A1: De que forma a comunicação entre processos influencia a eficiência dos Sistemas Distribuídos? Compare as diferentes abordagens, tais como mensagens assíncronas e síncronas.

A comunicação entre processos é essencial para sistemas distribuídos, pois coordena a troca de informações e ações entre diferentes componentes. As abordagens de mensagens síncronas e assíncronas diferem, afetando fatores como latência, escalabilidade e simplicidade do design.

A seguir, analiso as características de cada abordagem:

Na comunicação síncrona: os processos origem e destino são sincronizados a cada mensagem. Operações como send e receive causam bloqueio, ou seja, o processo origem aguarda até que o destino realize a receção da mensagem;

Na comunicação assíncrona: o emissor pode continuar as operações após enviar uma mensagem, sem esperar pela resposta, ou seja, a transmissão ocorre paralelamente, utilizando buffers locais.

Relativamente aos Impactos na Eficiência verificamos o seguinte:

Na comunicação síncrona: a sincronização da comunicação garante que as mensagens sejam processadas por ordem. Por outro lado, este tipo de comunicação pode introduzir atrasos significativos, já que o emissor fica bloqueado até que o destinatário receba e processe a mensagem. Isso pode ser especialmente problemático em redes lentas ou congestionadas. Esta ideia é suportada na citação seguinte *"Quando um envio (send) é feito, o processo origem (ou thread) é bloqueado até que a receção (receive) correspondente seja realizada."* (Coulouris et al., 2013, p. 147,148).

Este tipo de comunicação é adequado para transações que exigem respostas imediatas e consistência, como consultas de banco de dados por exemplo.

Na comunicação assíncrona: Por não bloquear o processo origem, permite-nos um melhor uso de recursos e maior capacidade de escalabilidade. Podemos ver isto mesmo nesta citação *"Na forma assíncrona de comunicação, o uso da operação* send *é não bloqueante, no sentido de que o processo origem pode prosseguir assim que a mensagem tenha sido copiada para um buffer local."* (Coulouris et al., 2013, p.148)*, demonstrando assim* como a comunicação assíncrona permite maior fluidez no processamento, ao evitar que o processo de origem fique

parado enquanto a mensagem é transmitida. Este tipo de comunicação é ideal para sistemas com necessidade de escalabilidade, como aplicativos de mensagens e sistemas de e-mail

A meu ver a comunicação assíncrona torna os sistemas mais eficientes e capazes de lidar com grandes volumes de trabalho sendo mais utilizada em aplicativos de mensagens, sistemas de e-mail, processamento de tarefas em segundo plano e muitas outras áreas da computação, ao passo que a comunicação síncrona é preferível para aplicações que exigem interatividade e baixa latência, como jogos em tempo real.

Em suma, a escolha entre comunicação síncrona e assíncrona impacta a eficiência de sistemas distribuídos. A comunicação síncrona, embora garanta ordem e consistência, é menos eficiente em sistemas de alta carga devido ao bloqueio de processos ao passo que a comunicação assíncrona, por sua vez, oferece maior escalabilidade e desempenho, ideal para aplicações com grandes volumes de trabalho ou alta concorrência.

Bibliografia:

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim; BLAIR, Gordon. *Sistemas distribuídos: conceitos e projeto*. Tradução: João Eduardo Nóbrega Tortello. Revisão técnica: Alexandre Carissimi. 5. ed. Porto Alegre: Bookman, 2013.

A2 - Como é que os modelos de arquiteturas distribuídas, por exemplo peer-to-peer e cliente-servidor, têm impacto na segurança e a confiabilidade das aplicações distribuídas?

