Universidade Federal do Espírito Santo

CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA

Sistemas Distribuídos – Turmas 01 e 02 – 2023/1
Prof. Rodolfo da Silva Villaça – rodolfo.villaca@ufes.br
Monitor: Eduardo M. Moraes Sarmento – eduardo.sarmento@ufes.br
Laboratório V – Comunicação Indireta (*Publish/Subscribe*) – Parte II

1. Objetivos

- Experimentar a implementação de sistemas de comunicação indireta por meio de *middleware Publish/Subscribe* (Pub/Sub) e Filas de Mensagens (*Message Queues*);
- Sincronizar a troca de mensagens entre os componentes do sistema;
- Utilizar o *broker* EMQX MQTT para gerenciamento da fila de mensagens na implementação de sistemas distribuídos;
- Usar o modelo de comunicação indireta Pub/Sub para implementação de um protótipo de um resolvedor de provas de trabalho e minerador de *bitcoins*.

2. Conceitos Básicos

Publish/Subscribe é um padrão arquitetural onde existem os *Publishers* (Publicadores) que enviam as mensagens e os *Subscribers* (Assinantes) que recebem as mensagens. Em uma maneira mais prática, sempre que houver algum evento, o publicador vai enviar uma mensagem para que os assinantes sejam notificados. O padrão arquitetural *Publish/Subscribe* (*Pub/Sub*) se resume na imagem a seguir: existe um publicador, que vai enviar uma mensagem em um canal (*Channel*) que redistribui uma cópia da mensagem para cada assinante. Em nosso caso, o canal é representado por um conjunto de filas gerenciados pelo *broker*.

Nenhuma das partes se conhece, muito menos sabem os detalhes de suas implementações. Apenas conhecem o endereço do *broker*. O modelo de comunicação *Pub/Sub* pode trazer baixo acoplamento e facilidade de escalabilidade em um ecossistema de software. Por ser assíncrono não haverá problemas com *timeouts* ou quebras de processamento por erros encadeados.

EMQX[1] é um *broker* de mensagens com versão livre e de código aberto que implementa o protocolo MQTT (*Message Queue Telemetry Transport*). O protocolo MQTT fornece um método leve e simples para implementação do modelo de comunicação indireta usando *Pub/Sub*.

1 https://www.emqx.com/en/try?product=broker

Atividade 1: Mineração de Criptomoedas usando gRPC

Com base nestes exemplos, 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[2] MQTT e o *broker* utilizado deverá ser o EMQX. Dois processos deverão ser implementados: minerador e controlador.

O processo controlador deverá ter o seguinte funcionamento:

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 no *broker* Pub/Sub:

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[3].

d) Sempre que receber uma proposta de solução, o controlador deverá:
d.1) Verificar se a *TransactionID* ainda 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.

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

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 processo minerador deverá ter o seguinte funcionamento:

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:

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 2 (dois) mineradores. **Ponto extra** se usar threads (ou processos) concorrentes tentando resolver o desafio nos mineradores.

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: 23/05/2023, por meio da Sala de Aula Virtual da disciplina no *Google Classroom*, na atividade correspondente ao Laboratório V. 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;
- Os trabalhos dos grupos da pós-graduação deverão ser interoperáveis.

Bom trabalho!