INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

LICENCIATURA EM ENGENHARIA DE INFORMÁTICA E COMPUTADORES

SEMESTRE DE VERÃO 2010/2011

Sistemas Distribuídos

1ª Serie - Relatório

Grupo:

- G05D

Alunos:

- João Pires 31933
- José Casimiro 32713
- António Dente 33168

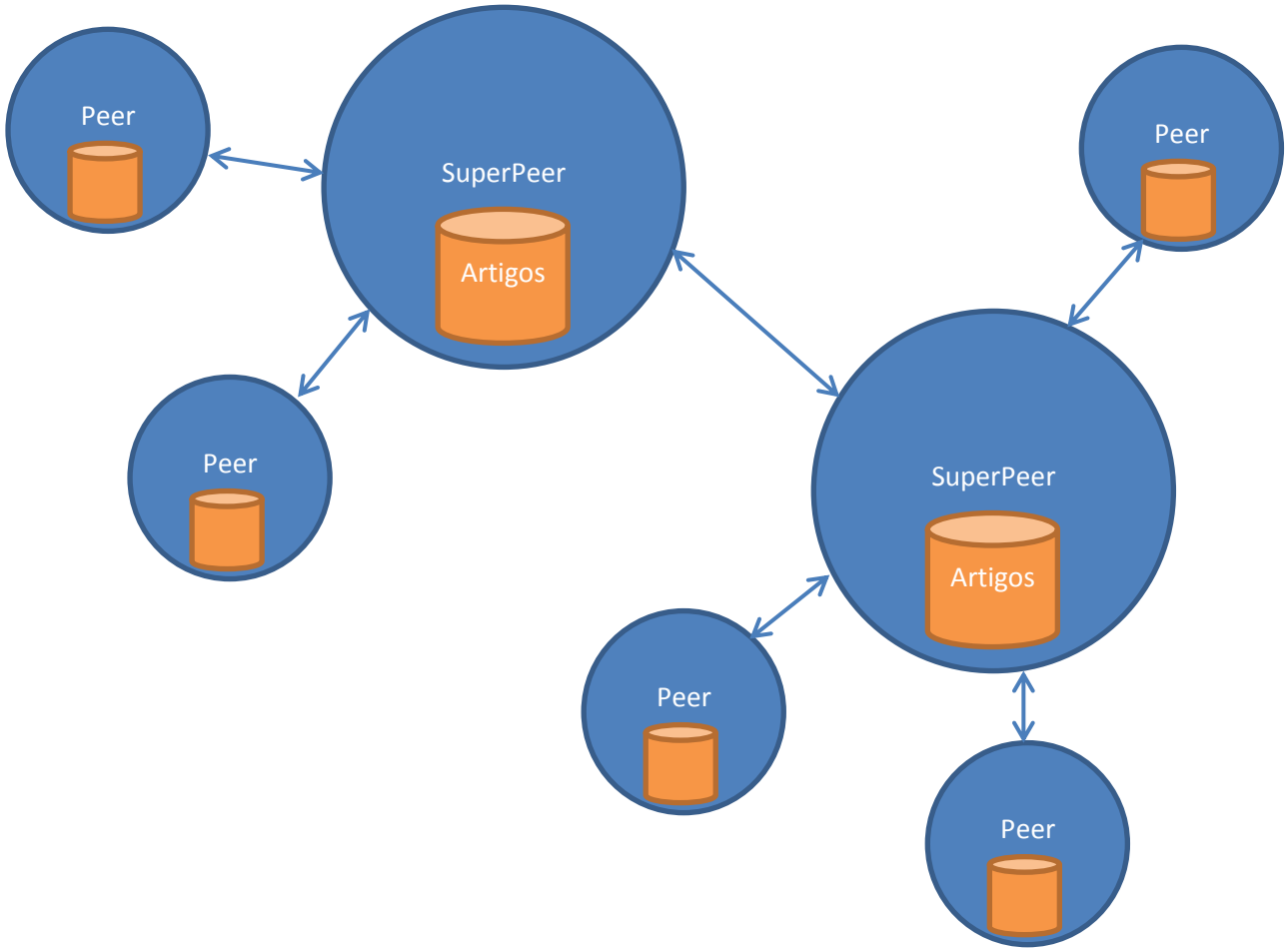
Índice

Introdução

Este relatório tem como finalidade apresentar uma solução de implementação de uma aplicação distribuída que consiste num sistema de partilha de artigos seguindo uma arquitetura “Peer-to-Peer”. Ao longo de relatório serão apresentadas as