Universidade Federal do Espírito Santo

CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA

Sistemas Distribuídos – Turmas 01 e 02 – 2022/1
Prof. Rodolfo da Silva Villaça – rodolfo.villaca@ufes.br
Monitor: Eduardo M. Moraes Sarmento – eduardo.sarmento@ufes.br
Laboratório VI – Eleição e Coordenação Distribuída

1. Objetivos

- Utilizar a implementação de sistemas de comunicação indireta por meio de *middleware Publish/Subscribe* (Pub/Sub) com Filas de Mensagens;
- Realizar uma eleição de coordenador em sistemas distribuídos por meio da troca de mensagens entre os participantes do sistema;
- Realizar votação sobre o estado de transações distribuídas por meio da troca de mensagens entre os participantes do sistema.

2. Especificação do Sistema

Você precisará construir um protótipo similar a resolvedor de provas de trabalho em um minerador de criptomoedas, com implementação baseada em comunicação indireta usando o modelo *Publish/Subscriber* e *Filas de Mensagens*. A implementação do minerador deverá ser realizada em Python com a biblioteca Paho[1] MQTT e o *broker* utilizado deverá ser o EMQX. Todo participante do sistema deverá implementar, simultaneamente, o papel do minerador e do controlador, conforme especificado no Laboratório V. Na inicialização do sistema, todos os participantes do (processos) deverão realizar uma eleição para definição do controlador (1) e dos mineradores (2).

O papel de controlador terá o funcionamento igual do descrito no Laboratório V, conforme descrição a seguir (relembrando):

a) Manter, enquanto estiver em execução, uma tabela com os seguintes registros:

TransactionID	*Challenge*	*Solution*	*Winner (ClientID)*
int	int	str	int

- *TransactionID*: Identificador da transação, representada por um valor inteiro;
- *Challenge*: Valor do desafio criptográfico associado à transação, representado por um número [1..6], onde 1 é o desafio mais fácil. Gere desafios aleatórios ou sequenciais (experimente as diferentes abordagens);
- *Solution*: *String* que, se aplicada a função de *hashing* SHA-1, solucionará o desafio criptográfico proposto;
- *Winner*: *ClientID* do usuário que solucionou o desafio criptográfico para a referida *TransactionID* (mesma linha da tabela). Enquanto o desafio não foi solucionado, considere que o *ClientID* = -1 (indicador de que o desafio ainda está pendente);

b) Ao carregar, deverá gerar um novo desafio com *TransactionID* = 0;
c) Assinar a seguinte fila de mensagens:

1 https://www.eclipse.org/paho/

Nome da Fila	*Codificação da Mensagem*	*Significado*
sd/solution	*ClientID <int>*, *TransactionID <int>*, *Solution <string>*.	*Solution* usada pelo *ClientID* para resolver o desafio associado à *TransactionID.* Importante: sempre enviar/ receber em formato *string* codificada em JSON[2].

d) Sempre que receber uma proposta de solução, o controlador deverá:
d.1) Verificar se a *TransactionID* está pendente e se a solução atende aos requisitos do desafio proposto;
d.2) Caso atenda, atualiza a tabela de estado das transações e publica o resultado na fila *sd/result;*
d.3) Caso não atenda, publica o resultado na fila *sd/result* informando que a solução foi rejeitada.

e) O controlador deve ficar em *loop*, só podendo ser interrompido com um <ctrl+c> ou similar, e prover um menu com as seguintes opções:

Item do Menu	Ação
newChallenge	Publica na fila *sd/challenge* um novo desafio (*challenge*), imprime a tabela atualizada e bloqueia até que o desafio seja resolvido por algum minerador. Sempre que houverem novas submissões de solução, mostrar a submissão e o resultado da avaliação (aprovado/ reprovado). Importante: sempre enviar/ receber em formato *string* codificada em JSON!
exitController	Sai do controlador e pára de assinar a fila *sd/result*.

O papel de minerador deverá ter o funcionamento igual ao descrito no Laboratório V, relembrando:

a) Manter, enquanto estiver em execução, uma tabela com os seguintes registros:

