E-fólio A | Folha de resolução para E-fólio

UNIDADE CURRICULAR: Sistemas Distribuídos

CÓDIGO: 21108

DOCENTE: Nelson Russo

A preencher pelo estudante

NOME: Luís Carlos Crispim Pereira

N.º DE ESTUDANTE: 2300163

CURSO: Licenciatura Engenharia de Informática

DATA DE ENTREGA: 03/04/25

TRABALHO / RESOLUÇÃO:

Questão 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?

Resposta A2:

A arquitetura de um sistema distribuído influencia profundamente a forma como são geridas falhas, ameaças e o desempenho geral das aplicações. Os modelos cliente-servidor e peer-to-peer (P2P) abordam estes desafios de maneiras distintas, afetando diretamente a segurança e a confiabilidade dos sistemas.

No modelo cliente-servidor, há uma clara centralização. Um ou mais servidores fornecem serviços e dados, enquanto os clientes os consomem. Esta organização facilita a aplicação de políticas de segurança, como autenticação centralizada, controlo de acesso e registo de atividades, pois a maior parte da lógica está concentrada num único ponto (Coulouris et al., 2011). No entanto, essa centralização também introduz riscos significativos. A existência de um ponto único de falha significa que, se o servidor for atacado ou tiver uma falha técnica, a aplicação pode tornar-se indisponível para todos os utilizadores. Ataques como negação de serviço (DDoS) exploram precisamente essa vulnerabilidade.

Além disso, em sistemas altamente centralizados, a escalabilidade horizontal pode tornar-se complexa e a confiança do utilizador depende fortemente da robustez da infraestrutura. Exemplos práticos incluem plataformas de jogos online e lojas digitais, como Steam ou League of Legends. Nestes casos, o modelo facilita a gestão da segurança, mas exige elevada disponibilidade e manutenção dos servidores para garantir fiabilidade.

Por oposição, a arquitetura peer-to-peer distribui funções por todos os nós da rede. Cada nó pode agir como cliente e servidor, o que reduz a dependência de qualquer entidade central. Esta descentralização melhora a tolerância a falhas, já que o sistema continua funcional mesmo que alguns nós se tornem inativos (Coulouris et al., 2011). Contudo, também dificulta a aplicação de políticas de segurança consistentes, como autenticação e verificação de integridade. Um exemplo conhecido é o BitTorrent, que permite uma partilha eficiente de ficheiros, mas sem mecanismos centralizados de controlo, tornando mais fácil a propagação de conteúdos maliciosos ou não verificados (Tanenbaum & van Steen, 2007).

A confiabilidade no modelo P2P resulta em grande parte da replicação de dados. Esta prática assegura que a informação está disponível em vários pontos da rede, mesmo em caso de falhas. No entanto, levanta desafios ao nível da consistência dos dados, exigindo mecanismos de sincronização adicionais.

Muitos sistemas modernos adotam uma abordagem híbrida, procurando equilibrar controlo e escalabilidade. Por exemplo, serviços de streaming utilizam uma arquitetura cliente-servidor para autenticação e gestão de utilizadores, mas distribuem conteúdos por meio de redes de entrega geograficamente distribuídas. As CDNs, descritas por Coulouris et al. (2011), melhoram a eficiência e a latência ao aproximar os dados dos utilizadores finais, sem comprometer a centralização das funções críticas.

Em síntese, o modelo cliente-servidor oferece maior controlo sobre a segurança e simplifica o desenvolvimento inicial. Já o modelo P2P destaca-se pela sua resiliência e escalabilidade, embora exija maior complexidade para manter a integridade e a confiança no sistema. A escolha

da arquitetura depende do contexto e das prioridades da aplicação, podendo inclusive combinar características de ambos os modelos.

Referências Bibliográficas A2:

Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2011). Distributed systems: Concepts and design (5th ed.). Addison-Wesley.

Tanenbaum, A. S., & van Steen, M. (2007). Distributed systems: Principles and paradigms (2nd ed.). Pearson Education.

Questão A3:

Quais são os desafios da gestão de consistência de dados em Sistemas Distribuídos, e como é que os diferentes os modelos de consistência podem ser relevantes no desempenho e a confiabilidade?

