Resumos Exame

Resumo "t10_indirect_communication"

Resumo do Documento sobre Comunicação Indireta (t10_indirect_communication.pdf)

Introdução à Comunicação Indireta

A comunicação indireta permite a interação entre remetentes e destinatários sem um vínculo direto e simultâneo, eliminando o acoplamento rígido encontrado na comunicação síncrona.

Propriedades da Comunicação Indireta

- **Desacoplamento Espacial:** O remetente não precisa conhecer ou ter um endereço específico do destinatário.
- **Desacoplamento Temporal:** O remetente e o destinatário não precisam estar ativos ao mesmo tempo; as mensagens podem ser armazenadas temporariamente.
- **Desacoplamento de Sincronização:** Permite que as partes interajam de maneira assíncrona, sem necessidade de sincronização de tempos de execução.

Modelos de Comunicação Indireta

1. **Filas de Mensagens (Message Queues):**
   - Utilizadas para armazenar mensagens até que o destinatário esteja pronto para recebê-las.
   - Exemplos: RabbitMQ, Amazon SQS.
   - Propriedades:
     - **Persistência:** As mensagens podem ser armazenadas em disco.
     - **Fiabilidade:** Garante a entrega de mensagens.
     - **Escalabilidade:** Suporta alto volume de mensagens.
2. **Espaços de Tuplas (Tuple Spaces):**
   - Um espaço compartilhado onde as tuplas (estruturas de dados) podem ser colocadas e recuperadas.
   - Baseado no modelo de programação Linda.
   - Propriedades:

    - **Coordenação Descentralizada:** Nenhum nó específico controla a interação.
    - **Transparência:** As operações são realizadas de maneira transparente.
3. **Publicação/Assinatura (Publish/Subscribe):**
    - Os remetentes (publicadores) enviam mensagens a tópicos, e os destinatários (assinantes) recebem mensagens dos tópicos de interesse.
    - Exemplos: MQTT, Apache Kafka.
    - Propriedades:
        - **Assincronia:** Mensagens são enviadas e recebidas de maneira assíncrona.
        - **Multiponto:** Permite múltiplos destinatários para uma única mensagem.

Desafios da Comunicação Indireta

1. **Garantia de Entrega:** Assegurar que as mensagens sejam entregues mesmo em caso de falhas.
2. **Ordenação de Mensagens:** Manter a ordem correta das mensagens quando necessário.
3. **Consistência de Dados:** Garantir que todas as partes vejam os mesmos dados ou estado do sistema.
4. **Segurança:** Proteger a confidencialidade, integridade e autenticidade das mensagens.

Benefícios da Comunicação Indireta

- **Flexibilidade e Escalabilidade:** Facilita a adição e remoção de componentes do sistema sem impacto significativo.
- **Tolerância a Falhas:** Sistemas podem continuar a operar mesmo que partes dele falhem.
- **Desempenho Melhorado:** Reduz a necessidade de espera sincronizada entre remetentes e destinatários.

Exemplos de Implementações

- **RabbitMQ:** Um sistema de filas de mensagens robusto, oferecendo alta disponibilidade e persistência.
- **Apache Kafka:** Uma plataforma de streaming distribuída, otimizada para throughput e latência baixos.
- **Amazon SQS:** Um serviço de filas de mensagens na nuvem, escalável e totalmente gerenciado.

Conclusão

A comunicação indireta é uma abordagem poderosa para construir sistemas distribuídos flexíveis, escaláveis e resilientes, utilizando modelos como filas de mensagens, espaços de tuplas e publicação/assinatura. Ela oferece várias propriedades desejáveis como desacoplamento espacial e temporal, mas também apresenta desafios, especialmente em termos de garantia de entrega e consistência de dados.

Resumo "t11_http"

Comunicação Indireta (t10_indirect_communication.pdf)

Comunicação Síncrona vs. Indireta

- **Comunicação Síncrona:** Envolve interação direta entre o remetente e o destinatário, resultando em um acoplamento rígido.
- **Comunicação Indireta:** Utiliza intermediários para eliminar o acoplamento direto, permitindo desacoplamento espacial (remetente e destinatário não precisam se conhecer) e temporal (não precisam existir simultaneamente).

Aplicações Distribuídas na Internet e o Protocolo HTTP (t11_http.pdf)

Componentes Principais da Web

- **HTML (HyperText Markup Language):** Linguagem para especificar o conteúdo e layout das páginas web.
- **URLs (Uniform Resource Locators):** Identificadores de documentos e recursos na web.
- **Arquitetura Cliente-Servidor:** Baseada no protocolo HTTP (HyperText Transfer Protocol), onde clientes (navegadores) solicitam e recebem documentos e recursos de servidores web.

