UNISENAI JOINVILLE
TECNÓLOGO EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MURILO CÉSAR FERREIRA

DESENVOLVIMENTO DE SISTEMAS DISTRIBUÍDOS

JOINVILLE
2025

Resumo

Este trabalho apresenta uma análise aprofundada sobre os sistemas distribuídos, explorando suas origens, características fundamentais e aplicações contemporâneas. Iniciando com uma contextualização histórica, o texto define sistemas distribuídos como ambientes computacionais formados por múltiplos nós interligados que cooperam para prover serviços de forma integrada e transparente ao usuário. As principais características discutidas incluem transparência (de localização, replicação, acesso, falhas, etc.), escalabilidade, heterogeneidade, confiabilidade e tolerância a falhas.

A comparação entre sistemas distribuídos e centralizados evidencia as vantagens dos primeiros em termos de escalabilidade, disponibilidade e resiliência, apesar da maior complexidade de implementação e manutenção. O trabalho também descreve diferentes modelos arquiteturais, como microserviços, cliente-servidor, peer-to-peer, sistemas baseados em agentes e computação em grade, além das tecnologias que os sustentam — como containers, bancos de dados distribuídos, sistemas de arquivos e frameworks de processamento paralelo.

Em seguida, são analisadas aplicações práticas em domínios como jogos online multiplayer, serviços em nuvem, plataformas de streaming, aplicativos de transporte e assistentes virtuais, destacando o papel dos sistemas distribuídos na viabilização de serviços em tempo real, com alta disponibilidade e desempenho. A seção sobre comunicação entre processos discute os modelos síncrono e assíncrono, bem como protocolos e formatos de serialização utilizados para garantir interoperabilidade e eficiência na troca de dados.

Por fim, são abordados os principais desafios na construção e manutenção de sistemas distribuídos: sincronização e consistência de dados, gerência de estado, controle de latência e desempenho, além da segurança e autenticação em ambientes distribuídos. A conclusão destaca o papel central desses sistemas na transformação digital e na sustentação de tecnologias emergentes como Internet das Coisas (IoT) e computação de borda, consolidando sua importância estratégica na computação moderna.

Sistemas Distribuídos

O que são sistemas distribuídos?

Sistemas distribuídos referem-se a “uma coleção de computadores independentes que se apresenta ao usuário como um sistema único e coerente” (TANENBAUM; VAN STEEN, 2007, p. 2). A concepção desse modelo computacional ganhou destaque a partir da década de 1970, com a expansão das redes e a necessidade de recursos computacionais mais distribuídos e eficientes. A descentralização proporcionou ganhos substanciais, como escalabilidade, tolerância a falhas e melhor aproveitamento da infraestrutura disponível.

Principais características

Transparência: É a ocultação da complexidade do sistema do usuário. Existem diferentes tipos de transparência, como:

- Transparência de localização: o usuário não precisa saber onde o recurso está localizado.
- Transparência de acesso: os métodos de acesso são uniformes independentemente da localização.
- Transparência de replicação: múltiplas cópias dos dados são apresentadas como uma só.
- Transparência de concorrência: múltiplos usuários podem usar o sistema simultaneamente sem interferência.
- Transparência de falha: o sistema continua operando mesmo em caso de falhas.

Escalabilidade: Capacidade do sistema de crescer em número de nós ou volume de dados sem degradação significativa de desempenho.

Heterogeneidade: Integração de diferentes plataformas de hardware, sistemas operacionais e linguagens de programação.

Confiabilidade: Alta disponibilidade e funcionamento consistente mesmo sob condições adversas.

Tolerância a falhas: Capacidade de continuar operando corretamente mesmo após falhas parciais no sistema.

