Universidade Federal
do Espírito Santo

CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA

Processamento Paralelo e Distribuído – Turmas 01 e 02 (EARTE) – 2022/1

Prof. Rodolfo da Silva Villaça – rodolfo.villaca@ufes.br

Laboratório VI – Comunicação Indireta, Eleição, Coordenação Distribuída, Segurança e Tolerância a Falhas (Trabalho Final) – Alteração 26/07

1. Objetivos

- Experimentar a implementação de sistemas de comunicação indireta por meio de *middleware Publish/Subscribe* (Pub/Sub);
- Realizar 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.
- Experimentar a verificação de assinaturas de mensagens por meio de chaves públicas/ privadas;
- Tratar algumas situações de falhas ou inconsistências em Sistemas Distribuídos.

2. Funcionamento do Sistema

Você precisará construir um protótipo similar a um minerador de criptomoedas do tipo *bitcoin*, que consiste na solução de uma prova de trabalho baseada em *hashing* usando modelo de comunicação indireta Pub/Sub. Neste laboratório você deverá usar o **RabbitMQ** como *broker* de troca de mensagens. A arquitetura seguirá o modelo descentralizado, ou seja, sem a existência de um servidor centralizado para coordenar as ações do sistema.

Todos os nós deverão manter, localmente, enquanto estiver em execução, uma tabela constantemente atualizada com os seguintes registros:

TransactionID	***Challenge***	***Seed***	***Winner (Node ID)***
int	int	str	int

- *TransactionID*: Identificador da transação, representada por um valor inteiro positivo de 32 *bits*. Use valores incrementais começando em 0 (zero);
- *Challenge*: Valor do desafio criptográfico associado à transação, representado por um número no intervalo [1..10], onde 1 é o desafio mais fácil. O desafio está explicitamente definido no Anexo I deste documento;
- *Seed:* Sequência de caracteres que, se aplicada a função de *hashing* SHA-1, solucionará o desafio criptográfico proposto. Tamanho variável;
- *Winner*: Identificador do participante do sistema que solucionou o desafio criptográfico para a referida *TransactionID*. Considere que o identificador do usuário (*NodeID*) é representado por um valor inteiro positivo de 32 *bits*. Enquanto o desafio proposto não foi solucionado, considere que esse identificador é igual a -1;

A tabela deverá ser impressa na tela sempre que houver alguma atualização no estado do sistema. Na inicialização do sistema, todos os nós participantes deverão limpar essa tabela e iniciar com *TransactionID*=0. A Figura 1 mostra o estado inicial do sistema (*Init*) e suas interações principais até a migração para o estado seguinte (*PubKeys*).

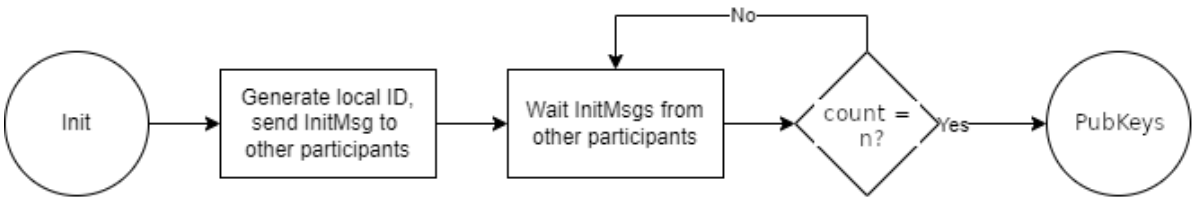

Figura 1: Estado de Inicialização (Init) e suas interações até a publicação e compartilhamento das chaves públicas.

De acordo com a Figura 1, o sistema deverá inicializar sempre no estado *Init*, onde cada participante terá o seu identificador próprio na rede (*NodeID, inteiro positivo, 32 bits)*. Esse valor deverá ser gerado aleatoriamente. Após gerar o seu *NodeID*, o nó participante deverá enviar seu *NodeID* para os demais por meio da publicação de uma mensagem texto (*InitMsg*) no *broker* de comunicação. Considere a existência de um *broker* único e todos os participantes (nós) deverão publicar e assinar a fila “ppd/init” neste *broker*.

Importante:

- A mensagem *InitMsg* não deverá ser assinada, e seu conteúdo deve, obrigatoriamente, ser formatado como um arquivo JSON contendo somente o *NodeID* em formato texto;
- Faça com que o número de participantes no sistema (*count*) seja um valor configurável (*n*), que pode ser alterado como um parâmetro de configuração do nó durante sua inicialização;
- Desconsidere falhas de indisponibilidade ou desconexão por parte dos participantes do sistema.

Ainda de acordo com a Figura 1, após o envio da sua própria mensagem de inicialização, o nó deverá aguardar a chegada de um total de *n* mensagens de inicialização distintas, provenientes dos demais participantes. Ao receber mensagens de todos os participantes, encerra-se a fase de inicialização (*Init*).