História da Web

- Criada por Tim Berners-Lee no CERN, Suíça, em 1989, com o objetivo de facilitar a troca de documentos entre físicos.
- Primeira proposta ignorada pelo CERN, mas continuou desenvolvendo HTTP, HTML e o primeiro navegador web.
- Em 1991, disponibilizou o navegador e o servidor web na Internet, o que levou à popularização global da web.

Protocolo HTTP

- **HTTP 1.0:** Cada conexão é fechada após a solicitação/resposta, requerendo múltiplas conexões para múltiplos objetos.
- **HTTP 1.1:** Introduz conexões persistentes, permitindo que múltiplos objetos sejam transferidos através de uma única conexão.
- **HTTP 2.0 e 3.0:** Melhorias em eficiência e performance.

Mensagens HTTP

- **Solicitações HTTP (GET, POST, HEAD):** Diferentes tipos de requisições enviadas pelos clientes.
- **Respostas HTTP (Códigos de Status):** Informações sobre o resultado da solicitação, incluindo códigos de status como 200 (OK), 404 (Not Found).

Conexões HTTP

- **Não-Persistentes:** Requerem 2 RTTs (Round-Trip Time) por objeto, aumentando o tempo total de transmissão.
- **Persistentes:** Mantêm a conexão aberta, reduzindo a sobrecarga e melhorando a eficiência.

Resumo Geral

Os documentos abordam conceitos fundamentais de comunicação indireta e a infraestrutura da web baseada no protocolo HTTP. A comunicação indireta destaca a importância do desacoplamento espacial e temporal entre remetente e destinatário, enquanto o documento sobre HTTP fornece uma visão abrangente sobre a estrutura da web, história do desenvolvimento da web, e detalhes técnicos do protocolo HTTP, incluindo suas diferentes versões e o funcionamento das conexões persistentes e não-persistentes.

Resumo "t12_servlets"

O documento é um conjunto de slides sobre aplicações web, focando principalmente em Servlets Java, uma tecnologia usada para criar aplicações web dinâmicas. Aqui está um resumo dos principais pontos abordados:

1. **Aplicações Web Baseadas em Servidores**:
    - Existem servidores web que servem páginas estáticas e outros que geram conteúdo dinâmico.

- Tecnologias como ASP.Net, PHP, Perl, Servlets & JSP, Ruby-on-Rails, Django, e TurboGears são mencionadas como opções para desenvolver aplicações web dinâmicas.

2. **Servlets Java**:
   - Um Servlet é um componente web gerenciado por um contêiner/engine que gera conteúdo dinâmico.
   - Servlets são carregados uma vez e persistem entre requisições, o que mantém o uso de memória baixo e elimina a necessidade de criar novos objetos a cada requisição.
   - São escaláveis devido ao uso de multi-threading.
3. **Arquitetura Servlet-Web Server**:
   - Servlets leem dados enviados pelo usuário, processam informações do pedido HTTP, geram e formatam resultados, definem parâmetros de resposta HTTP e enviam o documento ao cliente.
4. **Ciclo de Vida de um Servlet**:
   - Um servlet passa por um ciclo de vida que inclui carregamento da classe, instanciação, inicialização (init), serviço (service) e destruição (destroy).
   - Métodos importantes incluem doGet (para requisições GET) e doPost (para requisições POST).
5. **Processamento de Pedidos HTTP**:
   - Os métodos doGet e doPost são usados para processar pedidos HTTP.
   - Exemplo de um simples servlet que responde com a data atual usando doGet.
6. **Manipulação de Dados de Formulários**:
   - Formulários HTML podem enviar dados usando métodos GET ou POST.
   - Os dados do formulário podem ser acessados usando request.getParameter(String paramName).
7. **Respostas HTTP**:
   - Métodos como setStatus, sendError, e sendRedirect são usados para definir códigos de status HTTP e redirecionamentos.
   - Cabeçalhos de resposta podem ser definidos com setHeader.
8. **Cookies e Rastreamento de Sessão**:
   - Cookies são usados para identificar usuários durante sessões e armazenar dados persistentes.
   - Técnicas de rastreamento de sessão incluem cookies, reescrita de URL e campos de formulário ocultos.
   - A API de rastreamento de sessão permite armazenar objetos arbitrários dentro de uma sessão usando métodos como setAttribute e getAttribute.
9. **Exemplos de Código**:
   - Exemplos de código são fornecidos para ilustrar a implementação de Servlets, manipulação de dados de formulários e rastreamento de sessão.

Este resumo cobre os principais conceitos e exemplos práticos apresentados nos slides, fornecendo uma visão geral abrangente sobre a criação de aplicações web dinâmicas usando Servlets Java.