problemáticas envolvidas na implementação de um sistema distribuído bem como as soluções por nós encontradas e a sua motivação.

Arquitetura

Existem duas entidades que constituem a base da arquitetura, Peer e SuperPeer.

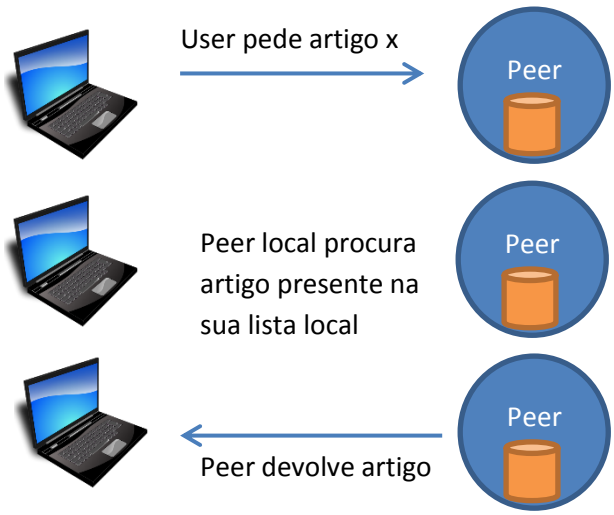


Um Peer aloja e partilha um conjunto de artigos científicos e está sempre ligado a um SuperPeer. Um SuperPeer para além de oferecer as mesmas funcionalidades que um Peer ainda possibilita a agregação de vários Peers bem como a associação a outros SuperPeers servindo assim de intermediário para outras redes de SuperPeers.