Sistemas Distribuídos vs. Sistemas Centralizados

Sistemas distribuídos oferecem maior escalabilidade e tolerância a falhas, enquanto sistemas centralizados são mais simples de implementar e gerenciar. Como explicam Machado e Maia (2020, p. 36), "sistemas centralizados possuem um único ponto de controle, o que simplifica a administração, mas os torna vulneráveis a falhas únicas; já os distribuídos diluem esse controle, o que aumenta a complexidade, mas melhora a robustez do sistema". Em termos de desempenho, sistemas distribuídos podem superar os centralizados ao distribuir a carga de trabalho, mas também enfrentam desafios como latência de comunicação. Em termos de segurança, os sistemas centralizados podem ser mais fáceis de proteger devido à sua simplicidade, porém apresentam ponto único de falha. A manutenção é mais complexa em sistemas distribuídos devido à diversidade de componentes, enquanto o custo pode variar dependendo da escala.

Exemplos práticos:

- Sistemas centralizados: Mainframes, servidores de arquivos únicos.
- Sistemas distribuídos: Google Search, Netflix, Amazon Web Services.

Modelos de sistemas distribuídos

- **Microserviços**: Arquitetura onde aplicações são divididas em serviços independentes que se comunicam por APIs.
- **Modelo cliente-servidor**: Estrutura básica onde clientes solicitam serviços e servidores os fornecem.
- **Modelo peer-to-peer (P2P)**: Todos os nós atuam simultaneamente como clientes e servidores.
- **Modelo baseado em agentes ou serviços**: Componentes autônomos interagem com base em objetivos e eventos.
- **Computação em grade (grid computing)**: Compartilhamento de recursos computacionais geograficamente dispersos.

Ferramentas e Tecnologias envolvidas

A implementação de sistemas distribuídos exige o uso de uma infraestrutura tecnológica robusta, composta por ferramentas que viabilizam a comunicação entre nós, a distribuição de dados e o balanceamento de carga. De acordo com Kalinka (2012), o avanço dessas tecnologias permitiu que arquiteturas complexas fossem construídas de forma modular, escalável e tolerante a falhas, contribuindo significativamente para a viabilização de serviços distribuídos em tempo real e de larga escala.

Containers: Tecnologias como Docker e orquestradores como Kubernetes permitem implantar e escalar serviços distribuídos de forma eficiente.

Sistemas de arquivos distribuídos: Exemplo notável é o Hadoop Distributed File System (HDFS), utilizado para armazenar grandes volumes de dados.

Bancos de dados distribuídos: Cassandra e MongoDB permitem armazenamento escalável e replicado de dados.

Frameworks para processamento distribuído: Apache Spark permite processamento paralelo de grandes volumes de dados.

Usos comuns no dia a dia

Jogos online multiplayer:

Jogos digitais em ambiente multiplayer representam um dos exemplos mais dinâmicos e exigentes do uso de sistemas distribuídos. Esses jogos dependem de comunicação em tempo real e consistência de estado entre múltiplos jogadores espalhados geograficamente. Para garantir que todos os participantes compartilhem a mesma visão do jogo, são necessários servidores de jogo distribuídos que sincronizam eventos como movimentação, interações e atualizações de ambiente com latência mínima.

Tais sistemas utilizam protocolos de baixa latência como UDP para tráfego de dados críticos, além de estruturas de replicação e mecanismos de rollback para manter consistência frente a perdas de pacotes ou falhas de conexão. Grandes títulos, como Fortnite, League of Legends e Call of Duty, implementam arquiteturas de servidor altamente escaláveis e redundantes, muitas vezes empregando regiões geográficas espelhadas, balanceamento de carga e cache distribuído para garantir desempenho global.

Serviços em nuvem:

Plataformas como Google Drive e iCloud são exemplos emblemáticos de armazenamento distribuído acessível ao público geral. Esses serviços empregam sistemas de arquivos distribuídos para armazenar fragmentos de dados redundantes em diversos datacenters ao redor do mundo, garantindo tanto disponibilidade quanto durabilidade. As modificações em arquivos são rastreadas, sincronizadas em tempo quase real e replicadas entre nós, assegurando que o usuário possa acessar versões atualizadas em múltiplos dispositivos, independentemente da localização. Além disso, esses serviços integram mecanismos de autenticação robusta, como OAuth 2.0 e autenticação multifator (MFA), e utilizam criptografia em repouso e em trânsito para proteger a privacidade dos dados dos usuários. A consistência eventual ou forte (dependendo da operação) é gerenciada por meio de protocolos internos e algoritmos de controle de versão, que evitam conflitos de escrita concorrente.

