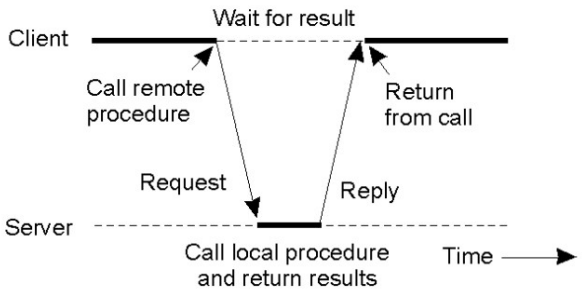
Conceitos Básicos

Berkeley Sockets

- A API de sockets permite que os desenvolvedores criem aplicativos de rede que podem enviar e receber dados através de conexões de rede, como TCP (Transmission Control Protocol) e UDP (User Datagram Protocol). Ela oferece uma maneira padronizada de lidar com a comunicação em rede, independentemente do sistema operacional ou da tecnologia de rede subjacente.
- Sockets
    - Processo de comunicação que permite dois diferentes processos de conversarem e trocarem informação entre si.
    - Os sockets funcionam a partir da arquitetura Client/Server
    - Comunicação entre processos (programas) que estão em execução em diferentes computadores, geralmente através de uma rede.
        - Sockets TCP (Transmission Control Protocol): Esses sockets fornecem uma comunicação orientada à conexão, o que significa que uma conexão é estabelecida entre os processos antes que eles possam trocar dados.
        - Sockets UDP (User Datagram Protocol): Os sockets UDP oferecem uma forma mais simples de comunicação, mas não garantem a entrega dos dados nem a ordem em que eles chegam.

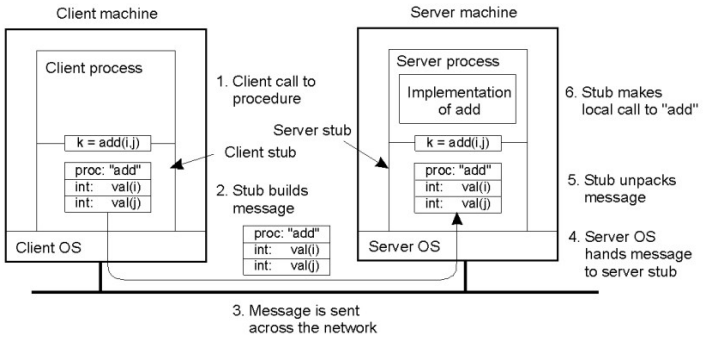
RPC – Remote procedure call?

- **RPC (Remote Procedure Call)** é um protocolo de comunicação que permite que um programa em um computador solicite a execução de um procedimento (função ou método) em um computador remoto, como se fosse uma chamada de função local. Isso permite que programas distribuídos cooperem e compartilhem funcionalidades de forma transparente, como se estivessem em um único sistema.



- O RPC é usado em programação para sistemas paralelos e distribuídos para facilitar a comunicação e a colaboração entre diferentes partes de um sistema distribuído.
- RPC envolve o conceito de stubs nos lados cliente e servidor
    - Os stubs são componentes de código que atuam como **representantes locais das funções remotas em um sistema distribuído**. No lado do cliente, o stub parece ser uma função local que o programa pode chamar, mas ele **cuida da comunicação com o servidor remoto**. No lado do servidor, o stub **recebe a solicitação do cliente, a traduz em uma chamada de função real no servidor e, em seguida, passa os resultados de volta ao cliente por meio da rede**. Isso torna a chamada de procedimento remoto transparente para o programador, simplificando o desenvolvimento de aplicativos distribuídos.