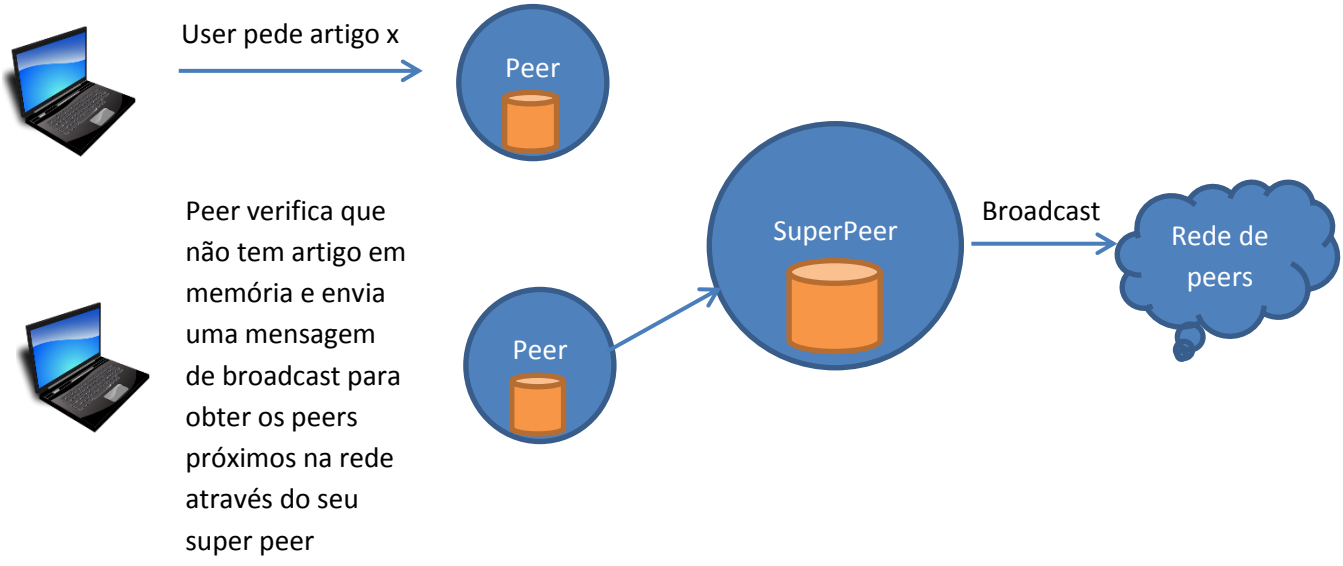
Interação entre as partes

Existem diferentes cenários de interação entre os intervenientes do sistema tais como:

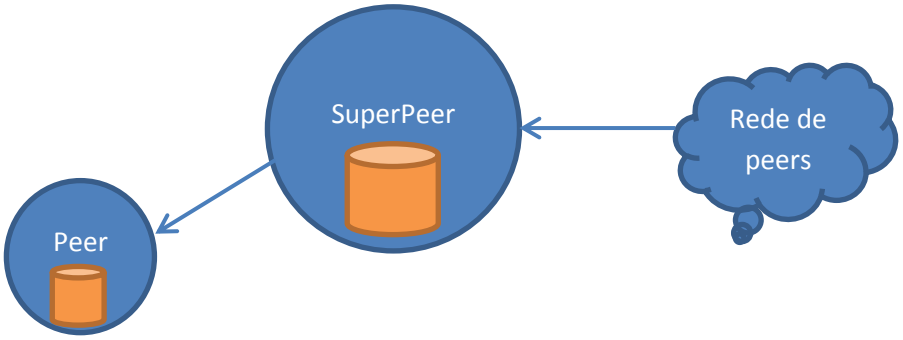
1 º - Utilizador pede ao peer local um artigo que este contem



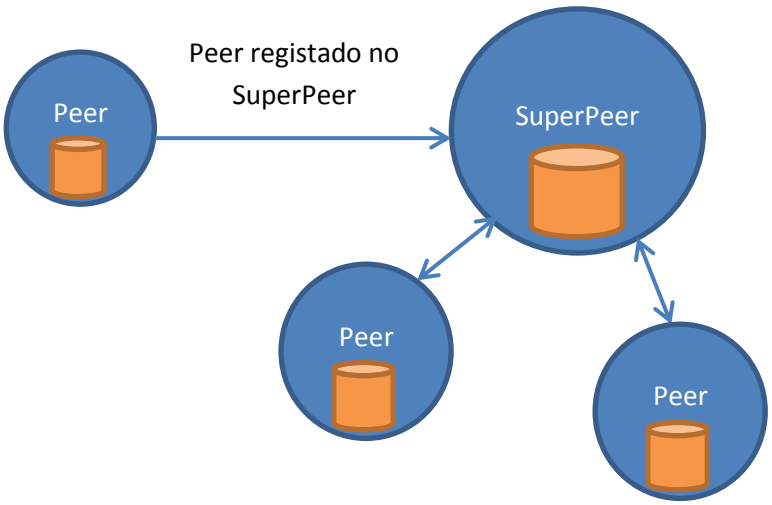
2 º - Utilizador pede ao peer local artigo existente noutro peer da rede