Plataformas de streaming:

Serviços como Netflix e Spotify ilustram o uso intensivo de redes de entrega de conteúdo (CDNs – Content Delivery Networks), que são estruturas de sistemas distribuídos criadas para otimizar a transmissão de mídia em grande escala. Esses sistemas armazenam cópias do conteúdo em servidores próximos aos usuários finais, minimizando a latência e o congestionamento da rede.

A transmissão adaptativa, por sua vez, ajusta dinamicamente a qualidade do conteúdo de acordo com as condições da rede, promovendo uma experiência fluida mesmo em conexões instáveis. Além disso, algoritmos de recomendação personalizados processam grandes volumes de dados distribuídos em tempo quase real, utilizando frameworks como Apache Kafka e Spark para análise de logs de acesso, preferências e padrões de uso. A resiliência desses serviços depende de arquiteturas tolerantes a falhas, replicação contínua de dados e escalabilidade horizontal.

Aplicativos de transporte:

Plataformas como Uber e 99 são exemplos sofisticados de sistemas distribuídos que integram geolocalização, comunicação em tempo real, processamento de pagamentos e gerenciamento de logística em larga escala. O sistema de matching entre motorista e passageiro, por exemplo, exige consultas e atualizações contínuas a bancos de dados distribuídos de

localização, bem como o roteamento inteligente com base em condições de tráfego em tempo real.

A arquitetura desses aplicativos geralmente se baseia em microserviços, permitindo o desacoplamento funcional entre módulos como cálculo de tarifas, mapas, suporte ao usuário e notificações push. Para garantir a disponibilidade global e o desempenho, esses serviços utilizam serviços de backend escaláveis em nuvem, cache distribuído (como Redis) e mecanismos de mensageria assíncrona. A segurança também é crítica, com autenticação de usuários, criptografia de dados e sistemas antifraude baseados em aprendizado de máquina.

Assistentes virtuais:

Tecnologias como Siri, Alexa e Google Assistant exemplificam a complexidade dos sistemas distribuídos modernos ao integrarem reconhecimento de voz, processamento de linguagem natural, recuperação de informação e controle de dispositivos IoT. Essas assistentes operam como orquestradoras de diversos microserviços em nuvem, cada um responsável por tarefas específicas, como análise de intenção, busca semântica, execução de comandos ou interação com serviços de terceiros (calendários, e-mails, controles residenciais).

A voz capturada no dispositivo local é codificada, criptografada e enviada a servidores distribuídos para processamento intensivo com uso de GPUs e modelos de linguagem de larga escala. A resposta é então transmitida de volta ao dispositivo com baixa latência para garantir interações naturais. Além disso, mecanismos de personalização baseados em perfis distribuídos de usuários, replicação de sessões e aprendizado contínuo conferem um alto grau de adaptabilidade e responsividade a esses sistemas.

Comunicação entre processos

A comunicação entre processos, ou Inter-Process Communication (IPC), é um dos pilares dos sistemas distribuídos, pois permite que componentes, frequentemente executando em diferentes máquinas e ambientes, troquem informações de maneira estruturada, coordenem ações e mantenham a coesão do sistema como um todo. Essa comunicação pode se dar de forma síncrona ou assíncrona, cada uma com implicações diretas sobre o desempenho, a escalabilidade e a complexidade das aplicações.

Como observam Tanenbaum e Van Steen (2007, p. 177), “a comunicação entre processos em sistemas distribuídos é fundamental para a coordenação e o compartilhamento de recursos, devendo ser projetada com cuidado para evitar bloqueios, perdas e inconsistências”. Esse

aspecto é especialmente crítico em aplicações que dependem de interações em tempo real ou em ambientes de alta disponibilidade, onde a confiabilidade das trocas de mensagens é determinante para o funcionamento adequado do sistema.