Resposta A3:

A consistência de dados é uma das preocupações centrais nos sistemas distribuídos, onde múltiplas réplicas de informação são mantidas em diferentes nós. O principal desafio consiste em garantir que todas essas réplicas mantêm o mesmo estado lógico, apesar da latência da rede, falhas de comunicação e atualizações concorrentes (Coulouris et al., 2011). Este problema é agravado pelas limitações apontadas no teorema CAP, segundo o qual é impossível, em presença de falhas de rede, garantir ao mesmo tempo consistência, disponibilidade e tolerância a partições (Gilbert & Lynch, 2002).

A gestão eficaz da consistência exige que se escolha um modelo adequado, consoante os requisitos da aplicação. O modelo de consistência forte, conforme descrito por Coulouris et al. (2011), assegura que todas as leituras de um dado devolvem sempre o valor mais recente escrito, independentemente de qual réplica é consultada. Este comportamento garante uma visão uniforme e previsível do sistema, sendo altamente relevante para aplicações que exigem confiabilidade elevada, como sistemas bancários ou clínicos. No entanto, o custo para manter essa consistência é elevado em termos de desempenho, pois requer sincronização entre nós antes de concluir operações de escrita.

Por oposição, o modelo de consistência eventual, descrito por Coulouris et al. (2011), permite que, na ausência de novas atualizações, as diferentes réplicas acabem por convergir para o mesmo valor. Este modelo favorece a disponibilidade e o desempenho, pois não impõe barreiras à propagação imediata de atualizações. É amplamente adotado em sistemas de larga escala, como redes sociais e serviços de cache distribuído. No entanto, este comportamento pode levar a situações em que diferentes utilizadores visualizam versões distintas dos mesmos dados, o que diminui a confiabilidade percebida (Coulouris et al., 2011).

Entre estes dois modelos, a consistência causal, apresentada por Coulouris et al. (2011), representa um compromisso. Este modelo garante que operações logicamente relacionadas (por exemplo, uma resposta que depende de uma pergunta anterior) sejam vistas na ordem correta. No entanto, permite que operações independentes apareçam em ordens distintas. A sua aplicação é particularmente útil em ambientes colaborativos, como editores de documentos online, onde manter a sequência dos eventos é essencial para a coerência do conteúdo partilhado.

Além da escolha do modelo, há ainda desafios técnicos relacionados com a deteção e resolução de conflitos, sobretudo quando diferentes réplicas são atualizadas em simultâneo. Em sistemas com consistência eventual, a lógica de aplicação precisa de lidar com estados divergentes, recorrendo a estratégias como versões baseadas em carimbos temporais ou políticas de "última escrita vence" (Coulouris et al., 2011).

Muitos sistemas modernos adotam abordagens configuráveis, permitindo ao programador ajustar o nível de consistência conforme a operação, o utilizador ou a localização. Esta flexibilidade reflete a necessidade de equilibrar o desempenho e a confiabilidade de forma contextual, especialmente em ambientes globais e altamente distribuídos.

Em síntese, a gestão da consistência em sistemas distribuídos exige decisões bem fundamentadas sobre o modelo a adotar, tendo em conta os compromissos entre resposta rápida, integridade dos dados e tolerância a falhas. Compreender os desafios e os efeitos práticos de cada modelo é essencial para projetar sistemas robustos e eficientes.

Referências Bibliográficas A3:

Adya, A. (1999). Weak consistency: A generalized theory and optimistic implementations for distributed transactions (Doctoral dissertation). Massachusetts Institute of Technology.

Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2011). Distributed systems: Concepts and design (5th ed.). Addison-Wesley.

Gilbert, S., & Lynch, N. (2002). Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services. ACM SIGACT News, 33(2), 51–59.

Questão B3:

De que forma o uso de algoritmos de consenso, por exemplo Paxos e Raft, afeta a confiabilidade e o tempo de resposta em ambientes distribuídos?