Resumo "t13_mvc_and_thymeleaf"

O documento é uma apresentação sobre a arquitetura Model-View-Controller (MVC) e o uso do Thymeleaf, um motor de templates para Java. Abaixo está um resumo dos pontos principais abordados:

Arquitetura MVC
Arquitetura de Camadas:

Apresentação (Presentation Logic): Lida com a interação do usuário e a atualização da visualização.
Aplicação (Application Logic): Processamento específico da aplicação, também chamado de lógica de negócios.
Dados (Data Logic): Armazenamento persistente da aplicação, geralmente em um sistema de gerenciamento de banco de dados.
Arquiteturas de Dois e Três Níveis:

Dois Níveis: A lógica da aplicação é dividida entre o cliente e o servidor.
Três Níveis: Cada elemento lógico é mapeado para um servidor físico separado, facilitando a separação de interface de usuário, lógica de negócios e armazenamento de dados.
Clientes Magros (Thin Clients):

Clientes Magros: Movem a complexidade para os serviços na Internet, oferecendo uma interface baseada em janelas no dispositivo do usuário, enquanto executam programas ou acessam serviços remotamente.
Padrão Arquitetural MVC:

Modelo (Model): Captura o comportamento relacionado a dados, focando na lógica de dados e regras de aplicação.
Visão (View): Representa a informação, como páginas web com gráficos, imagens e textos.
Controlador (Controller): Recebe a entrada do usuário, envia comandos para o modelo e decide qual visão apresentar.
Vantagens do MVC:

Separação das preocupações de implementação.
Redução de duplicação de código.

Facilita a manutenção e atualização.
Simplificação dos testes de unidades de software.
Thymeleaf
Thymeleaf:

Motor de templates moderno para ambientes web e standalone.
Objetivo principal: trazer templates naturais elegantes que podem ser exibidos corretamente em navegadores e funcionar como protótipos estáticos.
Templates Naturais:

Templates HTML escritos em Thymeleaf continuam a funcionar como HTML, permitindo que os templates usados na aplicação funcionem também como artefatos de design úteis.
História e Uso:

Thymeleaf é um motor de templates para Java que pode ser usado tanto em ambientes web (baseados em servlets) quanto em ambientes não-web.
Não é um framework web, mas um motor de templates que processa HTML, XML, texto, JavaScript e CSS.
FAQs:

Pode substituir completamente JSP e JSTL.
Pode ser usado fora de aplicações web.
Integração com Spring é opcional.
Dialetos:

Thymeleaf é extensível e permite a definição e personalização detalhada do processamento dos templates.
O núcleo do Thymeleaf fornece um dialeto chamado Standard Dialect, adequado para a maioria dos usuários.
Processadores de Atributos:

A maioria dos processadores do Standard Dialect são processadores de atributos, permitindo que os arquivos de template HTML sejam exibidos corretamente nos navegadores mesmo antes do processamento.
Este resumo cobre os conceitos principais e as funcionalidades do Thymeleaf e da arquitetura MVC, conforme descritos nos slides do documento.](<O documento é um conjunto de slides sobre aplicações web, focando principalmente em Servlets Java, uma tecnologia usada para criar aplicações web dinâmicas. Aqui está um resumo dos principais pontos abordados:

1. **Aplicações Web Baseadas em Servidores**:

- Existem servidores web que servem páginas estáticas e outros que geram conteúdo dinâmico.
- Tecnologias como ASP.Net, PHP, Perl, Servlets & JSP, Ruby-on-Rails, Django, e TurboGears são mencionadas como opções para desenvolver aplicações web dinâmicas.

2. **Servlets Java**:
   - Um Servlet é um componente web gerenciado por um contêiner/engine que gera conteúdo dinâmico.
   - Servlets são carregados uma vez e persistem entre requisições, o que mantém o uso de memória baixo e elimina a necessidade de criar novos objetos a cada requisição.
   - São escaláveis devido ao uso de multi-threading.
3. **Arquitetura Servlet-Web Server**:
   - Servlets leem dados enviados pelo usuário, processam informações do pedido HTTP, geram e formatam resultados, definem parâmetros de resposta HTTP e enviam o documento ao cliente.
4. **Ciclo de Vida de um Servlet**:
   - Um servlet passa por um ciclo de vida que inclui carregamento da classe, instanciação, inicialização (init), serviço (service) e destruição (destroy).
   - Métodos importantes incluem doGet (para requisições GET) e doPost (para requisições POST).
5. **Processamento de Pedidos HTTP**:
   - Os métodos doGet e doPost são usados para processar pedidos HTTP.
   - Exemplo de um simples servlet que responde com a data atual usando doGet.
6. **Manipulação de Dados de Formulários**:
   - Formulários HTML podem enviar dados usando métodos GET ou POST.
   - Os dados do formulário podem ser acessados usando request.getParameter(String paramName).
7. **Respostas HTTP**:
   - Métodos como setStatus, sendError, e sendRedirect são usados para definir códigos de status HTTP e redirecionamentos.
   - Cabeçalhos de resposta podem ser definidos com setHeader.
8. **Cookies e Rastreamento de Sessão**:
   - Cookies são usados para identificar usuários durante sessões e armazenar dados persistentes.
   - Técnicas de rastreamento de sessão incluem cookies, reescrita de URL e campos de formulário ocultos.
   - A API de rastreamento de sessão permite armazenar objetos arbitrários dentro de uma sessão usando métodos como setAttribute e getAttribute.
9. **Exemplos de Código**:

- Exemplos de código são fornecidos para ilustrar a implementação de Servlets, manipulação de dados de formulários e rastreamento de sessão.

Este resumo cobre os principais conceitos e exemplos práticos apresentados nos slides, fornecendo uma visão geral abrangente sobre a criação de aplicações web dinâmicas usando Servlets Java.>)

Resumo de "t14_springboot"

Resumo de "t15_security.pdf"

O documento aborda a segurança em sistemas distribuídos, identificando riscos e requisitos de segurança, bem como os mecanismos e algoritmos usados para mitigar esses riscos. Aqui estão os pontos principais:

Riscos de Segurança

- **Riscos para Cliente e Servidor**:
  - Escuta clandestina (eavesdropping)
  - Manipulação maliciosa (tampering)
  - Supressão de mensagens
  - Reprodução de mensagens (message replaying)
- **Riscos específicos para o Cliente**:
  - Conteúdo ativo ilegítimo
  - Violação de integridade
  - Ataques à privacidade
  - Personificação (masquerading)
- **Riscos específicos para o Servidor**:
  - Falsificação de servidor (webjacking)
  - Invasão de servidor (backdoors, cavalos de Troia, ataques internos)
  - Ataques de negação de serviço (DoS)

Requisitos de Segurança

- **Confidencialidade**: Proteção contra divulgação não autorizada.
- **Integridade**: Garantia de consistência dos dados.
- **Autenticação**: Confirmação da identidade.
- **Não-repúdio**: Garantia de que o emissor não possa negar a autoria da comunicação.
- **Disponibilidade**: Acesso garantido aos usuários legítimos.

- **Controle de Acesso**: Restrição de acesso a usuários não autorizados.

Mecanismos de Segurança

- **Criptografia**: Usada para confidencialidade e pode prover autenticação e integridade.
- **Assinaturas Digitais**: Usadas para autenticação, integridade e não-repúdio.
- **Algoritmos de Checksum/Hash**: Usados para integridade e autenticação.

Algoritmos de Criptografia

- **Criptografia Simétrica**:
  - Algoritmos como TEA, DES, 3DES, IDEA, AES.
  - Problema: Compartilhamento seguro da chave simétrica entre partes anônimas.
  - Solução: Uso de um Centro de Distribuição de Chaves (Key Distribution Center).
- **Criptografia Assimétrica (Chave Pública/Privada)**:
  - Cada parte tem uma chave pública e uma chave privada.
  - Exemplo de algoritmos: RSA, DSA.

Ataques e Contra-Medidas

- **Criptoanálise**: Tentativas de decifrar mensagens sem chave.
- **Ataques de Força Bruta**: Testar todas as chaves possíveis.
- **Bugs no Software de Criptografia**: Exploração de vulnerabilidades no software.
- **Quebra do Sistema**: Obtenção da chave privada ou do texto claro.

Tamanho das Chaves Privadas

- **Chave de 40 bits**: Facilmente quebrada por ataques distribuídos de força bruta.
- **Chave de 128 bits**: Considerada segura atualmente, com um número de possibilidades maior que o número de moléculas nos oceanos.

Este documento fornece uma visão abrangente dos desafios e soluções na segurança de sistemas distribuídos, focando em criptografia, autenticação e os mecanismos usados para proteger a integridade e a confidencialidade dos dados (t15_security).

Resumo "t16_rest"

O documento fornece uma visão geral sobre serviços web RESTful (Representational State Transfer) e compara diferentes abordagens de serviços web, como RPC, SOAP e REST. Aqui estão os principais pontos:

Serviços Web

- Exemplos de serviços web incluem APIs para dados (The Guardian), atividade social (Facebook), transferências bancárias (Paypal), SMS (Twilio), envio (Fedex, UPS), computação em nuvem (Amazon EC2) e tarefas humanas (Amazon Mechanical Turk).
- Tipos de serviços web:
  - **RPC/RMI**: Utilizado dentro da mesma rede/empresa, requer uma interface pre-negociada (ex: Corba, Java RMI).
  - **SOAP**: Evolução do XML-RPC, encapsula carga útil XML dentro de solicitações/respostas XML, usa WSDL para descrever serviços.
  - **REST**: Usa HTTP como mecanismo de transporte, URLs definem recursos e verbos HTTP descrevem operações.

