Universidade Federal do Rio Grande do Sul - Instituto de Informática
INF01151 - Sistemas Operacionais II - Prof. Alberto Egon Schaeffer Filho

Bruno Fonte Costa 00308814
Diego Hommerding Amorim 00341793
Gabriel Kenji Ikuta 00337491
Leonardo Gonzatti Heisler 00344125

Data: 06/01/2025

Descrição do Ambiente de Teste

- **Sistema operacional e versão:**
  Ubuntu 22.04 LTS

- **Configuração da Máquina:**
  Processador(es): AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx
  Memória RAM: 6GB

- ***Compiladores Utilizados:***
  g++ 11.4.0

Pontos do Relatório

(A) Funcionamento do Algoritmo de Eleição de Líder

- Explicação do funcionamento

  Foi utilizado o algoritmo do Anel. A partir do qual temos uma lista de servers de backup, e geramos uma conexão encadeada entre eles formando um anel, a conexão formada é unidimensional, onde temos para cada servidor um incomming_socket, utilizado para receber mensagens do servidor imediatamente anterior, e o outgoing_socket, usado para mandar mensagens ao servidor imediatamente posterior. Dessa maneira, quando o servidor principal falha e não manda os pacotes de heartbeat para os seus servidores backup, o primeiro a perceber a falha inicia a eleição através da função start_ring_election(), e para cada servidor backup subsequente, a função handle_election() vai receber a mensagem do servidor anterior na ordem, verificar quais ações deve fazer de acordo com o algoritmo do anel, e mandar uma mensagem para o servidor subsequente de acordo.

  Ao invés de termos mensagens de <election, id> e <elected, id>, definimos tipos de packets diferentes, um Packet::ELECTED_PACKET e um Packet::ELECTION_PACKET, e dessa maneira o payload se resume a somente o id do servidor sendo eleito como líder.

- Justificativa da escolha

Essa escolha foi feita por motivos de simplicidade de implementação. Além disso, entendemos o requisito de teste para o sistema e percebemos que o algoritmo do anel seria suficiente para passar nos testes que o sistema será exposto. Mais precisamente foi verificado que durante a testagem do sistema não teremos falhas nos servidores de backup durante a etapa de eleição, assim, o algoritmo do anel é suficiente para os requisitos.

(B) Implementação da Replicação Passiva

- Descrição da implementação

  Quando um cliente se conecta ao servidor principal, seu nome de usuário e hostname são adicionados à lista de clientes do servidor. Essas informações são propagadas para todos os servidores de backup usando a função start_communications(), garantindo que os backups tenham uma visão consistente dos clientes ativos.
  Qualquer arquivo enviado, baixado ou excluído por um cliente é sincronizado com os servidores de backup em tempo real. Por exemplo:No método upload(), os arquivos recebidos de um cliente são propagados para todos os servidores de backup usando FileTransfer::send_file(). Da mesma forma, o método delete_server_file() garante que comandos de exclusão de arquivos sejam enviados para todos os servidores de backup.
  Quando um novo servidor de backup é adicionado, o servidor principal envia o estado atual para o servidor de backup:

  A função backup_client_list() propaga a lista de clientes ativos para o servidor de backup recém-adicionado.

  A função backup_server_list() garante que o novo servidor de backup tenha conhecimento de todos os outros servidores de backup.

  A função backup_sync_dir() envia todos os arquivos do diretório de sincronização para o novo servidor de backup.

  Dessa maneira mantemos todos os backups atualizados em tempo real.

- Desafios encontrados

Problemas Durante a Implementação

- Problemas encontrados

  Um novo ambiente de desenvolvimento foi utilizado para a etapa 2. Tendo em vista a necessidade de executar a aplicação em diferentes máquinas, utilizamos o Docker para simular a rotina de testes que o sistema será submetido. A adição do Docker gerou facilidades, mas também ocasionou em uma maior dificuldade de realizar os testes de funcionalidades durante o período de desenvolvimento. Enquanto

anteriormente precisávamos apenas rodar a rotina de build, agora surgiu a necessidade de não só rodar a nova rotina de build do docker, que se tornou mais vagarosa pelo overhead de sempre ter que gerar novas imagens, como também pela necessidade de primeiro rodas todos os containers, e só depois todos os agentes do sistema. Isso gera ao longo do tempo um desgaste na equipe de desenvolvimento, pois agora, o tempo para subir o ambiente duplica, e se tornou necessário subir o ambiente a qualquer mínima alteração no código.

- Soluções aplicadas (ou justificativa da ausência de solução)

  Não conseguimos encontrar nenhuma alternativa a esta maneira de desenvolver o sistema, por mais que tornasse a etapa de desenvolvimento mais cansativa e vagarosa, não impediu que conseguíssemos verificar o funcionamento das features com frequência razoável.

Funções Adicionadas

- **serverComManager::backup_sync_dir -** Função utilizada pelo main server para propagar TODOS os arquivos localizados no seu sync dir (todos os arquivos localizados nos subdiretórios dentro de userDirectories) para o socket especificado no parâmetro.
- **serverComManager::backup_server_list -** Função utilizada pelo main server para propagar os *hostnames* de cada servidor da sua lista de servidores backups para o socket especificado no parâmetro.
- **serverComManager::backup_client_list -** Função utilizada pelo main server para propagar o *username* e os *hostnames* dos devices de cada cliente da sua lista de clientes para os socket especificado no parâmetro.
- **serverFileManager::receive_sync_dir_files -** Função utilizada por um servidor backup para receber TODOS os arquivos do diretório de sincronização do servidor principal (socket parâmetro).
- **serverComManager::receive_server_list -** Função utilizada por um servidor backup para receber as informações da lista de servidores do servidor principal.

