Universidade Federal de Santa Catarina – UFSC
Departamento em Informática e Estatística
INE 5645 – Programação Paralela e Distribuída
Semestre 2024/1

Definição do Trabalho 2: Programação Distribuída

Este trabalho visa explorar o uso de padrões para programação distribuída. Cada grupo irá explorar pelo menos um padrão de projeto para programação distribuída. A seguir são apresentados quatro enunciados e cada grupo ficará encarregado de desenvolver o projeto enunciado em um deles. A escolha de enunciado por grupo será feita por sorteio.

1. Implementação de um serviço do tipo *pub/sub* que funciona como um *feed* de rede social

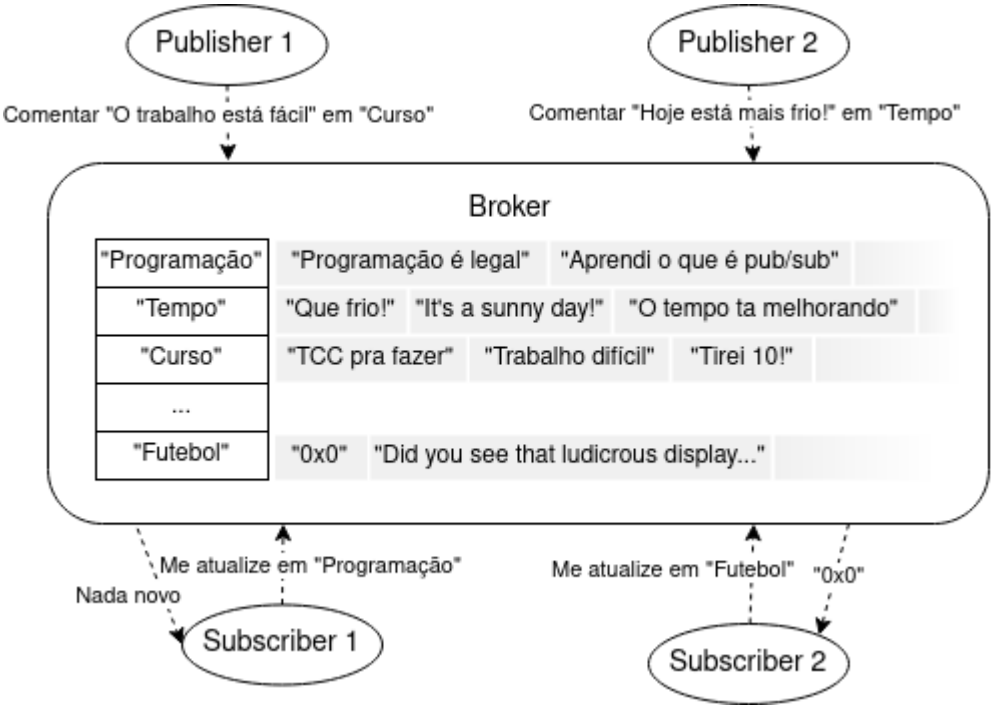
Publishers são aqueles que produzem comentários na rede social, sendo que cada comentário é associado a um assunto, e os *subscribers* são aqueles interessados em determinado(s) assunto(s).

- Cada comentário é composto por um texto e é associado a um único assunto;
- Um nó *broker* armazena os assuntos recebidos dos *publishers*, mantendo um conjunto de comentários para cada assunto. Ao receber requisições pedindo por atualizações em um assunto de interesse, ele deve enviar os comentários mais recentes ainda não recebidas pelo *subscriber;* Ao receber uma requisição de publicação de um novo comentário, ele armazena o comentário recebido;
- Cada *publisher* envia comentários ao *broker*;
- Cada *subscriber* pede ao *broker* comentários atualizados de assuntos que assina*;*

Outros requisitos:

- A comunicação deve ser implementada utilizando *sockets* ou *MPI. Se optar por usar MPI, não poderá ser utilizado recursos de comunicação coletiva ou implementações do MPI de padrões como scatter/gather e map/reduce;*
- Cada comunicação realizada deve ser exibida na tela (*print*);
- Para testes e simulação, devem ser criados arquivos de configuração, sendo as seguintes configurações:
    - Para o *publisher,* teria o próprio endereço, o endereço do *broker* e uma lista de pares (comentário, assunto), que serão enviados ao *broker* ao longo da execução;
    - Para o *subscriber,* o próprio endereço, o endereço do *broker* e uma lista de assuntos assinados;
    - A configuração do *broker* contém apenas o próprio endereço.
    - Publicações e pedidos por atualização devem ser feitos em intervalos aleatórios de 1 a 2 segundos.

