Universidade Federal do Espírito Santo

CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA

Processamento Paralelo e Distribuído – Turmas 01 e 02 (EARTE) – 2021/1
Prof. Rodolfo da Silva Villaça – rodolfo.villaca@ufes.br
Trabalho T2 – Uso de Middleware Publish/Subscribe e Filas de Mensagens (Parte 2)

Objetivo:
Experimentar a implementação de sistemas de comunicação indireta por meio de middleware Publish/Subscribe com Filas de Mensagens. Sincronizar a troca de mensagens entre os componentes do sistema. Utilizar brokers de troca de mensagens na implementação de sistemas distribuídos.

Definições Gerais:
1. O trabalho deverá possuir broker local para troca de mensagens. Sugere-se usar ou um broker baseado no protocolo MQTT (Message Queuing Telemetry Transport) ou AMQP (Advanced Message Queuing Protocol). No sistema operacional Linux recomenda-se os servidores Mosquitto[1] ou RabbitMQ[2];
2. O serviço a ser implementado é uma Tabela Hash Distribuída (DHT) com espaço de endereçamento com tamanho pelo menos n=32 bits) e um número indefinido de nós virtuais (entrada e saída dinâmica de nós). Lembrando que com n=32 bits de espaço de endereçamento, as chaves de hash (k) pertencem ao intervalo $k = [0..2^n-1]$ organizadas em uma topologia circular, crescente no sentido horário;
3. Os nós da DHT podem ser processos independentes, VMs, containers ou computadores. O sistema poderá ignorar a possibilidade de falhas nos nós da DHT: assuma que os nós sempre avisarão previamente ao entrar e sair da rede;
4. Cada um dos nós deverá sortear aleatoriamente um identificador de 32 bits ($NodeID_i$) no espaço de endereçamento. O NodeID serve para posicionar virtualmente o nó no espaço de endereçamento virtual da DHT. Como haverão i nós, teremos o espaço de endereçamento dividido em i intervalos:
($NodeID_0$..$NodeID_1$], ($NodeID_1$..$NodeID_2$], ($NodeID_2$..$NodeID_3$], ($NodeID_3$..$NodeID_4$], ($NodeID_4$..$NodeID_5$], ($NodeID_5$..$NodeID_6$], ($NodeID_6$..$NodeID_7$], …, ($NodeID_{i-1}$..$NodeID_0$];
5. Cada nó ficará responsável por armazenar o valor v associado à chave k pertencente ao intervalo de sua responsabilidade. Lembrando que k é a chave de indexação da tabela hash e v é o valor armazenado na tabela, no formato de armazenamento chave-valor <k, v>. Considere que v é uma string (caso queira, o grupo poderá impor limite de caracteres a essa string);
6. Os intervalos sob responsabilidade de cada nó da DHT obedecerão o seguinte relacionamento:

1 https://mosquitto.org/
2 https://www.rabbitmq.com/

- Nó 0 –> (NodeID$_{i-1}$..NodeID$_0$]; // aberto-fechado
- Nó 1 –> (NodeID$_0$..NodeID$_1$];
- Nó 2 –> (NodeID$_1$..NodeID$_2$];
- Nó 3 –> (NodeID$_2$..NodeID$_3$];
- Nó 4 –> (NodeID$_3$..NodeID$_4$];
- Nó 5 –> (NodeID$_4$..NodeID$_5$];
- Nó 6 –> (NodeID$_5$..NodeID$_6$];
- ...
- Nó i-1 –> (NodeID$_{i-2}$..NodeID$_{i-1}$];

7. Em resumo, se uma chave k qualquer, associada ao valor v, estiver contida no intervalo (NodeID$_a$..NodeID$_b$] então v deverá ser armazenado na tabela hash contida no Nó b;

