Universidade Federal de Santa Catarina – UFSC
Departamento em Informática e Estatística
INE 5418 – Computação Distribuída
Semestre 2025/2

Definição do Trabalho 2: Editor de Texto colaborativo com Consistência Eventual

O compartilhamento de recursos em ambientes distribuídos oferece uma grande variedade de aplicações, desde jogos, difusão de informação ou colaboração. Entretanto, a alta concorrência e a possibilidade de falhas impõem desafios para coordenar ações e garantir consistência de dados.

Neste trabalho, serão abordados mecanismos como CRDTs (*Conflict-Free Replicated Data Types*) e relógios vetoriais para garantir consistência eventual sobre operações em um ambiente distribuído. O objetivo é implementar um *editor de texto colaborativo*, onde um grupo de nós pode modificar o mesmo documento de texto concorrentemente, incluindo inserções e remoções em qualquer posição. O sistema deve utilizar um *CRDT de Sequência* para gerenciar a ordem dos caracteres e garantir *consistência eventual*, utilizando programação com *sockets* para a comunicação.

Conceitos a serem estudados:

Comunicação Ponto-a-Ponto e Sockets: Configuração de uma rede de nós que se comunicam diretamente, com troca de mensagens usando sockets TCP/IP para confiabilidade na entrega das operações.

CRDTs de Sequência (List CRDTs): Implementação de um CRDT especificamente projetado para sequências (listas), como o RGA (*Replicated Growable Array*).

Relógios Vetoriais (Vector Clocks): Uso de algum mecanismo de rastreamento causal para garantir que as operações sejam aplicadas na ordem causal correta, essencial para processar remoções e evitar a duplicação de operações.

Consistência Eventual Forte (Strong Eventual Consistency - SEC): Para a demonstração de que o sistema converge para o mesmo estado e ordem de caracteres em todos os nós.

Configuração da Rede e Protocolo de Comunicação (*Sockets*)

Os nós da rede participam de um grupo que mantém o texto replicado. Cada nó representa uma instância do editor. Utilize um mínimo de 3 nós para demonstrar a consistência do texto no ambiente distribuído.

- Cada nó recebe um ID único de localização (`site ID`);
- Cada nó deve manter conexões TCP ativas com todos os outros nós conhecidos (*full-mesh P2P*);
- Definir um formato de mensagem (*raw data*, *protocol buffer*, JSON ou similar) que encapsule uma *Operação CRDT* (`Op-based CRDT`);

- Sugestão para os campos da Mensagem:
  `{"type": "insert"|"delete", "op_id": <VectorClock>, "site_id": <ID_do_No>, "pos_id": <ID_do_Caractere_Anterior_ou_Alvo>, "char": <Caractere>}`

Implementação do CRDT de Sequência (Sugestão: RGA Simplificado)

O RGA (*Replicated Growable Array*) é um bom ponto de partida, pois lida com a ordem relativa de caracteres através de identificadores únicos. Caso o grupo queira explorar tipos mais sofisticados de CRDT, adequados ao problema, é permitido.

O texto em si será representado como um vetor de caracteres. Cada caractere inserido recebe um identificador único e totalmente ordenado (`Position ID`). Um `Position ID` pode ser um par `(timestamp_origem, site_id)`. Caso o grupo opte por outro tipo de CRDT, o `Position ID` pode conter outra informação, como uma sequência hierárquica, no caso de LSEQ (*Linear Sequence*).

Sugestão, por simplicidade, adote `Position ID = (VectorClock_insercao_local, site_id)`. Se a inserção ocorrer entre caracteres `A` e `B`, o novo caractere recebe um `ID` logicamente maior que `A` e menor que `B`. Se houver concorrência no mesmo ponto, o `site_id` atua como desempate final.

A réplica local é uma lista/vetor de objetos `Caractere`, onde cada `Caractere` armazena:
- `valor`: O caractere em si (p.ex., 'A', 'b', ' ');
- `id`: O `Position ID` único;
- `deleted`: Um valor booleano para marcar remoções.

As operações sobre caracteres são:

- `insert(caractere, posição)`:
  Gera um novo `Position ID` para o caractere, garantindo que ele caia na ordem correta. Então, aplica a inserção localmente; cria uma mensagem de `Operação` com o novo `Position ID` e o `caractere`; e envia a mensagem para todos os nós da rede.

- `delete(posição)`:
  Localiza o caractere a ser removido; altera o seu estado para `deleted=true`; cria uma mensagem de `Operação` de remoção que referencia o `Position ID` do caractere alvo; e envia a mensagem para todos os nós da rede.

- `merge(mensagem_op)`:
  Ao receber uma operação de outro nó (via *socket*), o nó local aplica a mudança:
  Inserção Remota: Insere o novo caractere na posição correta na lista, baseando-se em seu `Position ID` (que garantirá a ordem);
  Remoção Remota: Localiza o caractere pelo `Position ID` e o marca como `deleted=true`.
  A idempotência e comutatividade do `merge` (princípio do CRDT) devem ser garantidas ao aplicar a lógica.

Testes e Execução

Para a execução, o programa deve permitir visualizar a tela de cada nó. Deve ser possível observar o texto do documento incluindo os caracteres não removidos. Pode ser útil apresentar também um registro (*log*) de operações recebidas.

Para testar cenários com concorrência, pode-se definir cenários de interesse. Por exemplo:

- *Concorrência de Inserção*: Nó `ID1` insere 'X' e Nó `ID2` insere 'Y' na mesma posição (ex: no início). O CRDT deve garantir que 'X' e 'Y' apareçam na mesma ordem em ambos os nós (e.g., 'X Y' ou 'Y X', mas não 'X Y' em `ID1` e 'Y X' em `ID2`).

- *Concorrência Inserção/Remoção*: Nó `ID1` insere um caractere 'Z'. Concorrentemente, Nó `ID2` deleta o caractere anterior a 'Z'. Demonstração de que a operação é resolvida de forma consistente (o 'Z' inserido deve permanecer ou ser deletado em todos os nós).

Entrega

O trabalho consiste em:

1. Implementar o editor de texto compartilhado. A linguagem de programação é de livre escolha para os grupos.
2. Breve relatório que indique as principais decisões e estratégias de implementação utilizadas, instruções sobre como compilar e executar o código produzido. Ao final, discuta quais conclusões você observa, indicando limitações observadas na implementação atual.

O trabalho pode ser realizado em **grupos de até 3 participantes**. O trabalho será apresentado em laboratório.

O código-fonte e relatório devem ser enviados pelo Moodle para análise e avaliação.

Os nomes dos participantes do grupo devem constar no relatório entregue no Moodle. Participantes com nomes não referenciados não serão considerados como membros do grupo.

Referência úteis:

[1] Preguiça, Nuno. "Conflict-free replicated data types: An overview." *arXiv preprint arXiv:1806.10254* (2018).

[2] Kleppmann, Martin. "CRDTs: The Hard Parts" Disponível em: https://www.youtube.com/watch?v=x7drE24geUw