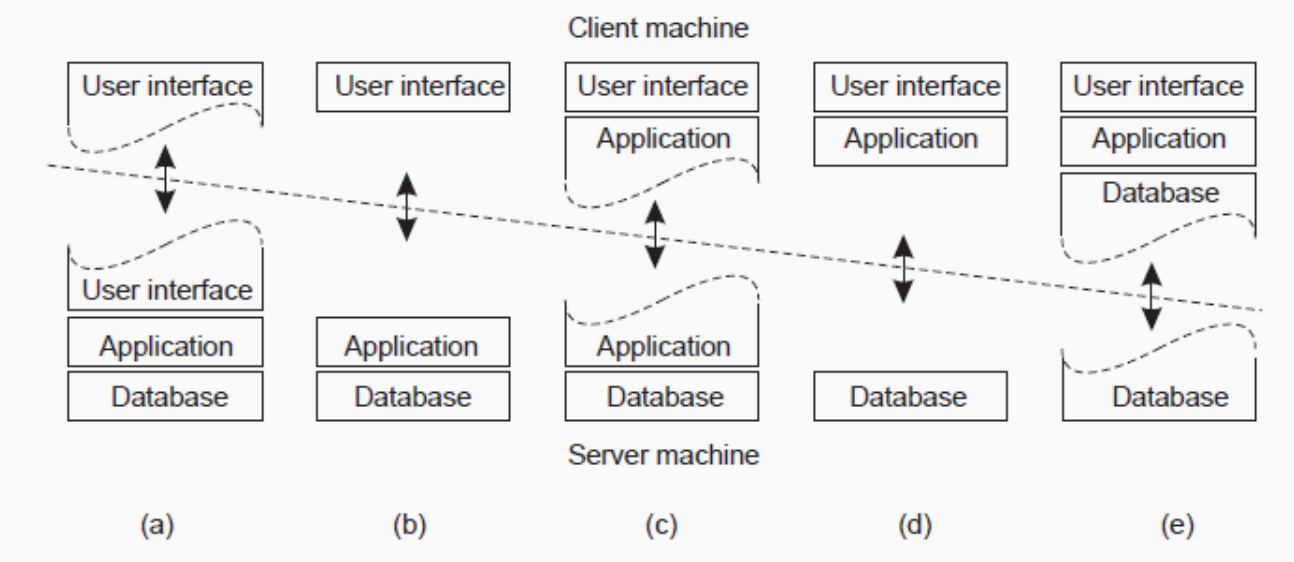
Prova SD da Manhã

Questão 1 - Cliente/Servidor [3,0]

(slide 4)



A - Um exemplo de uma aplicação real para cada uma delas.

A) **- Prompt de Comando (?) / Terminal de check-in do aeroporto - servidor exibe os dados (?)**

B) - Google (?)

C) Uma aplicação que utiliza um form que precisa ser preenchido antes de ser processado, em que a validade dos campos são checados no cliente / ou Google Docs

D)Aplicações bancárias, onde o cliente envia para o servidor apenas as querys de mudanças no banco.

E)Aplicações bancárias, onde parte do cliente tem parte dos dados, e envia para o servidor apenas as querys de mudanças no banco. Outro exemplo é o uso de cache em páginas Web, para guardar cookies e informações de sessão em disco local

B - Antigamente era mais cliente gordo e agora é mais cliente magro. Por que?

Resposta está no Note 2.4, da página 93 do livro 3rd

O termo fat client, é designado para as aplicações que deixam grande parte do processamento da aplicação do lado do usuário final, como pode ser visto nos exemplos (d) e (e) acima. Essa arquitetura tem sido deixada para trás pois, mesmo que a aplicação do cliente faça muito, é mais dificil de gerenciar. Ter mais funcionalidades do lado do cliente implica em assegurar que todos os clientes tenham essa capacidade de processamento, além disso, essas aplicações ficam reféns do SO e recursos, oq pode levar a necessidade de multiplas versões e manutenções . Assim, os clientes que são chamados de fat clients (os que carregam uma boa parte da aplicação no usuário final) não está sendo mais uma boa opção. Já os thin clients, são os (a)-(c), são mais faceis de implementar, mantém uma performance melhor no lado do cliente e menos sofisticados.

C - E hoje em dia web está voltando a ser cliente gordo, por quê?