Funcionamento:
1. Deverá ser implementada pelo menos 1 mensagem para o ingresso e 1 mensagem para a saída da DHT: join(NodeID) e leave(NodeID).
2. Qualquer nó poderá inicializar a DHT após a geração do seu NodeID sendo o único nó existente na rede. Se um nó é o único existente na DHT ele é responsável por armazenar todas as chaves no espaço de endereçamento e assinar mensagens do tipo join() para conhecer os NodeIDs dos demais nós que se juntarem à rede;
3. Cada novo nó que juntar-se a uma DHT previamente existente deverá publicar no broker o seu NodeID por meio da mensagem join(). Além de publicar o seu próprio NodeID, todos deverão assinar esse tipo de mensagem para conhecer os NodeIDs dos demais nós e definir os intervalos de responsabilidade no espaço de endereçamento: cada nó precisa conhecer o seu antecessor e o seu sucessor no anel virtual da DHT;
4. Após o ingresso de um novo nó na DHT é necessário que todos os nós recalculem o espaço de armazenamento sob sua responsabilidade. Por simplificação, não será requerido que as chaves anteriormente armazenadas sejam redistribuídas, elas podem ser simplesmente perdidas. A implementação da redistribuição das chaves ao novo nó será contado como pontuação extra na nota do Trabalho T2;
5. Os nós da DHT deverão assinar mensagens do tipo put(k,v) e get(k). A primeira é para inserir um conteúdo v na DHT por meio da chave k. A segunda é para recuperar o conteúdo v armazenado na DHT (ou nulo) a partir da chave k. Todos os nós da DHT receberão todas as mensagens de put() e get() e avaliarão se o armazenamento ou recuperação é de sua responsabilidade por meio da avaliação da chave k;
6. Os nós deverão publicar mensagens de confirmação para a mensagem put(), informando que o armazenamento foi realizado com sucesso e informando o NodeID do nó que se responsabilizou pelo armazenamento. Somente o nó responsável pelo

armazenamento da chave k precisa publicar essa confirmação. Pode-se ignorar erros de armazenado (assuma que <k,v> serão sempre válidos);
7. Os nós deverão publicar mensagens de resposta para a mensagem get(), informando o valor do conteúdo v armazenado. Somente o nó responsável pelo armazenamento da chave k precisa publicar essa resposta. Pode-se ignorar erros de armazenado (assuma que <k,v> serão sempre válidos);
8. As mensagens de put(), get() na DHT serão publicadas por nós (ou processos, ou VMs, ou threads, ou computadores) externos à DHT. Estes são considerados clientes do serviço de DHT. Esses clientes deverão assinar as mensagens de resposta aos put(), get() no broker;
9. A qualquer momento um nó poderá publicar uma mensagem de leave() indicando que deixará a DHT e aguardar a confirmação dos demais nós. Após receber as confirmações, o nó poderá sair da rede e o espaço de endereçamento sob responsabilidade de cada nó deverá ser novamente recalculado. Também não há necessidade de implementar a redistribuição das chaves sob a responsabilidade do nó que está deixando a rede, que valerá como pontuação extra no Trabalho T2;
7. O formato das mensagens é de livre definição do grupo, devendo ser documentado na documentação do trabalho. Os grupos têm autonomia para tomar decisões de implementação e criação de novas mensagens, desde que seja documentado.

Requisitos:
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 poderão implementar os trabalhos usando qualquer uma dentre as três linguagens de programação: C, Java ou Python. Os grupos também poderão usar quaisquer algoritmos ou bibliotecas para implementar as tabelas hash locais;
3. A submissão deverá ser feita por meio de um *link* com a disponibilização do código no Github, em modo de acesso público ou, minimamente, que meu e-mail rodolfo.villaca@ufes.br tenha direito de acesso ao código;
4. A documentação para uso/teste do seu programa deverá ser feita na própria página do Github através do arquivo README[3];
5. O grupo deverá gravar um vídeo de no máximo 10 min apresentando o funcionamento/teste do trabalho;
5. Avaliação: Adequação aos Requisitos (30%), Legibilidade do Código (30%), Documentação (40%), Extras (10%);
6. Data de Entrega: 06/10/2021 pela Sala Virtual da disciplina no Google Classroom;

Bom trabalho!

3 https://guides.github.com/features/wikis/