Ilustração de Apoio:



2. DNS particionado

Um ***Domain Name System*** (**DNS**) faz associação entre nomes de domínios e entidades participantes. A sua utilização mais convencional associa nomes de domínios mais facilmente memorizáveis a endereços IP, necessários à localização e identificação de serviços e dispositivos, processo esse denominado **resolução de nomes**. Um sistema como esse pode receber uma grande quantidade de requisições de resolução, e isso pode fazer com que seja muito caro encarregar um único servidor de armazenar todos os dados. Em vista disso, podemos particionar o serviço, adicionando mais servidores e distribuindo uma porção dos dados para cada um.

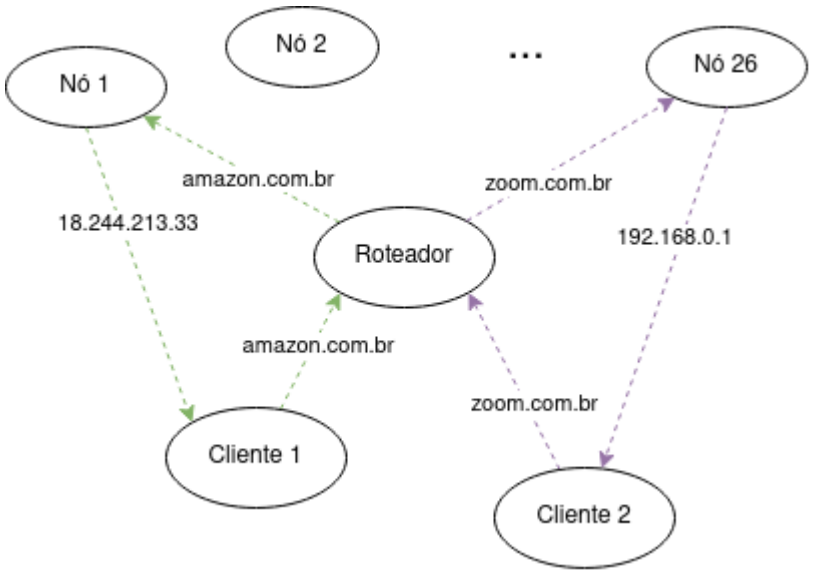
Para o trabalho você deve implementar um serviço de DNS particionado, composto por um **nó roteador**, e **nós de partição**. Cada **cliente** envia requisições ao nó roteador, que redireciona a um dos nós de partição, que por sua vez responde o cliente. O número de partições é fixo, sendo 26 partições, uma para cada letra do alfabeto. A primeira partição armazena os **nomes** iniciados por “A”, a segunda partição os nomes iniciados por “B”, e assim sucessivamente. A relação de nomes e endereços IP deve ser armazenada em arquivo.

Outros requisitos:

- A comunicação deve ser implementada utilizando *sockets* ou *MPI. Se optar por usar MPI, não poderá ser utilizado recursos de comunicação coletiva ou implementações do MPI de padrões como scatter/gather e map/reduce;*
- Cada comunicação realizada deve ser exibida na tela (*print*);
- Para testes e simulação, devem ser criados arquivos de configuração:
    - Para cada partição, o arquivo de configuração tem seu endereço e uma relação de nomes para endereços IP;
    - O roteador tem o próprio endereço e os endereços das partições;

- Os clientes precisam do endereço do roteador e uma lista de requisições que serão feitas durante a simulação. Requisições com nomes existentes e inexistentes.
- Requisições devem ser feitas em intervalos aleatórios de 1 a 2 segundos.