Na comunicação síncrona, o processo emissor envia uma mensagem e permanece bloqueado até que o receptor responda, estabelecendo uma dependência temporal entre os dois. Isso implica que o emissor deve aguardar a conclusão da operação remota antes de prosseguir, o que simplifica o controle de fluxo e o tratamento de resultados, mas também aumenta a sensibilidade a latências de rede, indisponibilidades temporárias e falhas nos nós destinatários. Essa forma de comunicação é típica em chamadas de procedimento remoto (RPC), como ocorre em APIs REST tradicionais, serviços baseados em SOAP e no uso do gRPC em modo síncrono. Em todos esses casos, o bloqueio permite uma interação mais previsível e favorece o controle transacional, embora o custo seja uma escalabilidade mais limitada e uma maior propensão a gargalos em cenários de alto volume.

Já na comunicação assíncrona, o processo emissor envia a mensagem e continua sua execução sem esperar pela resposta do receptor. Essa abordagem reduz o acoplamento temporal entre os processos e favorece a resiliência, permitindo que o sistema continue operando mesmo diante de falhas momentâneas ou sobrecarga em um de seus componentes. A comunicação assíncrona é amplamente empregada em arquiteturas orientadas a eventos, onde os processos se comunicam por meio de filas intermediárias ou brokers de mensagens. Ferramentas como RabbitMQ, Apache Kafka e Amazon SQS são comumente utilizadas para implementar esse padrão, armazenando as mensagens até que os receptores estejam prontos para processá-las. Além disso, padrões de publicação e assinatura (pub/sub) e fluxos contínuos de dados (data streaming) são baseados nesse modelo, proporcionando alta escalabilidade, desacoplamento e tolerância a falhas. Mesmo tecnologias como o gRPC suportam essa abordagem por meio de streaming assíncrono e bidirecional, permitindo interações complexas com eficiência.

A base técnica dessas comunicações está nos protocolos utilizados para transmissão das mensagens entre os processos distribuídos. O protocolo TCP/IP, orientado à conexão, garante a entrega correta e ordenada das mensagens e é amplamente utilizado em contextos que exigem confiabilidade, como conexões persistentes com bancos de dados e chamadas RPC. Em contrapartida, o protocolo UDP, que não estabelece conexão, oferece desempenho superior ao custo de confiabilidade, sendo indicado para aplicações como transmissões multimídia em tempo real, chamadas VoIP e jogos online, onde a perda de pacotes é aceitável em troca de

baixa latência. O protocolo HTTP, construído sobre TCP, é um dos mais utilizados em aplicações web e APIs RESTful, oferecendo simplicidade e compatibilidade com a maior parte dos dispositivos conectados à internet. Já o gRPC, criado pelo Google, é um framework de chamadas remotas moderno que utiliza HTTP/2 como camada de transporte, com suporte a multiplexação de streams, compressão de cabeçalhos e comunicação binária altamente eficiente, sendo ideal para aplicações baseadas em microserviços.

Em relação ao conteúdo das mensagens trocadas, é necessário serializar os dados para que possam ser transmitidos pela rede de maneira estruturada e compreensível por diferentes sistemas e linguagens. A serialização converte objetos ou estruturas de dados em representações padronizadas que podem ser reconstruídas no destino. O formato JSON, por exemplo, é amplamente utilizado por ser leve, textual e legível por humanos, o que o torna ideal para aplicações web e testes. Por outro lado, o XML, embora mais verboso, oferece maior rigidez de estrutura e validação via esquemas (XSD), sendo ainda utilizado em integrações corporativas e sistemas legados. Para contextos onde o desempenho é crítico, formatos binários como Protocol Buffers (Protobuf) se destacam por gerar mensagens mais compactas e rápidas de transmitir e processar. O Protobuf é o padrão utilizado pelo gRPC e permite versionamento de mensagens, validação automática de tipos e geração de código cliente-servidor em diversas linguagens. Outros formatos eficientes, como Avro e MessagePack, também são utilizados em plataformas de processamento distribuído, como o Apache Kafka, e em aplicações embarcadas.

