1) As diferentes alternativas são:
- Uso de dispatchers, um thread despachante deve ler requisições que entram para operações de arquivos, o servidor escolhe um thread operário ocioso e lhe entrega a requisição

Arquitetura:
- Pool de Threads ou Worker Pool: Servidor cria um número fixo de threads para processar as requisições. Quando as requisições chegam, apenas distribui o trabalho
- Thread por requisição: Thread de IO gera um novo thread para cada nova requisição. Os threads criados são destruídos após realizarem o trabalho
- Thread por conexão: Servidor cria uma nova thread quando um cliente estabelece uma conexão e destrói essa thread quando a conexão é fechada
- Thread por objeto: Servidor associa uma thread com cada objeto remoto

2) Multiplos processos

Vantagens:

- Os servidores possuem memória separada um dos outros, ou seja, a modificação do mesmo endereço de memória tornasse impossível, tornando cada um independente
- Eliminação dos race-conditions
- Garanti sincronismo
- O uso de um servidor multiplos processos impede que se escale a implementação para um sistema multiprocessado.

Desvantagens:

- É mais custoso criar outro processo que criar um thread.
- Manter estado do servidor também fica mais difícil
- Um cliente pode acessar um processo, e na outra vez outro, como a memória é separada, temos que manter sempre o mesmo cliente pro servidor.
- Sem paralelismo

Múltiplas threads:

Vantagens:

- Criação de threads envolve menos custo computacional do que criar um novo processo;
- Threads podem compartilhar recursos dentro de um processo
- Uma Thread não é protegida de outras dentro de um mesmo processo, ou seja, independentes
- Recebe a requisição e passa a tarefa para uma thread
- Uso do paralelismo

Desvantagens:

- Quando várias threads acessam o mesmo recurso, como campos estáticos, temos que tomar cuidado para que garantir que duas ou mais threads não acessem ao mesmo tempo determinado recurso, corrompendo os dados;
- O responsabilidade do programador aumenta com o aumento das Threads

3) Para ocultar a latencia temos alguns meios:

- Escalabilidade Geográfica
- Por meio do tratamento da requisição seguinte enquanto a anterior está sendo respondida
- Comunicação assincrona
- Evitar esperar por respostas a requisições remotas

a) No lado do servidor quando nos usamos multithreads acaba ficando mais facil a ocultaçao de latencia porque quando recebemos uma requisição, o servidor joga essa requisição para um thread disponivel, e caso receba outra requisição ele faz a mesma coisa e assim por diante, evitando que o sistema pare ou tenha alguma latencia

b) No lado do cliente o navegador web varre uma página HTML recebida e descobre que mais arquivos precisam ser buscados. Cada arquivo é buscado por um thread separado, cada um fazendo sua própria requisição (bloqueante) HTTP. A medida que arquivos são recebidos, o navegador os apresenta ao usuário. Basicamente com multithreads, o cliente pode fazer várias chamadas ao mesmo tempo, cada uma por meio de um thread diferente.

4) As vantagens do uso de protocolos específicos de aplicação são você poder especificar o servidor para um fim, entao qualquer requisição que seja relacionado a esse caso o servidor resolve, a desvantagem desse tipo de protocolo é a criação de varios servidores para resolver requisiçoes diferentes, não tendo algo de forma geral, gerando perda de desempenho, gaste de energia e processamento. Já as vantagens do uso de protocolos independentes da aplicação é a criação de um servidor geral para resolver requisições gerais, gerando bastante desempenho e processamento, a única desvantagem que eu vejo é que se a aplicação for muito especifica, a utilização de um protocolo independente não se encaixa muito bem, porque além de não ser necessaria, as vezes ela pode consumir muito do sistema sem necessidade e a utilização de um protocolo especifico acaba sendo melhor

5) O middleware provê da seguinte forma:

a) Transparência de Acesso: stubs do lado cliente e servidor para RPC, olhando para o lado do servidor com os stubs se o cliente manda uma requisição na linguagem c++ por exemplo e o servidor é da linguagem java, os stubs recebe essa requisição inicialmente, transforma ela para a linguagem java e manda para o servidor para que ele entenda, a mesma coisa acontece com o cliente.

b) Transparência de Localização/Migração: usa software do lado cliente e servidor(stubs) para manter registro da localização dos serviços utilizados. Usa também proxys(simula um servidor) e skeleton(simula um cliente) para garantir transparencia de localização, parecendo que o cliente e o servidor estão recebendo chamadas locais

c) Transparência de Replicação: stub cliente e servidor lida com múltiplas invocações(replicadas), entao por exemplo caso o cliente mande sua requisição para multiplos servidores, o cliente nao vai receber multiplas respostas e sim uma única dando a transparencia de replicação, a mesma coisa acontece com o servidor

d) Transparência de falhas: geralmente, só pode ser implementada no lado cliente (para mascarar falhas dos servidores ou da comunicação). Caso o servidor falhe para garantir essa transparencia de falhas podesse usar timers para medir o tempo ou receber uma resposta atraves do stub falando para tentar outro servidor