TransactionID	*Challenge*	*Solution*	*Winner (ClientID)*
int	int	str	int

- *TransactionID*: Identificador da transação, representada por um valor inteiro;
- *Challenge*: Valor do desafio criptográfico associado à transação, representado por um número [1..6], onde 1 é o desafio mais fácil. Gere desafios aleatórios ou sequenciais (experimente as diferentes abordagens);
- *Solution*: *String* que, se aplicada a função de *hashing* SHA-1, solucionará o desafio criptográfico proposto;
- *Winner*: *ClientID* do usuário que solucionou o desafio criptográfico para a referida *TransactionID* (mesma linha da tabela). Enquanto o desafio não foi solucionado, considere que o *ClientID* = -1;

b) Ao iniciar, essa tabela deverá estar vazia e o minerador deverá bloquear até receber uma mensagem na fila *sd/challenge*. Ao receber essa mensagem, deverá imprimir em tela a recepção da mensagem, com o respectivo *TransactionId* e o desafio correspondente;

c) Assinar a seguinte fila de mensagens no *broker* Pub/Sub:

2 https://docs.python.org/pt-br/3/library/json.html

Nome da Fila	*Codificação da Mensagem*	*Significado*
sd/challenge	*TransactionID <int>*, *Challenge <int>*.	Publicado pelo controlador! Contém o novo *TransactionID* e o *Challenge* associado a ele <*int*>[1..6]. Importante: sempre enviar/ receber em formato *string* codificada em JSON!
sd/result	*ClientID <int>*, *TransactionID <int>*, *Solution <string>*, *Result <int>*.	Publicado pelo controlador! Contém a resposta do controlador a respeito da submissão de uma solução para o respectivo *TransactionID*. *Result* = 0 caso a solução seja inválida, e *Result* != 0 caso a solução seja válida. Importante: sempre enviar/ receber em formato *string* codificada em JSON!

d) Cada grupo deverá usar pelo menos 3 (três) processos durante os testes do sistema, sendo um controlador eleito, e dois mineradores.

3. Eleição Distribuída

A Figura 1 mostra os 3 (três) estados iniciais do sistema, que deverá ser implementado em cada participante.

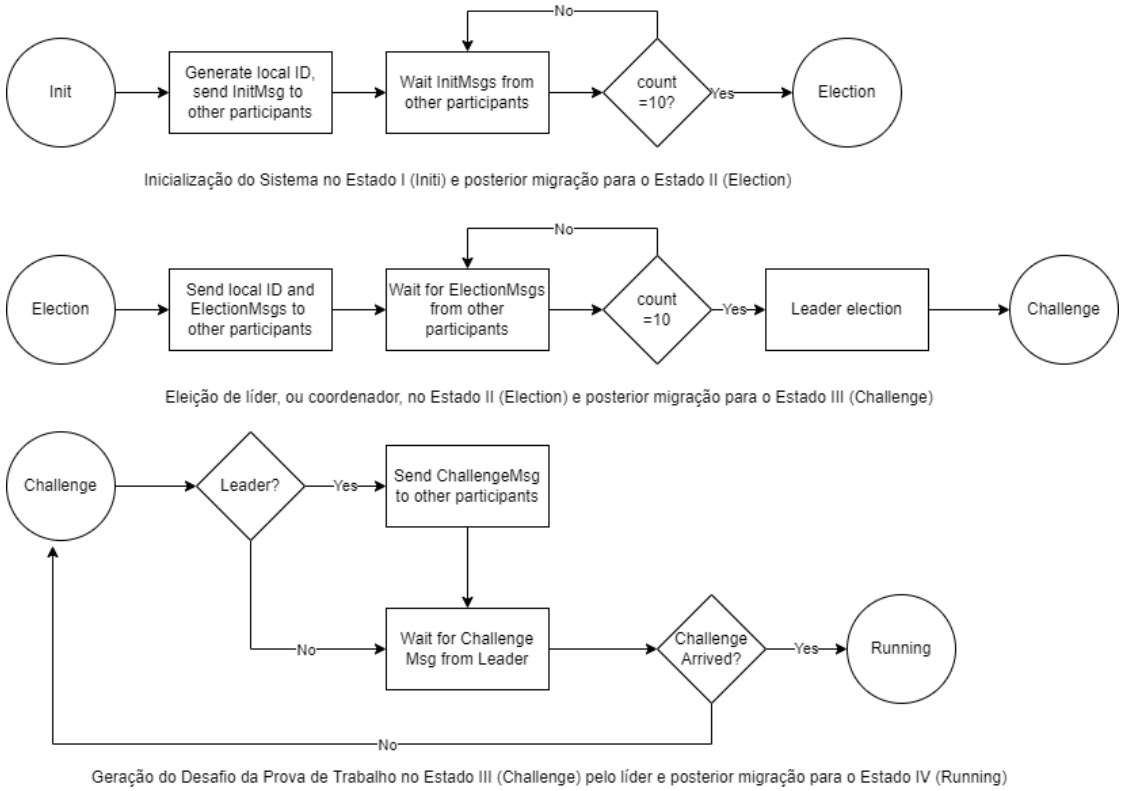

Figura 1: Estados de Inicialização (Init), Eleição (Election) e definição do Desafio (Challenge) de hashing criptográfico.