Filosofia do Design REST

- REST é uma coleção de princípios arquitetônicos para implementar aplicações web.
- Enfatiza a simplicidade e reutilização dos serviços web.
- Diferentes verbos HTTP (GET, PUT, POST, DELETE) são usados para interagir com recursos definidos por URLs.

Exemplos de Operações HTTP em REST

- **GET**: Recupera recursos.
- **PUT**: Cria ou substitui recursos.
- **POST**: Modifica recursos.
- **DELETE**: Exclui recursos.

Comparação REST vs SOAP

- **REST**: Melhor para serviços sem estado, baixa sobrecarga de serialização, mais escalável e eficiente em termos de largura de banda.
- **SOAP**: Inclui descrições WSDL, bom para automação de serviços web corporativos, suporta processamento assíncrono.

Callbacks em REST

- Utiliza Web Hooks onde o cliente informa ao servidor um recurso para ser chamado de volta.

OAuth

- Protocolo de autorização para serviços web que permite que um usuário conceda acesso limitado a suas informações sem compartilhar suas credenciais.
- Benefícios do OAuth incluem proteção contra roubo de senhas, possibilidade de revogar acesso de aplicativos e uso de permissões parciais.

Cenários Reais

- Exemplos de problemas de segurança e privacidade, como vazamento de senhas e uso indevido de dados pessoais.

Exemplos de OAuth na Prática

- Implementado por grandes plataformas como Dropbox, Facebook, GitHub, Google, Foursquare, SalesForce, Citrix, Twitter e Windows Live.

Essa visão geral resume os principais conceitos, operações e comparações envolvendo serviços web RESTful, destacando a importância da simplicidade, eficiência e segurança nas implementações de serviços web.

Resumo "t17_websockets"

O documento aborda o protocolo WebSocket, detalhando sua implementação, vantagens e comparação com métodos tradicionais de comunicação em tempo real na web. Aqui estão os pontos principais:

Comunicação em Tempo Real na Web

- **Desafios Anteriores**: Anteriormente, a comunicação bidirecional em tempo real exigia polling HTTP, causando problemas de sobrecarga de cabeçalhos, manutenção de conexões no servidor e mapeamento no cliente.
- **Soluções Antigas**:
  - **Polling**: Cliente faz requisições periódicas ao servidor.
  - **Long Polling**: O servidor mantém a conexão aberta até haver dados para enviar.
  - **Streaming**: O servidor mantém a conexão aberta indefinidamente.

WebSockets

- **Protocolo WebSocket**: Permite comunicação bidirecional através de uma única conexão TCP, superando as limitações das soluções antigas.
- **Handshake Inicial**: Processo de estabelecimento de conexão, onde o cliente e o servidor trocam cabeçalhos HTTP para atualizar o protocolo para WebSocket.

Filosofia de Design

- WebSocket é construído sobre TCP, adicionando nomeação, enquadramento, handshakes e endereçamento.
- A única relação com HTTP é o handshake inicial.
- Utiliza as mesmas portas que HTTP (80 e 443 para TLS).

Implementação de WebSockets

- **Servidor**: Uso da anotação `@ServerEndpoint` em Java para declarar pontos de extremidade WebSocket.
- **Eventos**:
  - `@OnOpen`: Método chamado quando uma conexão é aberta.
  - `@OnClose`: Método chamado quando uma conexão é fechada.
  - `@OnMessage`: Método chamado quando uma mensagem é recebida.

Envio de Mensagens

- **Interface `Session`**: Método `getBasicRemote()` retorna um objeto `RemoteEndpoint` para enviar mensagens ao cliente.

Implementação do Cliente (HTML + JavaScript)

- **HTML**: Estrutura básica da página com JavaScript para manipulação de WebSocket.
- **JavaScript**:
  - Conexão com WebSocket.
  - Manipulação de eventos (abertura, fechamento, mensagem, erro).
  - Envio de mensagens e atualização da interface do usuário.

Exemplo Completo de `ServerEndpoint`

- Código detalhado para um servidor de chat WebSocket em Java, incluindo

gerenciamento de conexões e transmissão de mensagens.

Bibliografia

- Referências para o protocolo WebSocket (RFC6455) e API WebSocket (W3C).

Este resumo cobre os aspectos essenciais do documento sobre WebSockets, enfatizando sua importância para a comunicação em tempo real na web e fornecendo detalhes de implementação tanto no servidor quanto no cliente.