Dessa forma, a comunicação entre processos em sistemas distribuídos não é apenas uma questão de transmitir dados, mas envolve um conjunto complexo de decisões técnicas que impactam diretamente a confiabilidade, escalabilidade e performance da aplicação como um todo. A escolha entre comunicação síncrona ou assíncrona, o protocolo de transporte e o formato de serialização deve ser orientada por critérios de latência, volume de dados, resiliência e compatibilidade entre sistemas heterogêneos.

Desafios em sistemas distribuídos

Sincronização e consistência de dados:

A sincronização e a consistência de dados são desafios centrais em sistemas distribuídos devido à natureza descentralizada desses ambientes e à inexistência de um relógio global confiável. A sincronização envolve garantir que todas as réplicas de um dado ou todas as instâncias de um serviço compartilhem um estado coerente, mesmo em cenários com falhas de comunicação, latência de rede ou operações concorrentes.

A consistência pode assumir diversas formas, conforme os modelos adotados. A consistência forte garante que todas as operações de leitura em qualquer nó retornem o valor mais recente de uma escrita confirmada. Já a consistência eventual, adotada por muitos sistemas de alta disponibilidade como Amazon DynamoDB e Apache Cassandra, admite que diferentes nós possam, temporariamente, apresentar visões divergentes dos dados, desde que eventualmente se tornem consistentes.

Protocolos como Two-Phase Commit (2PC), Three-Phase Commit (3PC), Paxos e Raft são utilizados para alcançar consenso entre múltiplos nós em decisões críticas, como commits transacionais. A complexidade desses algoritmos reside no fato de que mesmo mensagens perdidas ou atrasadas podem comprometer a integridade do sistema. Ferramentas modernas como o Google Spanner utilizam relógios atômicos e sincronização por GPS para oferecer consistência linearizável em escala global, algo historicamente considerado impraticável.

Gerência de estado:

A gerência de estado em sistemas distribuídos refere-se à habilidade de preservar e coordenar o estado de aplicações ou serviços entre diferentes componentes distribuídos, especialmente em ambientes com balanceamento de carga e replicação dinâmica. Aplicações stateless, como APIs RESTful, não mantêm qualquer informação sobre sessões de usuários entre requisições, facilitando a escalabilidade horizontal. No entanto, aplicações stateful, comuns em transações bancárias, jogos online e plataformas de e-commerce, requerem a persistência do contexto de execução.

Manter o estado de forma distribuída exige mecanismos de sincronização e persistência confiáveis, que possam tolerar falhas sem comprometer a integridade do sistema. Soluções incluem o uso de repositórios de sessão compartilhados, bancos de dados em memória distribuídos (como Redis com persistência AOF/RDB) ou event sourcing, onde todas as mudanças de estado são representadas como eventos imutáveis e armazenados em ordem cronológica. Essa abordagem facilita a reconstrução de estados históricos e melhora a auditabilidade.

Além disso, a coordenação de estado entre múltiplas instâncias exige mecanismos de coerência de cache, versionamento de dados e locks distribuídos, frequentemente implementados com ferramentas como Apache ZooKeeper, etcd ou Consul. Esses mecanismos garantem a consistência e disponibilidade de estados críticos, como filas de mensagens ou sessões de usuários.

Latência e desempenho:

Latência e desempenho são aspectos que afetam diretamente a usabilidade e eficiência de sistemas distribuídos, especialmente quando os componentes estão fisicamente dispersos em diferentes regiões geográficas. A latência é agravada por fatores como distância física, congestionamento de rede, retransmissões causadas por perda de pacotes, e tempo de serialização/desserialização de mensagens.

O desempenho global do sistema depende de estratégias eficazes de balanceamento de carga, particionamento de dados (sharding), uso de caches distribuídos, compressão de mensagens, e escolha adequada entre comunicação síncrona ou assíncrona. Arquiteturas baseadas em mensageria desacoplada, utilizando filas como RabbitMQ, Apache Kafka ou Amazon SQS, permitem processamentos paralelos e tolerância à intermitência, contribuindo para a mitigação de gargalos.