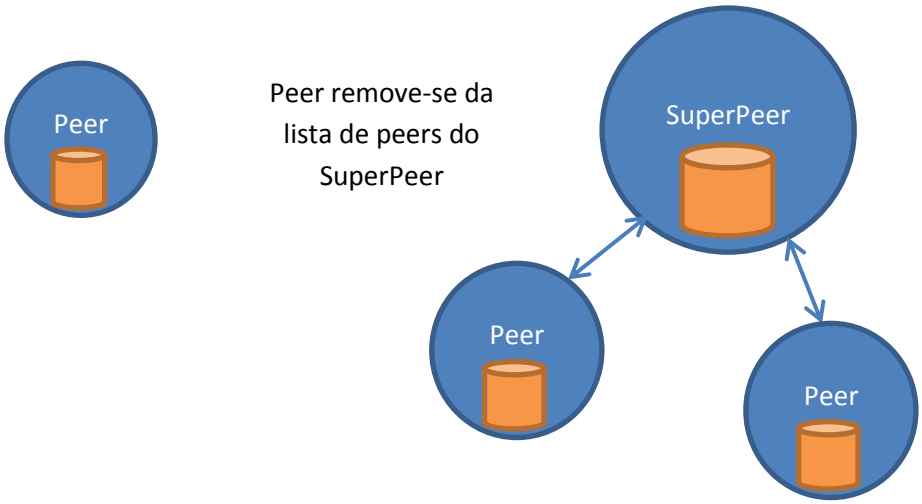


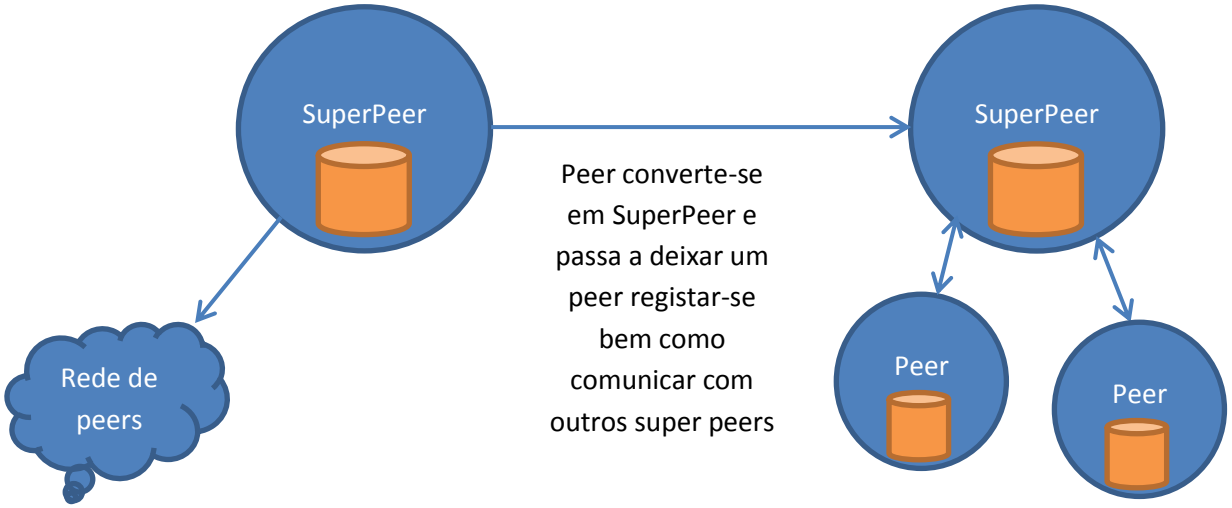
O super peer retorna a lista de potenciais peers para o peer local que por sua vez percorre-a e encontra o artigo na lista local de um dado peer obtido