Resumo "t18_distributed_file_systems"

Resumo do PDF sobre Sistemas de Arquivos Distribuídos

Capítulo 8 do Livro de George Coulouris

Sistemas de Arquivos Locais: Módulos

- **Operações no Sistema de Arquivos UNIX**:
  - `open(name, mode)`, `creat(name, mode)`: Abre ou cria um arquivo, retornando um descritor de arquivo.
  - `close(filedes)`: Fecha o arquivo aberto.
  - `read(filedes, buffer, n)`, `write(filedes, buffer, n)`: Transfere bytes entre o arquivo e o buffer.
  - `lseek(filedes, offset, whence)`: Move o ponteiro de leitura/escrita.
  - `unlink(name)`: Remove um nome de arquivo do diretório.
  - `link(name1, name2)`: Adiciona um novo nome a um arquivo existente.
  - `stat(name, buffer)`: Obtém os atributos do arquivo.
- **Estrutura dos Atributos de Arquivo**:
  - Comprimento do arquivo, timestamps (criação, leitura, escrita, atributo), contador de referências, proprietário, tipo de arquivo, lista de controle de acesso.

Sistemas de Arquivos Distribuídos

- Permitem que programas armazenem e acessem arquivos remotos da mesma forma que arquivos locais.
- **Requisitos**:
  - **Transparência**:
    - **Acesso**: Interface única para arquivos locais e distribuídos.

        - **Localização**: Espaço de nomes uniforme e independente de localização.
        - **Mobilidade**: Especificações de arquivo invariáveis, mesmo que movidos fisicamente.
        - **Desempenho**: Manutenção do desempenho com aumento de carga.
        - **Escalabilidade**: Crescimento incremental.
    - **Acesso concorrente**.
    - **Replicação de arquivos**.
    - **Tolerância a falhas**: Serviço correto na presença de falhas de comunicação ou servidor.
    - **Consistência**: Atualização de cópia única, garantindo que todos os clientes vejam o conteúdo idêntico do arquivo.
    - **Segurança**: Controle de acesso e autenticação de clientes.
    - **Eficiência**: Latência de acesso e escalabilidade.

Arquitetura do Serviço de Arquivos Distribuídos

- **Serviço de Arquivo Plano**:
    - Realiza operações de arquivo usando identificadores únicos de arquivo (UFIDs).
    - Interface baseada em RPC para operações de arquivo, normalmente não usada por programas de nível de aplicação.
- **Serviço de Diretório**:
    - Mapeia UFIDs para nomes de arquivos de texto e vice-versa.
- **Módulo Cliente**:
    - Fornece API para operações de arquivo disponíveis para programas de aplicação.

Operações no Serviço de Arquivo Plano

- **Leitura e Escrita**:
    - `Read(FileId, i, n)`: Lê até n itens de um arquivo a partir do item i.
    - `Write(FileId, i, Data)`: Escreve uma sequência de dados em um arquivo, começando no item i.
- **Criação e Exclusão**:
    - `Create() -> FileId`: Cria um novo arquivo de comprimento 0.
    - `Delete(FileId)`: Remove o arquivo do armazenamento.
- **Atributos**:
    - `GetAttributes(FileId) -> Attr`: Retorna os atributos do arquivo.
    - `SetAttributes(FileId, Attr)`: Define os atributos do arquivo.

Operações no Serviço de Diretório

- **Busca e Manipulação de Nomes**:
  - `Lookup(Dir, Name) -> FileId`: Localiza um nome de texto no diretório.
  - `AddName(Dir, Name, File)`: Adiciona um nome ao diretório.
  - `UnName(Dir, Name)`: Remove um nome do diretório.
  - `GetNames(Dir, Pattern) -> NameSeq`: Retorna todos os nomes de texto no diretório que correspondem ao padrão.

Sistema de Arquivos Distribuído: NFS (Network File System)

- **Histórico e Protocolos**:
  - Criado pela Sun em 1985, com várias versões (v1 a v4) usando UDP e TCP.
- **Objetivos e Recursos**:
  - Transparência, portabilidade entre sistemas operacionais, escalabilidade, recuperação rápida de falhas, cache local, independência de protocolo de rede e suporte a segurança.
- **Operações do Servidor NFS**:
  - Incluem operações como `lookup`, `create`, `remove`, `getattr`, `setattr`, `read`, `write`, `rename`, `link`, `symlink`, `readlink`, `mkdir`, `rmdir`, `readdir`, `statfs`.
- **Controle de Acesso e Autenticação**:
  - Uso de RPCs para transmissão de solicitações NFS, com informações de autenticação verificadas contra permissões de acesso.

