. Signal este que é enviado pela main thread quando existe uma transição de estado por parte do servidor (frozen para unfrozen).