- **serverComManager::receive_client_list -** Função utilizada por um servidor backup para receber as informações (*username* e *hostnames*) da lista de clientes do servidor principal.
- **serverFileManager::get_sync_dir_files_in_directory -** Função utilizada pelo servidor principal para obter todos os arquivos contidos dentro do diretório de sincronização userDirectories e todos seus subdiretórios.
- **serverComManager::heartbeat_protocol -** Função chamada pelo servidor principal (em uma thread separada) para emitir pacotes '*heartbeat*' a todos os servidores backup de 5 em 5 segundos.
- **serverComManager::add_backup_server -** Função chamada pelo servidor principal para fazer o procedimento de conexão de um novo servidor backup. Ele:
    - Propaga para todo os servidores backup conectados (na lista de servidores) o *hostname* do novo servidor backup que está se conectando;
    - Chama as funções de backup (sync_dir, server_list, client_list).
- **serverComManager::await_sync -** Função chamada pelo servidor backup para aguardar pacotes de sincronização (*download* e *delete* de arquivos, *add* e *remove* de clientes, *add* de servidores backup) e pacotes de *heartbeat*. Utiliza uma thread que roda um timer (**election_timer**) para monitorar a frequência de recebimento de pacotes. Se o último pacote recebido foi recebido a mais de 15 segundos, inicializa eleição.
- **serverComManager::election_timer -** Função que calcula de 15 em 15 segundos a diferença do horário atual e do horário de recebimento do último pacote em segundos.
- **serverComManager::start_election_socket -** inicializa os sockets que serão usados na eleição, outgoing election socket (socket para enviar pacotes com id para o próximo socket do anel) e incoming election socket (recebe o pacote do backup server anterior).

- **serverComManager::bind_incoming_election_socket -** apenas dá o bind no incoming election socket.
- **serverComManager::accept_election_connection() -** faz o listen do incoming election socket, ficando num loop. Essa função está sendo executada numa thread quando o server backup é inicializado (no server.cpp), para ficar dando listen até receber um pacote de eleição.
- **serverComManager::connect_election_sockets(hostent* backup_server) -** conecta com o próximo backup server do anel, usando o outgoing election socket.
- **serverComManager::start_ring_election -** É chamado na função serverComManager::await_sync, após não receber o heartbeat por 15 segundos (mais detalhes em serverComManager::election_timer). Em seguida, conecte os servidores de backup em um anel e envie os pacotes com o ID.
- **serverComManager::handle_election(int socket) -** A função recebe uma mensagem do servidor anterior, e utiliza a lógica da eleição por anel. Dependendo de qual etapa da eleição está o algoritmo, ele envia pacotes de Elected, Election ou retorna vazio sem encaminhar nenhuma mensagem.
- **serverComManager::evolve_into_main() -** Transforma o servidor de backup em servidor principal. Ela remove o próprio hostname da lista de servidores, reconecta-se aos clientes e servidores, e propaga a remoção do servidor para todos os servidores de backup enviando pacotes de exclusão.

Funções Modificadas

- **serverComManager::start_communications -** Adicionado propagação de pacote do tipo CLIENTINFO_PACKET para servidores backup com o *username* e *hostname* do device que está se conectando para sincronização dos client lists.

- **serverComManager::end_communications -** Adicionado propagação de pacote do tipo DELETEDEVICE_PACKET para servidores backup com o *username* e *hostname* do device que está se desconectando para sincronização dos client lists.

- **serverComManager::upload** - Adicionado propagação dos arquivos para os servidores backup listados em server_list.

- **serverComManager::delete_server_file -** Adicionado propagação do pacote de delete para os servidores backup listados em server_list.

- **main_server_connection_handler (server.cpp) -** Adicionado recebimento de pacote ACK informando ao main server se cliente se conectando é um device ou um servidor backup com seus respectivos tratamentos.

- **bind_main_server_socket (server.cpp) -** Função agora recebe parâmetro informando o número de porta.

- **main (server.cpp) -** Número de parâmetros define se servidor inicializado é o main server ou é um servidor backup. Nenhum parâmetro inicializa servidor principal na porta 4000 (./myServer), 1 parâmetro inicializa servidor principal na porta especificada (./myServer <#porta>), 2 parâmetros inicializa servidor backup se conectando ao servidor principal (./myServer <hostname_main_server> <#porta_main_server>) **.** Servidor principal agora cria thread com propagador de heartbeat para os backup servers conectados antes de iniciar o listen para as conexões.

- **clientComManager::connect_sockets -** Cliente agora envia pacote ACK para servidor principal identificar ele como device e não servidor backup. (device => seqn == 1).

- **main (client.cpp) -** Cliente agora cria uma thread adicional para aguardar conexões. Quando o servidor principal morre o servidor backup eleito inicia conexão com cada cliente que estava conectado anteriormente.

Dificuldades

- Estrutura de server_lists estava com valor default de socket 0. Esquecemos de remover o servidor quando ele é eleito da lista daí propagação estava propagando arquivos para o socket 0 e printando seu conteúdo binário no terminal. Extremamente confuso de debugar.

- Variável global estava sendo inicializada longe da função de election_timer, não sendo resetada quando o timer é inicializado novamente após uma reconexão de uma nova eleição.

- Algoritmo de eleição em geral foi complexo de programar corretamente.

- Administrar e sincronizar as estruturas de dados.