Importante:

- O nó deverá reenviar continuamente sua própria mensagem de inicialização com seu *NodeID* enquanto as *n* demais mensagens não forem recebidas;
- Trate as situações onde haverá a recepção de mensagens duplicadas, provenientes do mesmo nó!

Após concluir a fase de inicialização (*Init*), tem-se inicio a fase de troca das chaves públicas entre os participantes do sistema (*PubKeys*). Todos os nós participantes deverão possuir um par de chaves pública e privada, de criptografia assimétrica RSA e comprimento 1024 bits (consulte o Laboratório V).

A chave pública deverá ser trocada entre todos os participantes por meio de uma mensagem *PubKeyMsg*, na fila "ppd/pubkey" do *broker*. A mensagem deverá estar obrigatoriamente no formato JSON e conter exatamente: o *NodeID* e a chave pública (*PubKey*).

Importante:

- A chave privada (*PrivKey*) de cada participante é de uso local e não pode ser compartilhada entre os demais participantes;
- Todos os nós deverão aguardar o recebimento das chaves públicas de todos os participantes, e armazenar localmente essas chaves (pode ser em memória ou disco);
- Para evitar que o sistema trave por inanição, reenvie sua própria chave regularmente até que esta fase do sistema seja concluída. Use algum temporizador para isso!

Após concluir a fase *PubKeys*, quando todas as *n* chaves públicas dos participantes do sistema estarão compartilhadas entre eles, segue-se para a fase de eleição do coordenador (ou líder) do grupo (*Election*). Nesta fase casa nó deverá gerar um número aleatório de 32 *bits*, inteiro positivo (*ElectionNumber*) e enviar ao grupo por meio de uma mensagem de eleição (*ElectionMsg*) na fila "ppd/election".

A *ElectionMsg* de um nó deverá conter o seu próprio *NodeID* e seu "voto" na eleição (*ElectionNumber*), além de estar assinada digitalmente pelo remetente (*Sign*). Após enviar seu "voto", cada nó deverá aguardar o recebimento dos *ElectionNumbers* de todos os *n* participantes do sistema. Após receber todos os votos, deve-se verificar as assinaturas dos remetentes, escolher o participante com maior *ElectionNumber* como líder do grupo e encerrar essa fase.

Importante:

- Use o maior *NodeID* para decidir sobre eventuais desempates entre os participantes. É necessário tratar a possibilidade de empates na eleição de coordenador;
- Considere que pode haver uma tentativa de fraude na eleição, com assinatura errada da mensagem. Neste caso, descarte a *ElectionMsg* fraudada e imprima um *log* na sua tela local e não envie mensagem avisando sobre a tentativa de fraude, apenas a ignore;
- Para evitar que o sistema trave por inanição, reenvie sua própria *ElectionMsg*, com o mesmo valor, regularmente, até que esta fase do sistema seja concluída. Use algum temporizador local para controlar esse envio;
- Imprima na tela o resultado da eleição, atuando como *log* local;

Considerando que todos os participantes estão sincronizados e concordaram com o resultado da eleição, muda-se para o estado de desafio (*Challenge*). O líder pode definir um desafio e enviar por meio de mensagem (*ChallengeMsg*) para todos demais participantes na fila "ppd/challenge". Os demais participantes deverão aguardar o recebimento desta mensagem. A *ChallengeMsg* deverá estar obrigatoriamente no formato JSON e conter: o *NodeID* do coordenador da vez (líder), o desafio (*Challenge*, número inteiro positivo no intervalo [1..10]), e a assinatura do remetente (*Sign*).

Ao receber a mensagem, e certificar-se de que ela foi enviada pelo líder por meio da sua assinatura digital, procede-se a execução do processamento para encontrar uma solução para o desafio, migrando para o estado *Running*.

No estado *Running* cada participante pode usar qualquer estratégia local para resolver o desafio, seja criar múltiplos processos, seja criar zumbis remotos por RPC que farão o processamento do *hashing* criptográfico correspondente ao desafio. Se um nó local encontrar uma solução (*Seed*) deverá enviar sua proposta para os demais participantes por meio da mensagem *SolutionMsg na fila "ppd/solution"* e permanecer no estado *Voting*. A *SolutionMsg* deverá estar obrigatoriamente no formato JSON e conter: *NodeID* do remetente, a solução do desafio (*Seed*) e assinatura da mensagem (*Sign*). Observe que mesmo estando no estado *Running*, cada nó deverá, em paralelo, aguardar mensagens de outros participantes que correspondam à solução do desafio (*SolutionMsg*).

Importante:
1. Ao receber uma mensagem de solução proveniente de algum outro participante, cada nó deverá verificar a autenticidade da origem da *SolutionMsg*;
2. Se a origem é autêntica, verificar a solução proposta. Não é permitido verificar a solução se a origem não for autêntica;

Após verificar a solução, cada nó deverá votar pela aprovação ou reprovação da solução por meio da *VotingMsg*.A VotingMsg deverá estar obrigatoriamente no formato JSON e conter o NodeID da origem da mensagem, o voto (*Vote*, com valores *True*, em caso de aprovação, ou *False*, em caso de reprovação da solução) e a assinatura da mensagem. Neste estado (*Voting*) deve-se aguardar o voto dos demais participantes.