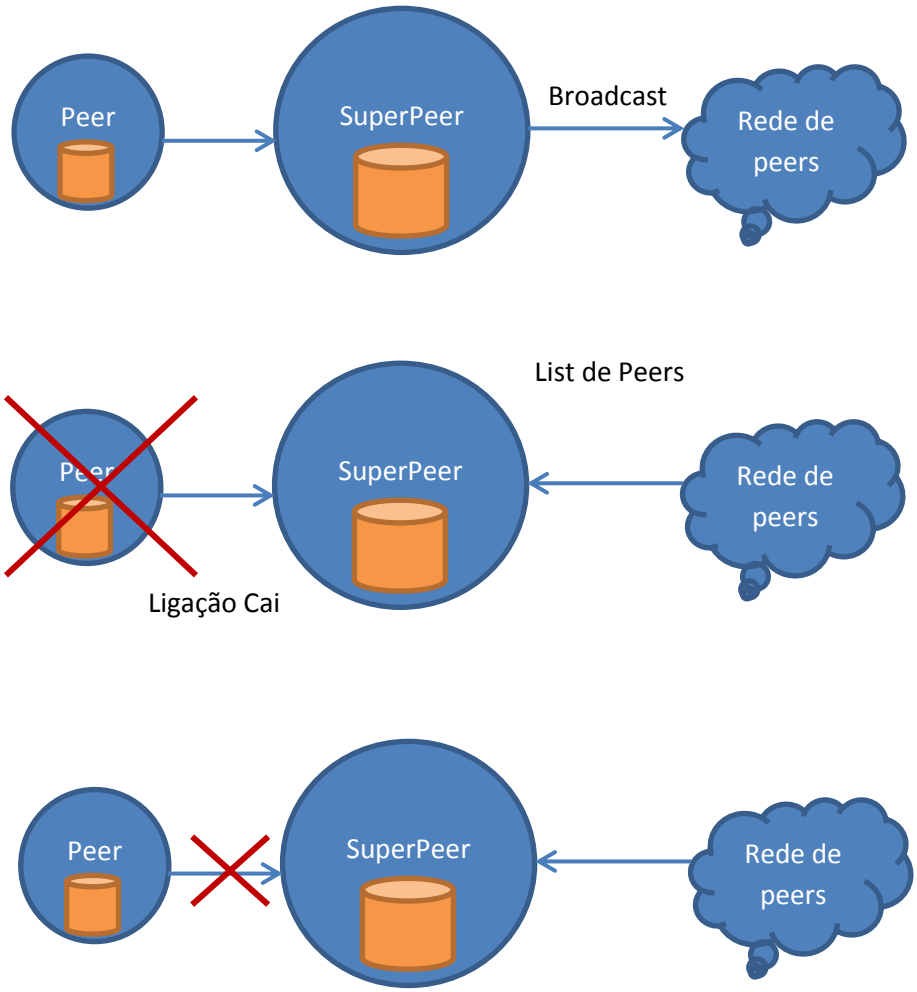
3 º - Peer local passa a SuperPeer







4 º - Peer pede ao SuperPeer associado potenciais peers e cai a ligação antes da resposta ao pedido



O SuperPeer verifica que não consegue comunicar com o peer e remove-o da sua lista de peers conhecidos

Requisitos funcionais e não funcionais

Após a análise do enunciado e um estudo sobre os requisitos base da arquitetura, realizámos um levantamento dos requisitos funcionais e não funcionais do sistema.

Requisitos funcionais

- Um *Peer/SuperPeer* pode pesquisar por artigos dado o nome específico;
- Um *Peer* pode a qualquer altura tornar-se num *SuperPeer;*
- A pesquisa de artigos num *Peer*/*SuperPeer* envolve, no caso de um artigo não existir localmente, numa procura pelos *Peers* conhecidos por artigos.
- Se uma pesquisa de artigo, por nome, não encontrar o artigo localmente ou nos *Peers* conhecidos, será feita então um pedido *GetPeers* ao *SuperPeer* associado, de forma a aumentar os *Peers* conhecidos, ao obter novos *Peers* será realizada uma nova pesquisa sobre os *Peers* mais recentes;
- Um pedido *GetPeers* a um *SuperPeer* resulta sempre numa cadeia de pedidos *GetPeers* aos *SuperPeers* conhecidos, gerando então uma *Network* de *Peer;*
- É possivel observar os pedidos feitos ao *Peer* local.