De acordo com a Figura 1, o sistema deverá inicializar sempre no estado *Init*, onde cada participante terá o seu identificador local (*ClientID*). Esse valor deverá ser gerado aleatoriamente num intervalo de 16 *bits* [0..65335]. Após gerar o seu *ClientID*, o nó participante deverá enviá-lo para os demais por meio da publicação de uma mensagem de registro *InitMsg* na fila sd/*init* no *broker* de comunicação. A mensagem InitMsg tem a formatação a seguir:

Nome da Fila	*Codificação da Mensagem*	*Significado*
sd/init	*ClientID <int>*	Contém o *ClientID* de 16 *bits* gerado aleatoriamente pelo processo participante do sistema no estado de inicialização (*Init*). Importante: sempre enviar/ receber em formato *string* codificada em JSON!

Ainda de acordo com a Figura 1, após o envio da sua própria mensagem de inicialização, o processo deverá aguardar a chegada de um total de *n* (*n*=10 na figura) mensagens de inicialização distintas, provenientes dos demais participantes – importante, n deverá ser passado como parâmetro de inicialização do processo. Ao receber um total de *n-1* mensagens do tipo *InitMsg* na fila *sd/init*, encerra-se a fase de inicialização. O processo poderá reenviar sua própria mensagem de inicialização enquanto as *n* demais mensagens não forem recebidas – esse reenvio é de definição livre.

Após concluir a fase de inicialização (*Init*), tem-se inicio a fase de eleição do coordenador (ou líder) do sistema (*Election*). Nesta fase cada processo deverá gerar um número aleatório de 16 *bits* (*VoteID*) e enviar ao grupo por meio de uma mensagem de eleição (*ElectionMsg*) na fila *sd/election*, contendo seu *ClientID* e seu "voto" (número aleatório) na mensagem de eleição. Após enviar a *ElectionMsg* com seu *VoteID*, cada nó deverá aguardar o recebimento dos votos dos demais participantes do sistema (*count*=10 na Figura 1). Após receber todos os votos, deve-se escolher o participante com maior *VoteID* como líder do grupo e encerrar essa fase. Use uma combinação de *VoteID* + *ClientID* para decidir sobre eventuais desempates entre os participantes (maior *ClientID* terá prioridade em caso de empate). Assuma que todos os participantes são honestos e gerarão números realmente aleatórios e aceitarão o resultado da eleição.

Nome da Fila	*Codificação da Mensagem*	*Significado*
sd/voting	*ClientID <int>; VoteID <int>*	Contém o *ClientID* de 16 *bits* gerado aleatoriamente pelo processo participante do sistema no estado de inicialização (*Init*). Contém, também, o voto *VoteID*, de 16 bits, gerado aleatoriamente. Importante: sempre enviar/ receber em formato *string* codificada em JSON!

Considerando que não há perda de mensagens no sistema, todos os participantes estarão sincronizados após a eleição e assumirão o estado de desafio (*Challenge*). O líder eleito pode assumir o papel de controlador e, então, definir um desafio e enviar para os demais participantes do sistema – conforme as especificações do Laboratório V.

Como requisitos e premissas, em seu projeto assuma que:
a) O número de participantes é fixo e conhecido;
b) Não haverá entrada/saída dinâmica de nós no sistema (*churn*);
c) O *broker* Pub/Sub é único, infalível, e de conhecimento prévio por todos os participantes;
d) Os nós participantes do sistema são honestos, não havendo comportamento malicioso para atrapalhar o funcionamento do sistema;

4. Instruções Gerais

- O trabalho pode ser feito em grupos de 2 ou 3 alunos. Não serão aceitos trabalhos individuais ou em grupos de mais de 3 alunos. Se for necessário, o professor reserva-se no direito de ter que subdividir grupos já existentes;
- Data de Entrega: 04/06/2023, por meio da Sala de Aula Virtual da disciplina no *Google Classroom*, na atividade correspondente ao Laboratório VI. 1 (uma) submissão por grupo é suficiente;
- Deve-se submeter apenas o *link* para o repositório virtual da atividade (*Github*, *Bitbucket*, *Google Colaboratory* ou similares) contendo: i) códigos-fonte; ii) instruções para compilação e execução; iii) relatório técnico (.pdf ou README); e iv) vídeo curto (máx 3 min) mostrando uma execução, resultado e análise dos resultados encontrados;
- O relatório técnico deverá conter: a descrição da implementação, testes e resultados encontrados;

Bom trabalho!