Os modelos de arquitetura distribuída, como peer-to-peer e cliente-servidor, têm características distintas que influenciam a segurança e confiabilidade das aplicações distribuídas devido às diferenças fundamentais na forma como os recursos e dados são geridos e compartilhados. Depois de ler a secção 2.3 Modelos de arquitetura para sistemas distribuídos do livro Sistemas distribuídos: conceitos e projeto, podemos destacar o seguinte:

1.Sobre a arquitetura Cliente-Servidor

a) Impacto na Segurança:
Oferece maior controle centralizado, permitindo que medidas de proteção sejam aplicadas diretamente no servidor. Nesta parte da secção, o autor refere que "*Os processos clientes interagem com processos servidores [...] para acessar os recursos compartilhados que estes gerenciam."* (Coulouris et al., 2013, p. 46). Significando assim, no meu entendimento que, o servidor é o ponto principal para gerir recursos e atender solicitações.
O autor dá mesmo um exemplo de que "*Os servidores Web, e a maioria dos outros serviços Internet, são clientes do serviço DNS, que mapeia nomes de domínio Internet a endereços de rede (IP)."* (Coulouris et al., 2013, p.46), aonde podemos ver que há uma cadeia de dependência em serviços centrais, o que pode amplificar riscos. Ataques, como falhas no DNS podem levar a um impacto direto nos servidores e, consequentemente, nos clientes.

b) Impacto na Confiabilidade:
A centralização pode provocar falhas críticas se o servidor principal ficar indisponível, prejudicando todos os clientes dependentes, ou seja, como podemos verificar na primeira citação aqui feita, o seu autor pretende destacar a ideia de que se o servidor centraliza o fornecimento de recursos, isto implica que, se o servidor principal ficar indisponível, os clientes não conseguirão aceder aos recursos necessários.
A falta de protocolos seguros ou a falha na proteção do servidor podem originar consequências graves para os dados e serviços confiados a este modelo, ou seja, as vulnerabilidades das comunicações cliente-servidor, sem medidas de segurança adequadas, podem propiciar que dados sensíveis fiquem expostos a riscos como interceção e ataques mal-intencionados.

2.Sobre a arquitetura Peer-to-Peer (P2P)

a) Impacto na Segurança:

Na arquitetura descentralizada, onde os dados e recursos estão distribuídos entre diversos peers, torna-se mais desafiador implementar controles de segurança uniformes, o que pode aumentar a vulnerabilidade dessa configuração.

Apesar disso, a descentralização pode diminuir o risco de ataques a um único ponto crítico, eliminando assim a dependência de um único ponto central. Esta ideia vai de encontro ao que os autores da seção 2.3 diziam: "*Cada objeto é replicado em vários computadores para distribuir a carga ainda mais e para fornecer poder de recuperação no caso de desconexão de computadores individuais."* (Coulouris et al., 2013, p.48)., ou seja, esta replicação aumenta a confiabilidade, pois garante que mesmo com falhas, os dados permaneçam acessíveis já que os dados e a funcionalidade estão distribuídos pelos vários peers. O que pretendia dizer era que mesmo que alguns nós fiquem indisponíveis, o sistema pode continuar operando graças à distribuição dos dados e serviços entre vários pares.

b) Impacto na Confiabilidade:

Como todos os peers contribuem com recursos e funções semelhantes, o sistema torna-se mais resistente a falhas de qualquer nó individual.

Porém, o funcionamento confiável depende da cooperação dos pares, que nem sempre podem ser confiáveis ou estáveis. *"A necessidade de colocar objetos individuais, recuperá-los e manter réplicas entre muitos computadores torna essa arquitetura significativamente mais complexa do que a arquitetura cliente-servidor."* (Coulouris et al., 2013, p.48)*, ou seja,* o autor chama a atenção que, o funcionamento confiável depende da colaboração entre os peers, a qual não é sempre garantida.

Em suma, as arquiteturas cliente-servidor e peer-to-peer têm impactos distintos na segurança e confiabilidade das aplicações distribuídas. No que diz respeito ao cliente-servidor, este oferece um controlo centralizado, facilitando a implementação de medidas de segurança, mas é vulnerável a falhas no servidor central. Já o P2P, embora mais resiliente a falhas devido à descentralização, apresenta desafios na implementação uniforme de controles de segurança.

Bibliografia :

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim; BLAIR, Gordon. *Sistemas distribuídos: conceitos e projeto*. Tradução: João Eduardo Nóbrega Tortello. Revisão técnica: Alexandre Carissimi. 5. ed. Porto Alegre: Bookman, 2013.

B1: Qual é o impacto da sobrecarga de comunicação na latência de resposta em Sistemas Distribuídos, e quais são as estratégias para minimizar esse impacto?

