INF01151 –SISTEMAS OPERACIONAIS II N
SEMESTRE 2020/1
RELATÓRIO TRABALHO PRÁTICO PARTE 2: REPLICAÇÃO E ELEIÇÃO DE LÍDER

Grupo: Andrews L. Rodrigues, Arthur K. Ferreira, Felipe Flores, Matheus F. Kovaleski

---

Complemento da Parte 1

Durante a primeira parte do trabalho não conseguimos implementar algumas funcionalidades, estas foram implementadas durante esta etapa:

Recuperação das últimas N mensagens

Ao se conectar ao servidor e após se registrar, o servidor busca e envia as N últimas mensagens antes de receber novas mensagens do cliente.

Desconexões de clientes

Quando o servidor percebe que um cliente se desconecta, ele remove o registro do mapa de conectados.

Número máximo de dispositivos

Antes de registrar, o servidor verifica se já existe o número máximo de usuários com o mesmo username registrados, se for o caso, notifica o cliente e finaliza a conexão.

Condições de corrida e acessos simultâneos

Estavam faltando alguns mutex para proteger alguns trechos de códigos, todos agora foram arrumado e estão protegidos (inclusive os novos trechos referentes ao T2).

Ambiente de Testes

Ubuntu 20.04.1 LTS - Intel Core i5-4690 - 8GB DDR3 - GCC 9
Ubuntu 18.10 - AMD C-50 - 1.56GB DDR3 - GCC 8
Ubuntu 18.04.5 LTS -AMD® Ryzen 3 2200g -5,58346 GB DDR4 - G++7

Replicação Passiva

Podemos dizer que possui duas etapas, a primeira etapa é quando um novo backup se conecta na réplica primária(RP), a RP precisa transmitir seus estados e dados internos de eventos que aconteceram antes do backup ser iniciado. A segunda etapa é quando o backup já está conectado e a RP réplica novos dados assim que eles vão chegando.

Primeira etapa: toda vez que um novo backup se conecta na RP, a RP passa todos os dados já adquiridos como: grupos, usuários conectados e as últimas N mensagens de todos os grupo para o novo backup. **[*register_new_connection()* no GCServer.cpp, type "backup"]**

Segunda etapa: toda vez que uma nova mensagem chega a RP, ela salva no arquivo, envia a mensagem para os backups e os mesmos salvam em seus arquivos, após isso, apenas o RP envia a nova mensagem para os apps. **[*Send_all()*]**
Quando um novo app se conecta ou desconecta, essa informação é também repassada para os backup para que eles mantenham os mesmos dados da RP. **[*listen_main_server()*]**

Eleição de Líder

A forma de eleição escolhida foi a do algoritmo do anel, o motivo da escolha foi por um maior entendimento do algoritmo pelo grupo. Ficamos com algumas dúvidas do funcionamento do algoritmo bully, e escolhemos o do anel porque conseguimos visualizar melhor de como seria a implementação no contexto do trabalho.

Quando iniciamos uma nova réplica, ela se conecta e se registra a réplica primária. Na réplica primária(RP) temos uma lista chamada ***backup_vector**,* depois que um backup n+1 se registra, a RP adiciona no fim da lista, avisa o backup n da lista do novo elemento à sua direita no anel, passando sua porta. Para o backup n+1, passa a porta do primeiro elemento da lista, que vai ser o elemento a sua direita no anel. Sempre que um novo backup se registra, repete os mesmos passos. Desta forma, sempre mantemos a ordem do anel, já que todas as réplicas sabem a réplica a sua direita. **[*register_new_backup()* e *listen_main_server()* no GCServer.cpp]**

Quando uma réplica percebe que a conexão com a primária foi perdida, ela inicia a eleição, caso já não esteja participando **[*start_election()*]**. O election id escolhido foi o número da porta da réplica, dando prioridade a portas maiores. A partir daí fica fácil seguir o algoritmo, a réplica se conecta na porta da réplica seguinte e manda seu election id. Seguimos o restante do algoritmo normalmente, no final, ao receber seu próprio id, a réplica se elege líder e avisa a réplica seguinte. Quando uma réplica recebe a mensagem de *elected*, junto na mensagem vem a porta desse novo líder. A réplica então se conecta e se registra ao líder. E o novo líder ao receber os registros, monta a lista de backups, avisando os backups da réplica as suas direita e assim todo o ciclo se repete. **[*handle_election()* e *register_new_connection()*]**

Quando a réplica primária cai, todas as conexões com os clientes são perdidas, e como são os clientes que se conectam ao servidor, não teria com restabelecer essa conexão. Para resolver isso, modificamos o app: antes dele se conectar pela primeira vez, o app começa esperar novas conexões em uma porta, então ele se conecta ao servidor e passa sua própria porta que é armazenada no servidor junto ao resto de seus dados. Essas informações são replicadas a todos os backups, e quando um novo líder é eleito, ele conhece as portas de todos os apps e consegue reestabelecer conexão com eles. Dessa forma, um novo líder consegue assumir, estabelecer conexão com todos os app e lidar com as mensagens, fazendo com que tudo siga normalmente na visão do app [***reconnect_to_all_apps()***].

O código entre a réplica primária e os backup é o mesmo, possuímos uma flag booleana ***main_replica*** que usamos para executar comando específicos da primária.

Dificuldades

Durante o envio de muitas mensagens em um curto período, pela lentidão do tratamento das mensagens, pode ocorrer de uma mensagem m1 e outra m2 se concatenar dentro do buffer. Para resolver esse problema enviamos cada mensagem com tamanho único e igual ao buffer nessas situações, isso só ocorre quando o server manda as últimas N mensagens para um novo cliente.
Outra dificuldade foi a parte de reconexão do servidor com os app quando um novo líder é eleito. Não sabíamos como resolver isso inicialmente, pois como passamos a porta do servidor quando o app inicia, nao teriamos como passar a porta da nova réplica que assumiu. Como já mencionado, para resolver isso, além de se conectar no server, o app também espera um conexão em um porta, o novo líder pode então reconectar com o app através dela.

Fora isso, não tivemos outra grande dificuldade, acredito que a boa organização do código no T1, possibilitou com que a gente implementasse as novas funcionalidades com facilidade.

Conclusão

O trabalho foi concluído de forma satisfatória, encontramos alguma dificuldades durante o caminho mas que foram resolvidas seguindo os aprendizados adquiridos em aula. No final conseguimos terminar as funcionalidades que faltaram do T1 e ainda implementar as do T2 com cuidado e robustez.