Requisitos não funcionais

- Devem ser utilizados ficheiros de configuração de forma a indicar a qual *SuperPeer* um *Peer* se vai registar;
- Os *Peers* devem ser completamente independentes, no caso de um *Peer* não conhecer um *SuperPeer* deve ser dada a possibilidade de se registar num em *runtime*;
- Os pedidos devem ter uma forma de não se propagarem indefinidamente dentro da *network*;
- Controlo sobre *Peers/SuperPeers offline* deve estar sempre presente sobre todas as comunicações realizadas entre as partes;

Aspetos relevantes da implementação dessa arquitetura realçando os pontos fortes e fracos da solução

Tracking dos Peers

Um ponto importante é como realizámos as listas de peers (proxys) nas outras instâncias. Escolhemos um dicionário de int para IPeer/ISuperPeer pois é criada uma nova proxy quando se passa uma entidade MarshalByRef por parâmetro pelo que não é possível comparar esta proxy com os armazenados numa lista através do método contains. Deste modo criámos um Id em cada instância utilizando Datetime.Now.Ticks.GetHashCode(), para que a probabilidade de haver um Id igual em entidades diferentes seja o mais baixo possível, e utilizamos este Id como Key do proxy no dicionário. Este método de armazenamento também facilita a remoção de proxys cujas conexões estão fechadas. Ao iterar por uma lista não nos é possível remover um elemento dessa lista no momento da iteração, ao conter um dicionário podemos armazenar as key's para remoção após a iteração.

Tratamento de falhas

Uma grande preocupação nossa foi a tolerância a falhas. Estas ocorrem maioritariamente por conexões mal terminadas (peers/superpeers terminados sem se “desregistarem”), pelo que sempre que acedemos a uma proxy, seja no getPeers ou no getArticle, verificamos se a conexão com o peer em questão ainda se encontra ativa, em caso negativo removemo-lo da lista.

Uma grande falha seria um pedido de getArticle ou getPeers ficar preso em deadlock ou um loop infinito. Para tratar estas situações colocámos um parâmetro booleano no método getArticle que indica se o método chama getPeers ou apenas verifica nos seus artigos e nos dos peers que consegue contactar diretamente. Deste modo, a instância que está à procura de um artigo procura em si, nos peers que consegue contactar diretamente e, caso ainda não tenha encontrado o artigo, nos peers retornados pelo método GetPeers do (s) SuperPeer (s) a que está conectado, mas estes apenas procuram em si e nos diretamente acessíveis.

O método getPeers recebe dois parâmetros, um indica o Id da instância que chamou o getPeers para que este não seja testado se está "vivo" pois ocorre um deadlock visto o método ser síncrono, o outro parâmetro é um contexto de pedido. Este contexto é criado no método getArticle e indica o número de saltos possíveis (valor omissão = 5) e contém uma lista de id's que identifica os SuperPeers a quem já foi chamado o método getPeers. O objeto é Serializable para que o valor de saltos seja copiada para cada "nível" de SuperPeers e a lista em si encontra-se envolvida por um objeto MarshalByRef para que seja partilhada entre todas as instâncias. Assim temos a garantia que o pedido não vai passar por um SuperPeer mais do que uma vez e que não passa o número de "níveis" especificado pelo número de saltos.

Sincronismo

Também para tentar evitar alguns erros, a nossa aplicação que utiliza windows forms realiza o "desregisto" do peer/superpeer quando é terminada, mesmo que não seja especificado pelo utilizador.

Um ponto fraco na nossa arquitetura é o método getPeer estar a chamar o método respetivo nos SuperPeers a que se encontra conectado, sincronamente. Isto causa um tempo de espera maior além de ser um caso onde se podia chamar o método assincronamente em todos os SuperPeers e esperar que todos respondessem. Não foi concretizada a solução descrita devido ao acréscimo de complexidade ao método em si além de não ser um dos objetivos do trabalho, embora seja um ponto que gostaríamos de melhorar.

Thread Confinement

Tivemos também alguns problemas na atualização da UI para informação sobre métodos chamados por outros Peers, visto a atualização de ui ter de ser realizada na thread de criação da ui. Para resolver este caso temos dois passos. Primeiro utilizamos um evento que despoletamos na chamada aos métodos, este utiliza depois um BackgroundWorker para que a atualização seja feita na thread correta.

Conclusão

A realização deste trabalho permitiu-nos ficar a conhecer algumas das problemáticas envolvidas na implementação de sistemas distribuídos bem como algumas das soluções. Permitiu-nos ainda conhecer um pouco da framework de .NET Remoting e algumas das funcionalidades que esta oferece.