- RPC (Remote Procedure Call) gerencia a passagem de parâmetros entre funções em hosts diferentes de forma transparente para o programador. Os passos envolvidos na computação usando RPC incluem:



    - **Chamada de Procedimento Remoto**: O cliente chama uma função remota como se fosse uma função local.
    - **Conversão de Parâmetros**: Os parâmetros da chamada são convertidos em uma representação que pode ser transmitida pela rede.
    - **Transmissão**: Os parâmetros são enviados para o servidor por meio da rede.
    - **Execução Remota**: No servidor, a função remota é executada com base nos parâmetros recebidos.
    - **Retorno**: Os resultados são convertidos em uma representação adequada para a rede e enviados de volta ao cliente.
    - **Recepção e Retorno Local**: O cliente recebe os resultados e os apresenta como se fossem retornados por uma função local.
- RPC (Remote Procedure Call) lida com a passagem de parâmetros entre funções em hosts distintos, garantindo que os dados sejam transmitidos e interpretados corretamente, independentemente da arquitetura do hardware.
    - **Mensagem original em um Pentium**: A mensagem é empacotada com base na arquitetura do Pentium, que usa seu próprio formato de representação de dados.
    - **Mensagem após ser recebida em um host SPARC**: Quando a mensagem chega a um host SPARC, o RPC a traduzirá para o formato SPARC apropriado para que os dados possam ser compreendidos e processados pelo hardware SPARC.
    - **A mensagem após ser invertida**: Durante o processo de tradução, os bytes da mensagem podem ser reorganizados para coincidir com a ordem de bytes do host de destino. Isso é conhecido como inversão de bytes ou "byte swapping". Os números menores nas caixas indicam o endereço de cada byte após a reorganização.
    -

Big Data

Definição

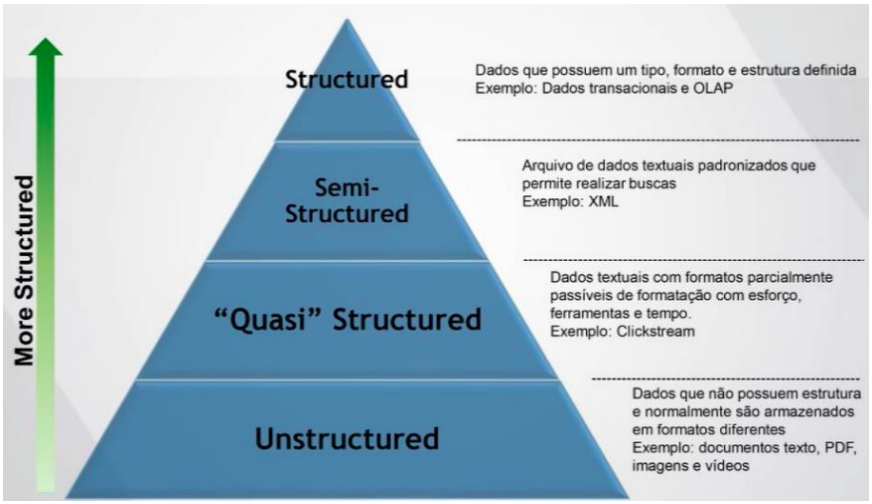
- Volume gigantesco de dados de várias fontes.
- Dados de várias fontes
- Camada de decisão
- Dados sobre pistas/rastros deixados pelo usuário

        - Que pistas/rastros são esses?
            - Viagens no fim de semana, próximo MBA, o que fez, onde andou, etc. (esses sistemas têm esquemas de geolocalização)
- **Dados de sensores – mundo IoT**
    - Dados de sensores em IoT (Internet das Coisas) são informações coletadas por sensores físicos ou dispositivos conectados à Internet que monitoram o ambiente físico. Esses dados podem incluir informações sobre temperatura, umidade, pressão, movimento, luz, som.
- **Dados mais genéricos**
    - Comportamento de ações da bolsa de valores, e outros contextos que não tratam do indivíduo, mas de processos que geram dados o tempo todo. (empresa e outros contextos mais genéricos)

Porque esses dados são importantes?

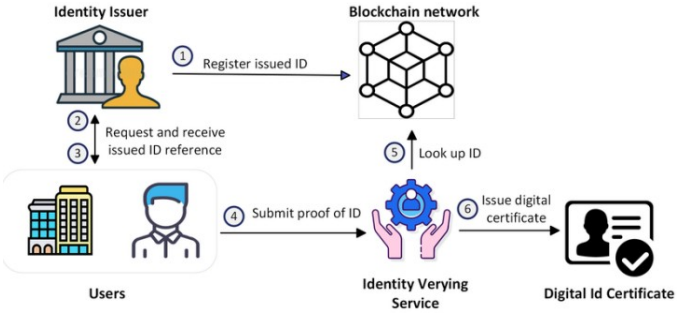
- Se bem trabalhados, podem gerar valor trazer **insights** para tomada de decisão e aquisição de conhecimento em contextos diversos. Exemplos práticos:
    - Onde investir, no caso de uma empresa
    - Gestão de cidades (smart cities) – aqui muitos dados cruzados permitem melhoria de serviços
    - Pesquisas focadas em dados e não em amostras – muda a trajetória das pesquisas

Como são os dados no Big Data?