6) O servidor é com estado porque ele está mantendo os registros do status dos clientes, ele verifica que o cliente está acessando certas paginas recentemente e então mapea os endereços IP dos clientes

para as respectivas páginas, entao ele já guarda em cache. Entao mesmo ele nem recebendo a requisição do cliente ainda, ele já retorna a página registrada pro cliente

7) Servidores com estado: Armazenam informação sobre cada cliente

Vantagens:
- Operações podem ser implementadas de forma mais eficiente.
- Mensagens com pedidos podem ser menores.
- Desempenho pode ser extremamente alto, desde que os clientes possam manter cópias locais (caches).
Desvantagens:
- Recuperação após uma falha, ele tem que restaurar o servidor e não é simplesmente ligar o servidor de novo, você precisa restaurar seu estado que foi registrado
- Replicas, todos os servidores replicas precisam saber o estado do cliente, mesmo que aconteça uma falha, todos precisam saber

Aplicação: redes sociais, youtube, editor de texto

Servidores sem estado: Cada pedido deve conter toda a informação necessária para seu processamento.

Vantagens:
- Apresentam maior escalabilidade.
- A implementação de um servidor sem estado é muito mais simples que a de um servidor com estado.
- Clientes e servidores são totalmente independentes
- Reduz inconsistências de estado devido a falhas do cliente ou servidor
Desvantagens:
- Perda de desempenho porque o servidor não vai lembrar que um cliente já fez várias requisições iguais, não tem resposta pré-computada

Aplicação: incluem o Protocolo de Internet(IP), que é a base para a Internet, e o Hypertext Transfer Protocol(HTTP), que é a base da comunicação de dados para a World Wide Web.

8)a) Migração forte: Move o componente, incluindo o estado de execução, move o objeto inteiro de uma máquina para a outra e inicia um clone e o configura com o mesmo estado

b) Migração fraca: Move apenas os segmentos de código e dados(reinicia a execução), relativamente simples, requer somente que máquina alvo possa executar aquele código (torná-lo portável)

c) Avaliação remota: Um componente de um sistema que suporta mobilidade pode invocar serviços providos por outros componentes, que estão distribuídos pelos nodos da rede, através do fornecimento dos dados necessários para desenvolver o serviço bem como o código que descreve como o serviço é feito. Avaliação Remota pode ser explicada da seguinte forma: um componente A, no site A , sabe (conhecimento) como realizar uma determinada tarefa (serviço) mas não tem os recursos necessários para que ela seja feita. Estes recursos estão localizados em outro site (site B). Conseqüentemente, A envia o conhecimento de como fazer o serviço para o componente computacional localizado no site remoto. O componente B realiza o serviço utilizando os recursos nele contidos. Após a execução, há novamente uma interação entre os dois componentes para enviar o resultado da computação de volta para A.

d) Código sob demanda: O código que descreve o comportamento de um componente em um sistema que suporta mobilidade pode ser buscado em um site remoto para ser executado localmente. Exemplificando este paradigma, um componente A tem acesso aos recursos localizados no Site A mas não tem o conhecimento de como manipular estes recursos. Portanto, A interage com o componente B no Site B, requisitando o conhecimento para acessar o recurso. Uma segunda interação acontece quando B envia o conhecimento para A e este realiza a computação (abordagem contraria a Avaliação Remota).

e) Agentes móveis: onde os agentes são considerados unidades de execução, todo o componente computacional é movido para o site remoto, inclusive o seu estado, o código e alguns recursos necessários para o desenvolvimento da tarefa. Para exemplificar, um componente A é dono do conhecimento, que esta no Site A, para realizar alguns serviços. Só que em um determinado momento da execução, ele necessita de alguns recursos que estão no Site B. Então A migra para o Site B carregando o conhecimento necessário para executar uma tarefa. Ao mover-se para o site remoto, A completa o serviço neste site. Este paradigma é diferentes dos outros paradigmas, pois nestes o foco é a transferência de código entre componentes enquanto que no paradigma de agentes móveis o código e o estado da computação são movidos para o site remoto (migração).

9) Uma estrategia eficiente seria:
- A máquina alvo pode não ser apropriada para executar o código migrado
- A definição do contexto de processo/thread/processador é altamente dependente do hardware local, do sistema operacional e do sistema de tempo de execução