[O.1.1]. Explicar por palavras próprias o conceito de Sistemas Distribuídos e as propriedades que os caracterizam, nomeadamente a existência de máquinas autónomas ligadas em redes, a não existência de relógio global, concorrência, falhas independentes com possibilidade de evitar pontos de falha única, heterogeneidade e prestação de funcionalidades integradas.

O SD é um sistema, constituído por várias máquinas autónomas e heterogéneas ligadas em rede, em que cada máquina executa determinados componentes que comunicam e coordenam as suas ações apenas através da troca de mensagens, sobre um protocolo. Assim consegue-se um melhor desempenho do sistema e uma maior capacidade de processamento a menor custo e é possível suportar num único sistema diversos tipos de rede, computadores, SO, linguagens de programação, etc. É, portanto, mais económico, tem maior desempenho, heterogeneidade, robustez e escalabilidade, distribuição inerente (algumas aplicações envolvem naturalmente computadores independentes), expansão (poder de computação pode ser adicionado gradualmente).

Caraterísticas fundamentais:

- Concorrência – um sistema distribuído partilha processamento entre vários hardwares, tem de haver concorrência – que permite que dois processos/threads acedam simultaneamente a um recurso sem causar problemas - que quando bem gerida aumenta o desempenho;
- Não há relógio global – se o processamento é distribuído entre várias máquinas, e estas têm diferentes localizações claro que o mecanismo de sincronização não poderá ser o relógio da máquina, existirão outros;
- Falhas independentes – uma das principais vantagens; se um componente de um SD falhar, pode não afetar outros sistemas e/ou componentes no mesmo;
- Heterogeneidade – qualquer máquina, com qualquer hardware pode incorporar um SD, devido à utilização de protocolos preparados etc...;
- Prestação de funcionalidades integradas – apesar de quando o utilizador solicita uma funcionalidade a um SD, este ter que distribuir o processamento por várias máquinas, esta funcionalidade aparece ao utilizador como se tivesse corrido apenas na máquina que ele correu a funcionalidade;

1.Explique o que significa conceber um Sistema Distribuído sem pontos de falha única.

É conceber o sistema distribuído sem um componente que seja crucial ao funcionamento do sistema. Se falha o serviço de mensagens, não falha o serviço de contactos. Por exemplo, No Skype, ao falhar o serviço central não dá para fazer login. Ao replicarmos, o sistema é mais escalável e tolerante a faltas e peers-to-peers.

[O.1.2]. Perante a descrição de um sistema informático caracterizá-lo do ponto de vista da sua abordagem à distribuição, identificando quais as partes desse sistema que correspondem de alguma forma ao conceito de Sistema Distribuído.

Distinguir um sistema distribuído de um sistema centralizado:

- SC – Informação numa única maquina. Servidor faz tudo;
- SD – Recursos divididos pelas varias maquinas. Há comunicação com varias entidades.

Os sistemas informáticos podem estar distribuídos em vários componentes, mas estes não são autónomos. Um sistema informático caracteriza-se por ser constituído por um programa que é escrito utilizando uma linguagem de programação e guardado na memória secundária, sendo depois carregado para a memória principal e utilizado pelo CPU originando um ou mais processos que podem interagir através da troca de mensagens. A memória e o processador são os elementos destes sistemas que podem ser distribuídos

[O.1.3]. Identificar os principais desafios ao desenvolvimento de Sistemas Distribuídos, nomeadamente Heterogeneidade, Abertura, Segurança, Escalabilidade, Tolerância a faltas, Concorrência e Transparência, e para cada um deles explicar por palavras próprias o seu significado e as suas implicações para o desenvolvimento de sistemas informáticos.

- Heterogeneidade – suportar várias máquinas com diverso hardware e SO's. Implica suportar diferentes redes, computadores, sistemas operativos linguagens de programação, componentes. Para ultrapassar isto usam-se middlewares, tipos de dados conformes, protocolos de comunicação, abstração de dados.
- Abertura – o sistema pode ser estendido e reimplementado de muitas formas, ou seja, cada um pode desenvolver uma implementação dos componentes do sistema, desde que mantenha a conformidade, para poder comunicar com os outros componentes existentes. Provoca reduções de custos, efeito de rede, concorrência entre fabricantes, inovação. Mas implica boa documentação.

- Segurança – o risco é acrescido em SD's, podem haver vários tipos de ataque, para os quais o sistema tem de estar preparado. Tem de garantir: confidencialidade, integridade, autenticação, não-repúdio, disponibilidade. Utiliza-se frequentemente: criptografia, certificados, sumários, etc...;
- Escalabilidade – consegue manter o seu funcionamento mesmo com um aumento de escala muito grande em termos de recursos e utilizadores. Utiliza-se frequentemente: replicação e caching;
- Tolerância a faltas – capacidade para manter o seu funcionamento, ainda que parcial, quando ocorre uma falta, ou seja, uma falha num dos componentes. Se uma falha num componente implicar uma falha no sistema, então quanto mais distribuído mais frágil. Num SD as falhas têm de ser independentes e parciais, um componente falhou, mas os outros podem funcionar (houve uma falta no sistema e não uma falha). Graceful degradation é o caso em que se há uma falha num ou mais componentes, os outros continuam a funcionar, mas o desempenho do sistema distribuído diminui. Técnicas habituais: replicação, redundância, desenhar o sistema sem pontos de falha únicos (falha um componente, falha tudo).