- **Dados estruturados**
    - Esquema estruturado, dados associados a um modelo de dados
    - Informações organizadas de forma predefinida e padronizada. Isso significa que os dados são armazenados de maneira que a estrutura deles seja conhecida e consistente. Por exemplo, um banco de dados relacional contém dados estruturados, onde as informações são armazenadas em tabelas com colunas definidas.

Identidade digital



-
- Explica:
-
- Identiy User
- 1 - Register issued ID
- 2 - Request and recive
- 3 - issued ID reference
-
- Users
- 4 - Submit proof of ID
-
- Identity
-

IoT e Big Data

- **Semelhanças**
  - Ambos IoT (Internet das Coisas) e Big Data envolvem a coleta, processamento e análise de grandes volumes de dados. IoT gera dados de sensores em tempo real, enquanto Big Data lida com o armazenamento e análise eficientes desses dados em grande escala.
- **Diferença**
  - IoT refere-se à rede de dispositivos conectados que geram dados em tempo real a partir de sensores físicos. Big Data é uma disciplina que lida com a gestão, armazenamento e análise de dados em grande escala, independentemente de sua origem. Em resumo, IoT é a fonte de dados, enquanto Big Data é a infraestrutura e as técnicas para lidar com esses dados. Ambos são frequentemente usados juntos para extrair insights e tomar decisões baseadas em dados em tempo real.

Página 27

Mudança na estratégias de escalabilidade

- **Escalabilidade vertical (scale up)**
  - Envolve aumentar a capacidade de um sistema ao adicionar mais recursos a uma única máquina, como CPU, RAM ou armazenamento. Isso geralmente tem limites físicos e pode ser mais caro. É adequado para cargas de trabalho que podem ser acomodadas em uma única máquina poderosa.
  - **Pontos fortes**
    1. Desempenho Individual: Máquinas maiores geralmente têm mais poder de processamento, memória e recursos, o que pode resultar

em um desempenho superior para tarefas que exigem muitos recursos.
2. Simplicidade de Gerenciamento: Gerenciar uma única máquina é mais simples do que gerenciar várias máquinas em um cluster.
3. Escalabilidade Limitada: Embora tenha limites físicos, a escalabilidade vertical pode ser uma solução rápida e direta para melhorar o desempenho quando os recursos adicionais de uma única máquina são suficientes.

- **Pontos fracos**
  - Custo Elevado: Atualizar uma máquina com componentes de alta qualidade pode ser caro.
  - Ponto Único de Falha: Se a máquina falhar, todo o sistema pode ficar inoperante.

- **Escalabilidade horizontal (scale out)**
  - Envolve aumentar a capacidade de um sistema distribuído adicionando mais máquinas ao cluster. Isso é altamente escalável e adequado para cargas de trabalho intensivas que podem ser distribuídas entre várias máquinas. É mais flexível e eficaz em termos de custo para lidar com cargas de trabalho em constante crescimento.
    - **Pontos fortes**
      1. Alta Escalabilidade: Adicionar mais máquinas a um cluster permite lidar com cargas de trabalho em constante crescimento de forma mais eficaz.
      2. Tolerância a Falhas: Um cluster distribuído pode ser mais robusto, pois não depende de uma única máquina; se uma máquina falhar, outras podem continuar operando.
      3. Custo-Efetividade: Em muitos casos, é mais econômico adicionar servidores menos potentes em vez de comprar uma máquina cara.
    - Pontos fracos
      1. Complexidade de Gerenciamento: Gerenciar um cluster distribuído pode ser complexo e requer ferramentas e práticas específicas.
      2. Overhead de Comunicação: A comunicação entre máquinas em um cluster pode introduzir um overhead de rede que precisa ser gerenciado.
      3. Dificuldade de Escala em Aplicações Legadas: Nem todas as aplicações são facilmente escaláveis horizontalmente; algumas podem exigir revisões significativas.