Desempenho e Cache

- **Cache no Servidor e Cliente**:
  - **Servidor**: Cache de disco e operações de escrita através de cache (Write-through e Write-gathering).
  - **Cliente**: Cache com políticas de escrita atrasada (Delayed-write) e escrita ao fechar (Write-on-close).
- **Otimização do Cache do Cliente**:
  - Redução de tráfego no servidor e algoritmos adaptativos para ajustar tempos de cache.
- **Preservação da Integridade dos Arquivos**:
  - Uso de bloqueio explícito de arquivos e consistência de cache para garantir a integridade dos dados.

Comparação NFS v3 vs. NFS v4

- **NFS v4**:
  - Introduz pedidos compostos, delegações, bloqueios integrados no protocolo, servidor com estado e mecanismos de callback para delegações de arquivos.

Este resumo abrange os principais pontos discutidos no documento, fornecendo uma visão geral das operações, arquitetura, requisitos e desempenho dos sistemas de arquivos distribuídos, com foco particular no NFS.

Resumo "t19_distributed_file_systems"

Nomes
• Identificação de objetos: Nomes são usados para identificar recursos para compartilhamento e comunicação, como os nomes de domínio da Internet e endereços
de e-mail.
Serviços de Nome vs Serviços de Diretório
• Serviços de Nome: Associam nomes a atributos, tipicamente endereços de rede.
• Serviços de Diretório: Permitem consultas baseadas em atributos, além de associar
nomes a atributos.
Serviços de Nome e Diretório
• DNS (Domain Name System): Mapeia nomes de domínio para atributos de um computador, como endereços IP.
• X.500: Serviço de diretório que mapeia nomes de pessoas para um conjunto de atributos.
• RMI Registry/CORBA Naming Service: Mapeia nomes de objetos remotos para suas referências.
• LDAP (Lightweight Directory Access Protocol): Implementação leve do X.500 usada em aplicações intranet.
• JINI: Serviço de descoberta para redes espontâneas.
• UDDI (Universal Description, Discovery, and Integration): Diretório de serviços Web.
Identificadores Uniformes de Recursos (URIs)
• URLs (Uniform Resource Locators): Endereços de recursos Web.
• URNs (Uniform Resource Names): Persistem mesmo se o recurso for movido, proporcionando modos mais ricos de encontrar recursos na Web.
Navegação e Resolução de Nomes
• Navegação Iterativa: O cliente contata servidores de nomes iterativamente até resolver o nome.
• Navegação Controlada por Servidor: Pode ser recursiva ou não recursiva, onde o servidor contata pares se não puder resolver o nome.
• Navegação Multicast: Apenas o servidor que detém os atributos nomeados

responde
à requisição.
DNS: Sistema de Nomes de Domínio
• História e Estrutura: DNS foi apresentado em 1987, substituindo um esquema de nomes centralizado que não escalava bem.
• Servidores de Nomes: Divididos em servidores locais e autoritativos, evitando centralização por motivos de falha única e volume de tráfego.
Particionamento, Replicação e Caching
• Particionamento: Dados DNS são particionados por domínio.
• Replicação: Melhora disponibilidade e desempenho com múltiplos servidores por domínio.
• Caching: Servidores podem armazenar resultados de resoluções de nome para melhorar eficiência.
DNS: Consultas e Navegação
• Resolução de Nomes: Traduz nomes simbólicos de host em endereços IP.
• Exemplo de Consulta: Um host pode contactar o servidor DNS local, que por sua vez contacta servidores raiz e autoritativos conforme necessário.
Serviços de Diretório
• Fundamentos do LDAP: Protocolo leve para acesso a diretórios, com suporte para operações de busca e modificação de entradas.
Serviços de Descoberta
• JINI: Registra serviços em redes espontâneas, com operações de registro e busca.
• UDDI: Diretório para integração de serviços Web, permitindo buscas por empresas e
serviços expostos na Web.

Resumo "t20_peer2peer"

Categorias de Sistemas P2P
• P2P File Sharing: Napster, Gnutella, KaZaA, eDonkey, BitTorrent.
• P2P Communication: Mensagens instantâneas e VoIP, como Skype.
• P2P Computation: Projetos como SETI@home e PlanetLab.
• DHTs (Distributed Hash Tables) e suas aplicações: Chord, CAN, Pastry, Tapestry.
Questões Chave para Sistemas P2P
• Join/Leave: Como os nós entram e saem? Quem é permitido?
• Busca e Recuperação: Como encontrar conteúdo? Como são construídos, armazenados e distribuídos os índices de metadados?
• Distribuição de Conteúdo: Onde o conteúdo é armazenado? Como é baixado e recuperado?
Primitivas P2P
• Join: Como entrar/sair do sistema P2P?
• Publish: Como anunciar um arquivo?
• Search: Como encontrar um arquivo?