A sobrecarga de comunicação em sistemas distribuídos aumenta a latência devido por exemplo: ao grande volume de mensagens trocadas entre processos, à complexidade dos protocolos de comunicação entre outras. Isto pode originar atrasos no tempo de resposta e degradação do desempenho geral.

Depois de ler o Capítulo 4 do livro Sistemas distribuídos: conceitos e projeto, nomeadamente a seção 4.4 - Comunicação por multicast (difusão seletiva) - verifico como o multicast pode ser usado para reduzir a sobrecarga na comunicação e assim minimizar a latência. Ora vejamos, em vez de enviar mensagens individuais para cada processo num grupo, o multicast transmite uma única mensagem para todos os destinatários. Na página 170 podemos ver este exemplo: *"(…) o multicast para um grupo pode ser usado para notificar os processos de quando algo acontece. Por exemplo, no Facebook, quando alguém muda seu status, todos os seus amigos recebem notificações. Do mesmo modo, os protocolos de publicar-assinar podem fazer uso de multicast de grupo para disseminar eventos para os assinantes."* (Coulouris et al., 2013, p.170). Isto de certa forma, economiza largura de banda e reduz o congestionamento na rede, resultando em menor latência.

Na seção 4.6 - Estudo de caso: MPI (Message Passing Interface) do mesmo livro, podemos ver como o MPI é utilizado para otimizar a troca de mensagens em sistemas distribuídos, apresentando técnicas que minimizam sobrecarga e melhoram o desempenho geral, diminuindo assim a latência de resposta em Sistemas Distribuídos. O autor refere que *"O padrão MPI também foi projetado de forma a ser flexível, e o resultado é uma especificação de passagem de mensagens abrangente em todas as suas variantes (…)"* (Coulouris et al., 2013, p.179)., ou seja, o MPI foi desenvolvido para oferecer um paradigma de programação distribuída leve e eficiente, com foco no desempenho. Ele utiliza buffers para gerenciar dados em trânsito e oferece operações de envio e “recebimento” de mensagens com variantes síncronas e assíncronas, permitindo assim um maior controle sobre a comunicação e minimizando a latência.

Pretendo também abordar a ideia de que, minimizar o tamanho dos pacotes de dados transmitidos por meio de técnicas de compactação, minimizará a sobrecarga de comunicação e desta forma diminuirá a latência.

No exemplo dado (Figura 1), ao usar o IDL (Interface Definition Language) do CORBA, permite que as mensagens sejam compactadas de maneira estruturada antes de serem enviadas. Esta prática diminui o tamanho dos pacotes transmitidos, aliviando a carga na rede e reduzindo os tempos de envio e processamento, impactando diretamente na latência. Com este exemplo pretendia demonstrar que, com operações de empacotamento e desempacotamento com base nas definições no IDL, o CORBA organiza os dados de forma eficiente, reduzindo assim a latência.

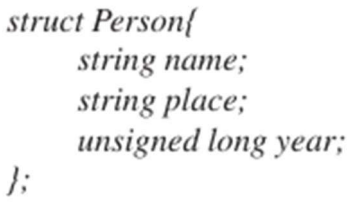

Figura 1: (Coulouris et al., 2013, p.161).

Penso que implementar algoritmos de prioridade para dados críticos, utilizando protocolos eficientes que reduzam o número de mensagens trocadas e melhorem o desempenho geral (como o MPI), são abordagens que aprimoram a eficiência na comunicação e minimiza o impacto da sobrecarga de dados na latência de resposta em sistemas distribuídos, garantindo assim um desempenho mais eficaz.

Em suma, poderia dizer que:

- Multicast reduz mensagens redundantes e economiza largura de banda.
- MPI otimiza a troca de mensagens com variantes flexíveis.
- Compactação diminui o tamanho dos pacotes, aliviando a rede.

Bibliografia:

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim; BLAIR, Gordon. *Sistemas distribuídos: conceitos e projeto*. Tradução: João Eduardo Nóbrega Tortello. Revisão técnica: Alexandre Carissimi. 5. ed. Porto Alegre: Bookman, 2013.

B2: Como é que as estratégias de caching e replicação de dados podem ser utilizadas para reduzir a latência e melhorar a escalabilidade de Sistemas Distribuídos?

As estratégias de caching e replicação de dados são fundamentais para reduzir a latência e melhorar a escalabilidade em sistemas distribuídos.