O uso de Content Delivery Networks (CDNs) para conteúdo estático, Edge Computing para pré-processamento na borda da rede, e replicação geográfica ativa em bancos de dados também são estratégias cruciais para reduzir a latência percebida pelo usuário final. Métricas como tempo de resposta médio, throughput e tempo de recuperação após falhas (MTTR) são fundamentais para avaliação e otimização do desempenho.

Segurança e autenticação:

A segurança em sistemas distribuídos é um campo vasto e desafiador, dado que a multiplicidade de nós e a comunicação sobre redes potencialmente inseguras aumentam o risco de ataques e violações. A segurança deve ser pensada em camadas, cobrindo autenticação, autorização, integridade de dados, confidencialidade e auditoria.

A autenticação garante que apenas entidades legítimas tenham acesso ao sistema. Isso pode ser feito por meio de senhas, certificados digitais, tokens JWT (JSON Web Tokens), ou protocolos federados como OAuth 2.0 e OpenID Connect. Já a autorização define o que cada entidade pode acessar, utilizando modelos como RBAC (Role-Based Access Control), ABAC (Attribute-Based Access Control) ou políticas personalizadas.

A comunicação entre componentes distribuídos deve ser protegida com criptografia forte, usualmente baseada em TLS (Transport Layer Security), que oferece confidencialidade, integridade e autenticação mútua. Além disso, os dados em repouso devem ser criptografados

com algoritmos como AES, e o gerenciamento de chaves deve ser feito por sistemas seguros como HashiCorp Vault ou serviços de cloud (AWS KMS, Azure Key Vault).

Outros aspectos de segurança incluem:

- Controle de acesso baseado em contexto (Zero Trust Architecture);
- Detecção e resposta a incidentes (IDR/EDR) em tempo real;
- Firewalls distribuídos e políticas de segmentação de rede;
- Logs auditáveis e rastreabilidade com ferramentas como ELK Stack ou Prometheus + Grafana.

A arquitetura de segurança deve, idealmente, seguir o princípio de segurança por design, onde cada componente é projetado com o pressuposto de que poderá ser comprometido, e deve limitar o escopo dos danos de forma isolada (compartimentalização).

Conclusão

Os sistemas distribuídos constituem a base da infraestrutura digital moderna, permitindo que serviços escalem globalmente com alta disponibilidade e desempenho. Desde seu surgimento, com as primeiras redes de computadores e modelos cliente-servidor, eles evoluíram para sustentar aplicações altamente complexas, descentralizadas e resilientes. Em contraste com os sistemas centralizados, os distribuídos oferecem vantagens como escalabilidade horizontal, tolerância a falhas e heterogeneidade de componentes, sendo ideais para ambientes que exigem flexibilidade e disponibilidade contínua.

As principais características desses sistemas, como transparência de localização, replicação e falhas, garantem uma experiência fluida ao usuário, ocultando a complexidade técnica por trás da comunicação entre componentes. Sua confiabilidade depende de técnicas avançadas de redundância, replicação de dados e coordenação entre processos. Além disso, a escalabilidade é assegurada por meio de arquiteturas modulares que permitem expandir a capacidade computacional de forma eficiente, conforme necessário.

Arquiteturalmente, os sistemas distribuídos podem assumir diferentes formas, como microserviços, peer-to-peer, cliente-servidor, computação em grade e modelos baseados em agentes. Cada modelo responde a diferentes demandas de aplicação e infraestrutura, oferecendo soluções específicas para modularidade, descentralização e paralelismo. O suporte a esses modelos é viabilizado por ferramentas como containers, orquestradores (ex.: Kubernetes),

sistemas de arquivos distribuídos, bancos de dados NoSQL e frameworks de processamento como Apache Spark.