• Fetch: Como baixar um arquivo?
Estratégias de Publicação e Busca
• Centralizado (Napster): Cliente contata servidor central para todas as operações.
• Flood the Query (Gnutella): Inunda a rede com consultas.
• Flood the Index (PlanetP): Distribui índices pela rede.
• Route the Query (Chord): Roteia consultas de forma estruturada.
P2P com Diretório Centralizado
• Problemas: Ponto único de falha, gargalo de desempenho e questões legais.
• Exemplo: Napster, onde o diretório é centralizado, mas a transferência de arquivos é descentralizada.
Redes Overlay
• Unstructured Overlays: Nós escolhem vizinhos aleatoriamente (ex: Gnutella, KaZaA, BitTorrent, Skype).
• Structured Overlays: Nós são organizados em uma estrutura restritiva (ex: Pastry, CAN, Chord, Tapestry).
Gnutella: P2P não Estruturado
• Query Flooding: Clientes contatam alguns nós vizinhos e propagam consultas.
• Descentralização: Difícil de desligar, mas inunda a rede com mensagens de consulta.
KaZaA: P2P com Super Nodes
• Super Nodes: Nós especiais que gerenciam consultas e armazenamento.
• Query Flooding Inteligente: Supernodes propagam consultas entre si.
• Paralelização de Downloads: Arquivos podem ser baixados de múltiplos pares simultaneamente.
BitTorrent: Foco em Distribuição Eficiente
• Swarming: Clientes baixam pedaços de arquivos de múltiplos peers.
• Tit-for-tat: Incentiva trocas justas, priorizando uploaders rápidos.
• Trackers: Servidores que mantêm listas de peers ativos para cada arquivo.
Structured P2P: DHTs
• DHTs: Redes estruturadas baseadas em endereçamento (ex: Chord, CAN, Pastry, Tapestry).
• Chord: Peers organizados em um anel, cada peer conhece seus sucessores e precursores.
• Finger Tables: Melhoram a eficiência da busca, reduzindo o número de saltos necessários.

Resumo "t21_blockchain"

Características de uma Moeda
• Meio de trocas comerciais: Utilizada para pagamentos.
• Unidade convertível de conta: Define preços de bens e serviços.
• Armazenar valor ao longo do tempo: Mantém seu valor.
Moedas Digitais (Criptomoedas)

• Surgiram várias criptomoedas na última década, como bitcoin, litecoin, worldcoin, zetacoin, novacoin.
• Descentralização: Não usam uma entidade central.
• Pagamentos digitais: Principal uso.
Bitcoin
• Dinheiro eletrônico peer-to-peer: Sistema de software para pagamentos eletrônicos sem entidade administrativa.
• Transações diretas: Elimina custo de intermediários, mas também a capacidade de
mediação.
Exemplo de Pagamento com Bitcoin
• Alice compra um item na loja de Bob usando bitcoin.
• Um QR code é gerado com informações da transação.
• Alice usa sua carteira digital para ler o código e autorizar o pagamento.
Problema de Double Spending
• Solução Centralizada: Uso de uma autoridade central (como um banco).
• Solução Descentralizada: Transações são anunciadas publicamente e verificadas por
toda a rede.
Blockchain

1. Registo de Transações: Todas as transações são registradas e agrupadas em blocos.
2. Cadeia de Blocos: Cada bloco inclui o hash do bloco anterior, formando uma cadeia.
3. Proof-of-Work: Criação de blocos exige uma operação computacionalmente demorada, impedindo alterações retroativas na cadeia.
   Carteiras de Bitcoin
   • Armazenamento: Bitcoins são armazenadas em carteiras digitais (software ou hardware).
   • Chave Pública e Privada: Cada carteira gera um par de chaves; a chave pública é
   usada como endereço bitcoin, e a chave privada é usada para acessar os fundos.
   Transações Eletrônicas• Cadeia de Assinaturas Digitais: Cada transação inclui a assinatura digital da
   transação anterior e a chave pública do beneficiário.
   • Verificação: Confirmação da cadeia de proprietários.
   Rede Bitcoin
4. Nova transação é transmitida a todos os nós.
5. Nós agrupam transações em blocos.
6. Cada nó realiza proof-of-work.

7. Prova de trabalho é transmitida a todos os nós.
8. Blocos são aceitos se todas as transações forem válidas.
9. Bloco aceito se torna o predecessor do próximo bloco.
10. A cadeia mais longa é considerada correta.

Incentivo à Mineração

• Criação de Novas Moedas: Primeira transação de um bloco gera uma nova moeda
para o criador do bloco.

• Dificuldade Progressiva: Mineração torna-se progressivamente mais difícil, mantendo o intervalo de 10 minutos entre blocos.

• Redução de Recompensa: Recompensa reduzida pela metade a cada 210.000 blocos,
terminando em 2140 com 21 milhões de bitcoins.