Para alguns casos, a aplicação com maior processamento no cliente final, ainda é a melhor opção, como aplicaçoes multimedia onde é preciso que o processamento seja feito no lado do cliente. Além disso, hoje em dia, com a evolução crescente dos browsers, é bem mais simples dinamicamente

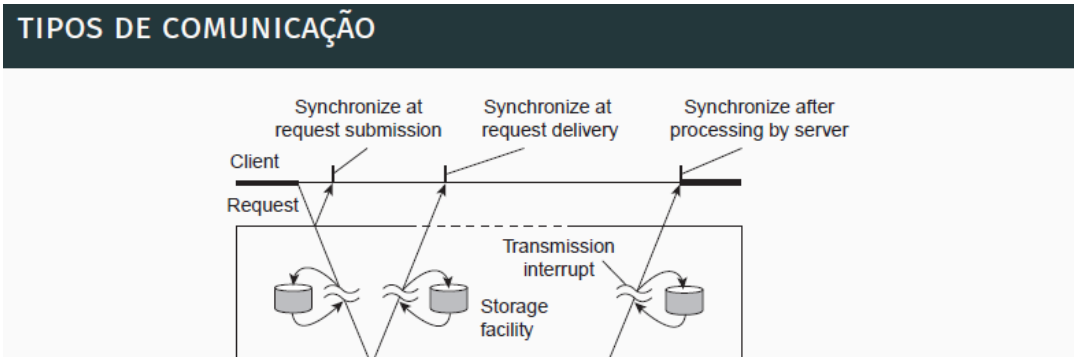
alocar e gerenciar as aplicações do lado do cliente através de scripts. Além disso, neste tipo de aplicação, o SW é implementado em ambientes que não dependem da plataforma, e assim a complexidade de gerenciamento cai.

Note 2.4 (More information: Is there a best organization?)
We note that there has been a strong trend to move away from the configurations shown in Figure 2.16(d) and Figure 2.16(e) in those cases that client software is placed at end-user machines. Instead, most of the processing and data storage is handled at the server side. The reason for this is simple: although client machines do a lot, they are also more problematic to manage. Having more functionality on the client machine means that a wide range of end users will need to be able to handle that software. This implies that more effort needs to be spent on making software resilient to end-user behavior. In addition, client-side software is dependent on the client's underlying platform (i.e., operating system and resources), which can easily mean that multiple versions will need to be maintained. From a systems-management perspective, having what are called **fat clients** is not optimal. Instead the **thin clients** as represented by the organizations shown in Figure 2.16(a)–(c) are much easier, perhaps at the cost of less sophisticated user interfaces and client-perceived performance.

Does this mean the end of fat clients? Not in the least. For one thing, there are many applications for which a fat-client organization is often still the best. We already mentioned office suites, but also many multimedia applications require that processing is done on the client's side. Second, with the advent of advanced Web browsing technology, it is now much easier to dynamically place and manage client-side software by simply uploading (the sometimes very sophisticated) scripts. Combined with the fact that this type of client-side software runs in well-defined commonly deployed environments, and thus that platform dependency is much less of an issue, we see that the counter-argument of management complexity is often no longer valid.

Questão 2 - MOM [1,5]

A - O que é assíncrono e persistência?



Transiente vs. persistente

Comunicação transiente: remetente descarta a mensagem se ela não puder ser encaminhada para o destinatário

Comunicação persistente: uma mensagem é guardada no remetente pelo tempo que for necessário, até ser entregue no destinatário

6/28

(slide 7)

Pontos de sincronização
• No envio da requisição
• Na entrega da requisição
• Após o processamento da requisição

CLIENTE/SERVIDOR

Computação Cliente/Servidor geralmente é baseada em um modelo de comunicação transiente síncrona:

- Cliente e servidor devem estar ativos no momento da comunicação
- Cliente envia uma requisição e bloqueia até que receba sua resposta
- Servidor essencialmente espera por requisições e as processa

Desvantagens de comunicação síncrona:

- o cliente não pode fazer nenhum trabalho enquanto estiver esperando por uma resposta
- falhas precisam ser tratadas imediatamente (afinal, o cliente está esperando)
- o modelo pode não ser o mais apropriado (mail, news)

7/28

(slide 7)

TROCAS DE MENSAGEM

Middleware orientado a mensagens
tem como objetivo prover comunicação persistente assíncrona:

- Processos trocam mensagens entre si, as quais são armazenadas em uma fila
- O remetente não precisa esperar por uma resposta imediata

pode fazer outras coisas enquanto espera

- Middleware normalmente assegura tolerância a falhas

8/28

(slide 7)

B - Explicar detalhadamente como funciona o MOM: put, get, poll, notify

PUT	Adiciona uma mensagem à fila especificada
GET	Bloqueia até que a fila especificada tenha alguma mensagem e remove a primeira mensagem
POLL	Verifica se a fila especificada tem alguma mensagem e remove a primeira. Nunca bloqueia
NOTIFY	Instala um tratador para ser chamado sempre que uma mensagem for inserida em uma dada fila

(slide 7)

(Na nossa pode ser que caia mensagens transientes: sockets, que é diferente do MOM, ou RPC)

Questão 3 - Arquitetura REST [2,5]
Explicar brevemente arquitetura REST e cada um de seus cinco conceitos.