- Concorrência – aceder concorrentemente a recursos, logo tem de haver programação concorrente. Técnicas utilizadas: semáforos, funções sincronizadas.
- Transparência – capacidade de mostrar as suas funcionalidades como se estas tivessem sempre integradas e não partilhadas. O utilizador não deve ter a necessidade de lidar com os detalhes de distribuição do sistema. Tipos de transparência: acesso, localização, concorrência, replicação de dados, faltas, mobilidade (recursos movidos sem se dar por ela), desempenho (mudar configuração para melhor desempenho), escalabilidade. Técnicas habituais: organização por camadas, API's;

[O.2.2]. Explicar por palavras próprias o modelo de programação cliente-servidor e as suas variantes, e.g. uso de proxy ou clusters de servidores.

O modelo de programação cliente-servidor estrutura o SD em dois grupos de processos:

- servidores: processos que oferecem serviços;
- clientes: processos que requerem serviços.

Neste modelo, os serviços são acedidos através de uma interface partilhada por todos os clientes (e.g. browser), sendo que todas as interações entre clientes e servidores são especificadas num protocolo que estabelece as regras de comunicação, definindo que mensagens podem ser trocadas, qual o conteúdo e o formato das mensagens (e.g. TCP/UDP). Já a especificação das invocações a realizar ao servidor encontra-se disponível em bibliotecas (API) ou através de middleware. Uma variante deste modelo é a estrutura cliente – servidor com proxy – que é um intermediário na interação com o servidor – particionado.

Uma segunda variante deste modelo é a estrutura cliente – servidor com cluster de servidores – em que um cliente comunica não só com um servidor, mas com vários – replicado.

O.2.3]. Explicar por palavras próprias o modelo de programação distribuída peer-to-peer fiP2P).

No modelo de programação peer-to-peer todas as entidades desempenham um papel semelhante e interagem de forma cooperativa sem distinção entre clientes e servidores. Os padrões de interação podem variar em função das circunstâncias ou da tarefa. Um exemplo de uma aplicação peer-to-peer é o Skype – todos as aplicações de chat.

[O.2.4]. Perante a descrição de um sistema distribuído caracterizá-lo do ponto de vista do seu modelo de distribuição – dizer se é peer-to-peer ou cliente-servido

- Código móvel – variação do modelo cliente-servidor em que o cliente começa por descarregar o código do servidor para aceder às funcionalidades do serviço. Os pedidos são feitos ao código transferido. Exemplos: javascript, java applets, jini.
- Agentes móveis – programa em execução que pode ser transferido de um computador para outro (código + dados = estado). É o modelo ideal para tarefas que envolvam muitas máquinas, em que cada uma delas manipula muitos dados. O agente é lançado e regressa com os resultados. A implementação é complexa, devido à segurança e controlo de acesso aos recursos.
- Modelo de comunicação em grupo – processos podem entrar ou sair de um grupo e a comunicação é feita para todos os membros do grupo. • Modelo de objetos distribuídos (abstração de invocação remota aplicada a métodos de objetos).
- Produtor-subscritor

[O.4.1]. Explicar por palavras próprias o conceito de Middleware e o seu papel no desenvolvimento de aplicações distribuídas.

Os sockets são uma abstração de muito baixo nível, pelo que dá muito trabalho para os programadores, e não suportam transparência de acesso nem de localização.

A finalidade do middleware é generalizar as funcionalidades envolvidas na maior parte dos programas: representação externa de dados, marshalling/unmarshalling, protocolos de pedido-resposta. Estas funcionalidades representam um processo complexo e com elevados custos. Assim utilizamos o middleware que é uma solução de mais alto nível, baseado numa linguagem de programação existente, que possa ser usado em varias aplicações e esconda toda a heterogeneidade das entidades envolvidas na comunicação. Vantagens Middleware: • Abstrai o que é comum a muitas aplicações; • Mais alto nível; • Desenvolvimento facilitado (desenvolvimento mais rápido, menos propicio a erros, e complexidade reduzida)

Desvantagens: • Menos flexível; • Menos desempenho; • Consome mais recursos;

[O.6.1]. Explicar o conceito de escalabilidade e os desafios gerados a vários níveis pelos aumentos de escala do sistema.

A escalabilidade é um desafio de um sistema distribuído que consiste em conseguir manter o seu funcionamento efetivo sem que para isso implique quebras significativas no seu desempenho, mesmo que haja um aumento significativo no número de utilizadores e de recursos. Os desafios pelos aumentos de escalas do sistema são: • distribuição, operações a realizar pelo sistema que podem ser distribuídas por diversas entidades; • replicação consiste em replicar partes do sistema por várias entidades de forma a melhor o seu desempenho; • cache: partes do sistema que contêm os dados estáticos de um sistema, ou seja que não se alteram, contêm dados que são só de leitura. Ex: sites universidades, das juntas de freguesias.