Ilustração de Apoio:



3. Serviço de armazenamento de chave-valor com um mecanismo de balanceamento de carga com replicação

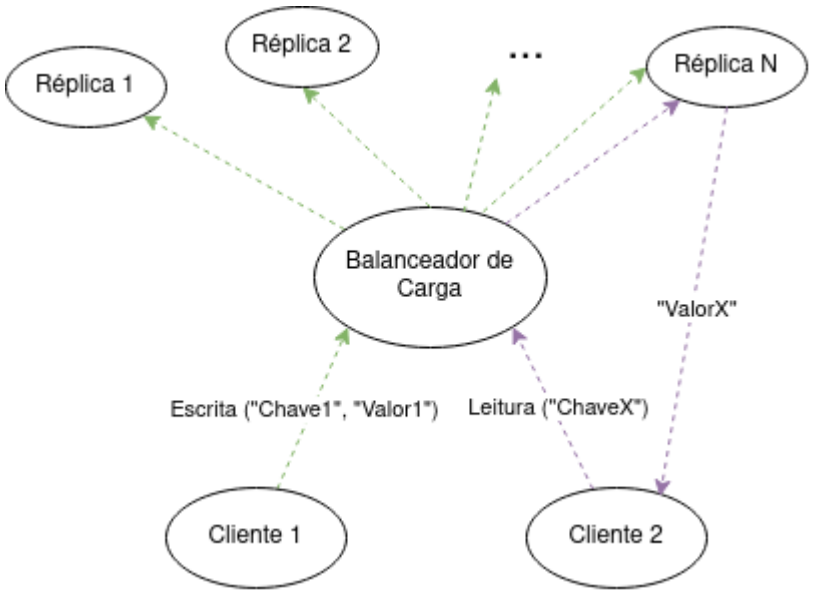
Um **banco de dados de chave-valor** é um sistema projetado para armazenar, recuperar e gerenciar dados de forma semelhante às estruturas de dados comumente conhecidas como dicionários ou tabelas *hash.* Uma forma de oferecer tolerância a falhas a um sistema desse tipo é utilizando replicação. Quando um sistema possui réplicas operando, é possível distribuir as requisições entre essas réplicas, com o objetivo de aumentar o desempenho do sistema.

Para o trabalho, você deve implementar um banco de dados chave valor replicado com balanceamento de carga. Cada **cliente** envia requisições de **leitura** ou **escrita** de uma chave ao **nó balanceador de carga**. Uma requisição de leitura (ex. leitura(chave): return valor) carrega uma “chave”, e deve ser respondida com o valor correspondente, enquanto uma requisição de escrita (ex. escrita(chave, valor): return void) é composta por uma chave e um valor, que devem ser associados pelo sistema, e não exige resposta. O balanceador de carga redireciona requisições de leitura a uma das **réplicas** do banco, que mantém os dados e respondem às requisições de leitura diretamente aos clientes. A réplica que atenderá a requisição de leitura é definida de forma **circular**. A replicação do sistema se limita à difusão das escritas. Portanto, requisições de escrita recebidas pelo balanceador de carga devem ser **difundidas** para todas as réplicas, utilizando qualquer estratégia de difusão.

Outros requisitos:

- A comunicação deve ser implementada utilizando *sockets* ou *MPI. Se optar por usar MPI, não poderá ser utilizado recursos de comunicação coletiva ou implementações do MPI de padrões como scatter/gather e map/reduce;;*
- Cada comunicação realizada deve ser exibida na tela (*print*);
- Para testes e simulação, devem ser criados arquivos de configuração:
    - Para as réplicas, basta seu endereço;
    - O balanceador precisa ter o próprio endereço e os endereços das réplicas;
    - Os clientes precisam do endereço do balanceador e uma lista de requisições que serão feitas durante a simulação.
    - Requisições devem ser feitas em intervalos aleatórios de 1 a 2 segundos

Ilustração de Apoio:



4. Serviço de recuperação de texto com o padrão *scatter/gather* para dividir o trabalho

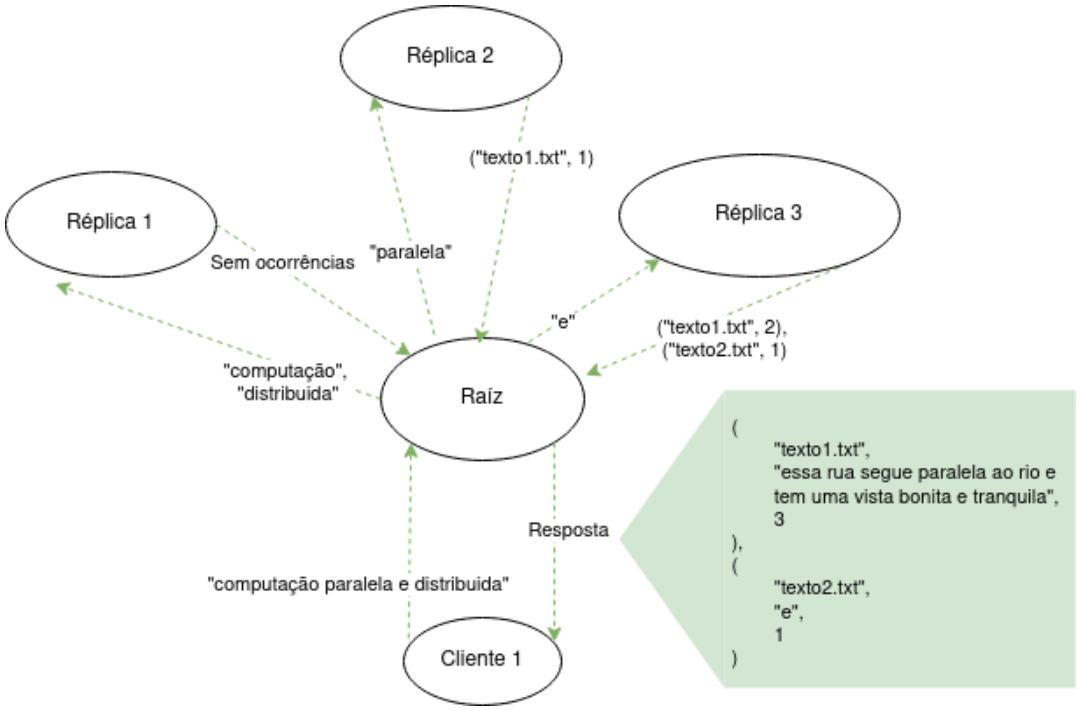
Um sistema de recuperação de texto (*text retrieval*) é um tipo de sistema de recuperação de informação especificamente projetado para encontrar e recuperar documentos ou trechos de texto relevantes de uma coleção maior com base nas consultas dos usuários.

Para o trabalho, você deve implementar um serviço de recuperação de texto utilizando o padrão *scatter/gather.* Um **cliente** faz requisições de consulta com um ***string* de busca** ao **nó raiz** do sistema. O nó raiz divide o *string* de busca em um conjunto de palavras-chave e envia porções iguais do total de palavras para as **réplicas**, que armazenam os **arquivos de texto plano**. Cada réplica responde ao nó raiz com os arquivos e o número de ocorrências de palavras chave. O nó raiz combina as respostas das réplicas e responde os clientes com uma lista de textos, juntamente com o número de ocorrências.

Outros requisitos:

- A comunicação deve ser implementada utilizando *sockets* ou *MPI. Se optar por usar MPI, não poderá ser utilizado recursos de comunicação coletiva ou implementações do MPI de padrões como scatter/gather e map/reduce;*
- Cada comunicação realizada deve ser exibida na tela (*print*);
- Para testes e simulação, devem ser criados arquivos de configuração:
    - Para as réplicas, basta seu endereço;
    - O nó raiz precisa ter o próprio endereço e os endereços das réplicas;
    - Os clientes precisam do endereço do nó raiz e uma lista de requisições que serão feitas durante a simulação;
    - Deve haver arquivos de texto à disposição do sistema para busca;
    - Requisições devem ser feitas em intervalos aleatórios de 1 a 2 segundos.

Ilustração de Apoio:



Execução

Você pode desenvolver o programa em qualquer linguagem de programação. Os testes devem seguir as especificações descritas em cada enunciado.

Entrega

O trabalho consiste em:

1. Implementar um programa que resolva o enunciado sorteado para o seu grupo;
2. Breve relatório que indique as principais decisões e estratégias de implementação utilizadas, instruções sobre como compilar e executar o código produzido, além de exemplos de saídas de execução com diferentes parametrizações.

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

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.

Referências úteis

https://www.enterpriseintegrationpatterns.com/patterns/messaging/toc.html

https://learn.microsoft.com/pt-br/azure/architecture/patterns/