A solução é aprovada ou reprovada por maioria simples dos votos. Se aprovada, encerra-se o estado *Running* e atualiza-se a tabela de controle no estado *Update.* Antes de iniciar um novo desafio, proceda uma nova eleição de líder no estado *Election.* Uma vez eleito, o novo líder poderá definir um novo desafio retornando o sistema ao estado *Challenge*.

Como premissas, em seu projeto assuma que:
a) Não haverá entrada/saída dinâmica de nós no sistema (*churn*);
b) O *broker* Pub/Sub é único, infalível, e de conhecimento prévio por todos os participantes;
c) Os nós participantes do sistema podem não ser bem comportados, havendo possibilidade comportamento malicioso para atrapalhar o funcionamento do sistema – falhas bizantinas. Dica: mantenha estados bem definidos, e ignore mensagens fora de seu contexto (estado).

4. Instruções Gerais

1. 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;
2. Os grupos deverão implementar os trabalhos usando Python;
3. Data de Entrega: 14/08/2022, 23:59 h, improrrogável;
4. A submissão deverá ser feita por meio de um *link* com a disponibilização dos códigos no Github, em modo de acesso público ou, minimamente, que meu e-mail rodolfo.villaca@ufes.br tenha direito de acesso ao código;
5. A documentação deverá ser feita na própria página do Github através do arquivo README;

6. O grupo deverá gravar um vídeo de no máximo 5 min apresentando o funcionamento e procedimento de testes do trabalho.

Bom trabalho!

ANEXO I – Desafio Criptográfico

A "Prova do Trabalho" (*Proof-of-Work* na versão em inglês) ou PoW, é o algoritmo de consenso original em uma rede *blockchain*. Na *blockchain*, este algoritmo é usado para confirmar transações e produzir novos blocos para a cadeia. Com a PoW, os mineradores competem entre si para completar as transações e serem recompensados.

Um sistema de registro de transações descentralizado reúne todas as transações nos blocos. No entanto, deve-se ter cuidado para confirmar as transações e organizar os blocos. Esta responsabilidade é produzida por computadores especiais chamadas mineradores. Esse processo é chamado mineração. O princípio mais importante da prova de trabalho é elaborar um desafio matemático complicado com a possibilidade de se provar a solução rapidamente. Um desafio comum é baseado em *hashing*, ou seja, como encontrar a entrada da função de *hashing* conhecendo a saída desejada?

Por exemplo, considere $h = hash(k)$, onde *hash* é uma função de *hashing* criptográfico, k é uma chave usada com entrada da função, e h é o resultado da aplicação da função com a entrada k. O desafio proposto é encontrar o valor de k que produza um valor conhecido para h.

Suponha que a função *hash* é o SHA-1[1]. Sabendo-se que a aplicação da função SHA-1 em uma entrada k gera um valor h de 160 bits, h varia no intervalo $[0..2^{160}-1]$.

Neste caso, suponha que o desafio é encontrar o valor de k que tenha como resultado um valor de h que possua exatamente seus 5 *bits* mais significativos iguais a 0, e os demais não importam (*). Ou seja, h é qualquer número que satisfaça a seguinte condição:

h em binário (160 *bits* no total)									
bit 159	bit 158	bit 157	bit 156	bit 155	bit 154	…	bit 2	bit 1	bit 0
0	0	0	0	0	*	*	*	*	*

Neste exemplo existem 2^{155} possíveis valores de h que atendem a condição exposta na tabela acima, e o desafio tem valor 5, por corresponder a 5 *bits* previamente conhecidos para o resultado da função *hashing*.

Um desafio com valor 6 deve satisfazer a seguinte condição:

h em binário (160 *bits* no total)									
bit 159	bit 158	bit 157	bit 156	bit 155	bit 154	…	bit 2	bit 1	bit 0
0	0	0	0	0	0	*	*	*	*

Neste exemplo existem 2^{154} possíveis valores de h que atendem a condição exposta na tabela acima. Quanto maior o valor do desafio, menor o espaço de soluções possíveis e mais difícil se torna achar

1 https://pt.wikipedia.org/wiki/SHA-1

a solução para esse desafio. Lembrando que a solução corresponde a encontrar uma entrada *k* que, aplicada à função *hash*, produzirá um valor *h* que satisfaça à condição imposta pelo desafio.

A medida em que o sistema cresce em número de participantes, as dificuldades dos desafios aumentam cada vez mais e os mineradores precisarão de mais e mais poder computacional para resolver o desafio. Neste laboratório o desafio sempre será formado por encontrar o valor de *k* que produza valores com *d* bits mais significativos iguais a 0, e *d* é o tamanho do desafio.