O caching armazena dados frequentemente consultados em locais próximos aos utilizadores ou componentes que os consomem, eliminando a necessidade de buscar informações em servidores remotos ou bancos de dados. Por exemplo, um cache na memória pode atender milhares de solicitações por segundo com latência significativamente menor do que um banco de dados baseado em disco. Assim sendo, ao evitar consultas repetitivas ao banco de dados ou servidores centrais, o caching reduz o tráfego de rede e a carga nos servidores, melhorando o desempenho geral do sistema.

Existem diversas técnicas de caching, mas gostaria de apresentar um exemplo onde o cache distribuído melhora significativamente a latência: *"o uso do Redis pela NVIDIA Triton para cache distribuído. Durante os testes no Google Cloud Platform com o modelo DenseNet, o Triton emparelhado com o Redis gerenciado 329 inferências por segundo com uma latência média de 3.030 µs. Sem o cache, o sistema só conseguiu 80 inferências por segundo com uma latência muito maior de 12.680 µs." ("Top 7 Data Caching Techniques for AI Workloads", s.d.)*. Este exemplo demonstra como o cache distribuído pode **drasticamente melhorar a eficiência e reduzir a latência**, descarregando operações para um sistema dedicado e permitindo que o Triton concentre seus recursos na execução de inferências.

A replicação de dados cria cópias dos dados em múltiplos servidores ou regiões geográficas, permitindo que várias solicitações sejam atendidas simultaneamente sem sobrecarregar um único servidor. Ao distribuir cópias de dados em locais geograficamente dispersos, os utilizadores podem aceder a dados de servidores mais próximos, reduzindo assim a latência. Um exemplo prático desta abordagem é o **modelo adotado pela Netflix**. Para garantir um acesso rápido e sem interrupções, a plataforma replica conteúdos populares em várias regiões, permitindo que os utilizadores acedam a vídeos a partir dos servidores mais próximos, otimizando a experiência de streaming. Isso significa que, se um usuário em Lisboa quiser assistir a "Stranger Things", o vídeo provavelmente virá de um servidor Netflix em Portugal, e não de um data center nos EUA. Isso reduz latência, melhora a qualidade do streaming e evita buffering.

Uma outra estratégia eficaz para manter a consistência dos dados replicados, reduzindo o impacto na latência, é o multicast, conforme abordado na seção 4.4 do livro *Sistemas distribuídos: conceitos e projeto*. Esta técnica permite a distribuição eficiente de atualizações para múltiplos nós simultaneamente. Um exemplo prático é o uso do multicast em clusters de bancos de dados, onde atualizações podem ser enviadas para várias réplicas ao mesmo tempo, garantindo sincronização rápida e minimizando a sobrecarga de comunicação.

Um exemplo disto é: O banco de dados central da Amazon regista que o último "Kindle" foi vendido. Em vez de enviar essa informação individualmente para todos os centros de distribuição (Londres, Paris, Berlim, etc.), o sistema envia uma mensagem multicast para todos os servidores que estão escutando. Cada servidor de armazém recebe a informação ao mesmo tempo: "Produto X: esgotado" //"Atualizar stock para zero". Isto garante sincronização rápida entre centros de distribuição, reduz a sobrecarga na rede e evita erros, como vendas de produtos já indisponíveis.

Em suma, as estratégias de caching e replicação podem ser combinadas para obter o máximo benefício. Por exemplo, um sistema pode replicar dados em vários servidores e, em seguida, usar o caching para armazenar cópias locais dos dados em cada servidor. Isto garante alta disponibilidade e baixa latência. Estas estratégias ajudam a minimizar a latência, tornando os sistemas distribuídos mais eficientes e escaláveis.

Bibliografia:

Top 7 Data Caching Techniques for AI Workloads. (s.d.). Serverion. Consultado a 04.04.2025. https://www.serverion.com/pt_br/uncategorized/top-7-data-caching-techniques-for-ai-workloads/

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim; BLAIR, Gordon. *Sistemas distribuídos: conceitos e projeto*. Tradução: João Eduardo Nóbrega Tortello. Revisão técnica: Alexandre Carissimi. 5. ed. Porto Alegre: Bookman, 2013.