ARQUITETURAS RESTFUL

Vê um sistema distribuído como uma coleção de recursos que são gerenciados individualmente por componentes. Recursos podem ser adicionados, removidos, recuperados e modificador por aplicações (remotas).

1. Recursos são identificados usando um único esquema de nomeação
2. Todos os serviços oferecem a mesma interface
3. Mensagens enviadas de ou para um serviço são auto-descritivas
4. Após a execução de uma operação em um serviço, o componente esquece tudo sobre quem chamou a operação

Operações básicas

Operação	Descrição
PUT	Cria um novo recurso
GET	Recupera o estado de um recurso usando um tipo de representação
DELETE	Apaga um recurso
POST	Modifica um recurso ao transferir um novo estado

11/20

,

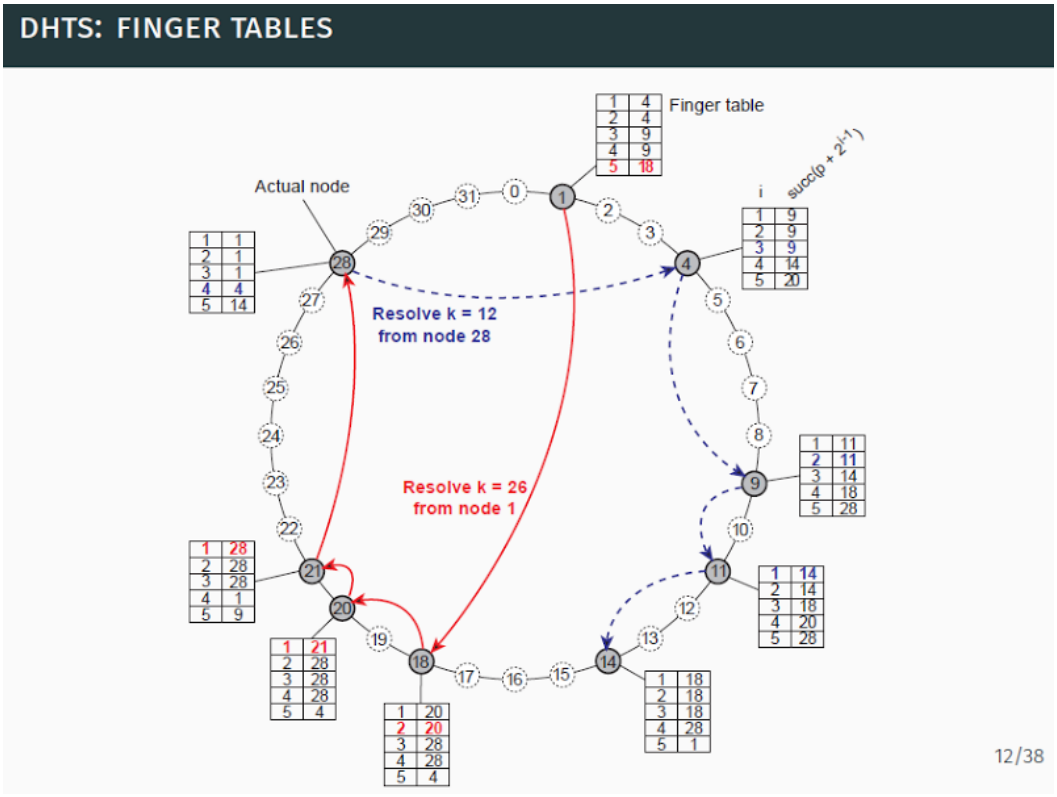
(slide 4)

Questão 4 - Finger Table [3,0]

Dado o tamanho de bits do nome e passou uma lista com os números para montar e realizar os cálculos.

Considere que os nós estejam organizados em forma de anel lógico
• A cada nó é atribuído um identificador aleatório de m-bits
• A cada entidade é atribuído uma única chave de m-bits
• Entidades com chave k estão sob a jurisdição do nó com o menor id k (seu sucessor)
Solução ruim
Faça com que cada nó mantenha o registro de seus vizinhos e faça uma busca linear ao longo do anel
Notação
Chamamos de nó p o nó cujo identificador é p.

Ideia:
• cada no p mantém um finger table FTp[] com no máximo m entradas
FTp[i] = succ(p + 2i1)
FTp[i] aponta para o primeiro nó que sucede p com distância de pelo menos 2i1 posições
• Para procurar por uma chave k, o nó p encaminha o pedido para o nó com índice j tal que:
q = FTp[j] k < FTp[j + 1]
• Se p < k < FTp[1], a requisição é encaminhada para FTp[1]



Onde tá isso?
Ngm sabe esse demônio
(slide 9)
https://www.youtube.com/watch?v=q29szpcnorA
https://www.youtube.com/watch?v=GOOXa2GkPws