No cotidiano, os sistemas distribuídos estão presentes em diversos serviços amplamente utilizados. Jogos online multiplayer, plataformas de streaming, aplicativos de transporte e assistentes virtuais dependem de arquiteturas distribuídas para funcionar de forma sincronizada, rápida e confiável. Da mesma forma, serviços em nuvem como Google Drive e iCloud permitem a sincronização de arquivos em múltiplos dispositivos, demonstrando o alcance e a importância dessas soluções para o usuário final.

A comunicação entre processos distribuídos exige protocolos robustos como TCP/IP, HTTP e gRPC, além de formatos de serialização eficientes como JSON, XML e Protobuf. Esses elementos garantem interoperabilidade e desempenho nas trocas de dados entre componentes heterogêneos e geograficamente distribuídos. No entanto, implementar sistemas distribuídos também impõe desafios críticos, como garantir a consistência de dados, lidar com latência de rede, manter o controle de estado e proteger a segurança das comunicações.

Por fim, os sistemas distribuídos são pilares fundamentais da transformação digital. Eles sustentam não apenas os serviços digitais atuais, mas também novas tecnologias como Internet das Coisas (IoT) e computação de borda, que descentralizam ainda mais o processamento de dados. Com a expansão dessas tecnologias, os sistemas distribuídos se tornarão ainda mais relevantes, promovendo conectividade inteligente, resiliência operacional e eficiência computacional em escala global.

Referências

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim. **Sistemas Distribuídos**: conceitos e projeto. 4. ed. Porto Alegre: Bookman, 2005. Disponível em: http://www.inf.ufsc.br/~bosco.sobral/ensino/ine5645/coulouris.pdf. Acesso em: 09 maio 2025.

TANENBAUM, Andrew S.; VAN STEEN, Maarten. **Sistemas Distribuídos**: princípios e paradigmas. 2. ed. São Paulo: Pearson Universidades, 2007. 416 p.

CALIXTO, Gustavo Moreira. **Computação em nuvem e tecnologias emergentes**. São Paulo: Senac, 2024. 162 p.

SHARIF ALDIN, Hesam Nejati et al. **Consistency models in distributed systems**: a survey on definitions, disciplines, challenges and applications. 2019. Disponível em: https://arxiv.org/abs/1902.03305. Acesso em: 9 maio 2025.

FREITAS, Elyda Laisa Soares Xavier. **Knowing**: um modelo para garantia de consistência dos dados em sistemas de banco de dados relacionais em nuvem. Recife, 2014. 111 f. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Pernambuco, 2014. Disponível em: https://repositorio.ufpe.br/handle/123456789/11363. Acesso em: 9 maio 2025.

KALINKA, Kalinka Regina Lucas Jaquie Castelo Branco. Contribuições na área de sistemas distribuídos e redes de computadores. São Carlos, 2012. 167 f. Tese (Livre-docência) – Universidade de São Paulo, Instituto de Ciências Matemáticas e de Computação, 2012. Disponível em: https://www.teses.usp.br/teses/disponiveis/livredocencia/55/tde-10102012-103142/publico/LD_Kalinka_Branco.pdf. Acesso em: 9 maio 2025.

PAES, Carlos. **Comunicação em Sistemas Distribuídos**. Pontifícia Universidade Católica de São Paulo, 2007. Disponível em: https://www4.pucsp.br/~dcc-tec2/Semana_02_Comunicacao.pdf. Acesso em: 9 maio 2025.

OLIVEIRA, Santos. **Consistência de dados**: o que é e por que você deveria se importar. LinkedIn, 30 ago. 2024. Disponível em: https://pt.linkedin.com/pulse/consist%C3%AAncia-de-dados-o-que-%C3%A9-e-por-qu%C3%AA-voc%C3%AA-deveria-santos-oliveira-j7ktf. Acesso em: 9 maio 2025.

SICREDI TECH. **Sistemas Distribuídos**: Conceito e Definições. Medium, 2022. Disponível em: https://medium.com/sicreditech/sistemas-distribu%C3%ADdos-conceito-e-defini%C3%A7%C3%B5es-f2baa4efc88d. Acesso em: 9 maio 2025.