Resposta B3:

Os sistemas distribuídos, ao dependerem da cooperação entre múltiplos nós, enfrentam desafios significativos ao nível da coordenação e decisão partilhada. Nestes contextos, os algoritmos de consenso têm um papel fundamental, pois permitem que todos os nós concordem sobre um determinado valor ou estado, mesmo na presença de falhas. A fiabilidade de muitos serviços distribuídos modernos, como bases de dados replicadas ou sistemas de configuração distribuída, depende diretamente da existência de um mecanismo de consenso eficaz (Coulouris et al., 2011).

Um dos algoritmos mais conhecidos é o Paxos, proposto por Lamport, que fornece uma solução teórica robusta para alcançar consenso num ambiente com falhas e comunicação assíncrona. O algoritmo é tolerante a falhas parciais e assegura que, se uma decisão for tomada por um subconjunto de nós, esta decisão será preservada, mesmo que outros nós entrem ou saiam da rede. No entanto, o Paxos é frequentemente criticado pela sua complexidade de implementação e por introduzir atrasos significativos, especialmente quando há contenção ou falhas na rede (Coulouris et al., 2011).

Como alternativa mais pragmática, surgiu o algoritmo Raft, que simplifica a lógica do Paxos, mantendo as mesmas garantias de segurança. Raft estrutura o consenso em torno de uma figura de líder, que centraliza as decisões e as propaga aos restantes nós. Essa abordagem melhora a compreensibilidade e facilita a implementação, sendo adotada em sistemas amplamente usados

como o etcd ou o Consul. Tal como o Paxos, o Raft melhora substancialmente a fiabilidade do sistema, ao garantir que todas as réplicas mantêm o mesmo estado, mesmo após falhas ou reinícios (Ongaro & Ousterhout, 2014).

Contudo, essa garantia de consistência vem com um custo. Ambos os algoritmos requerem várias trocas de mensagens para chegar a acordo, o que aumenta o tempo de resposta, especialmente em redes com latência elevada. Por exemplo, em Raft, cada decisão requer confirmação de uma maioria dos nós, o que implica pelo menos duas fases de comunicação: eleição do líder e replicação da entrada no log. Assim, embora a fiabilidade aumente, a latência média por operação também tende a subir, sobretudo em sistemas com elevado número de nós ou comunicação intercontinental (Coulouris et al., 2011).

Além disso, algoritmos de consenso limitam a escalabilidade horizontal pura. Em grandes clusters, o tempo necessário para obter quórum pode crescer, afetando a previsibilidade das respostas. Por isso, muitos sistemas modernos optam por isolar o consenso apenas nas partes críticas da arquitetura, mantendo outras componentes mais leves e rápidas.

Em suma, os algoritmos de consenso como Paxos e Raft são essenciais para garantir confiabilidade e consistência em sistemas distribuídos sujeitos a falhas. No entanto, a sua utilização deve ser cuidadosamente ponderada, tendo em conta o impacto no tempo de resposta e o contexto operacional do sistema.

Referências Bibliográficas B3:

Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2011). Distributed systems: Concepts and design (5th ed.). Addison-Wesley.

Ongaro, D., & Ousterhout, J. (2014). In search of an understandable consensus algorithm. USENIX Annual Technical Conference.

Questão B4:

Como é que as arquiteturas baseadas em microserviços podem influenciar a eficiência e a previsibilidade do tempo de resposta em Sistemas Distribuídos?

Resposta B4:

As arquiteturas baseadas em microserviços promovem a decomposição de um sistema em componentes pequenos, independentes e especializados. Esta abordagem contrasta com os sistemas monolíticos, em que a lógica de negócio está fortemente acoplada num único bloco. Ao separar funcionalidades em serviços autónomos que comunicam entre si, os microserviços têm um impacto direto na eficiência operacional e na previsibilidade do tempo de resposta (Coulouris et al., 2011).

Um dos principais benefícios dos microserviços está na eficiência do desenvolvimento e da operação. Cada serviço pode ser desenvolvido, escalado e mantido de forma independente, o que permite otimizações específicas por componente. Por exemplo, um serviço de autenticação pode ser escrito numa linguagem de alto desempenho, enquanto outro, de geração de relatórios, pode priorizar a fiabilidade. Esta especialização contribui para respostas mais rápidas e adaptadas à carga real (Newman, 2015).

A escalabilidade horizontal individualizada permite que serviços mais solicitados sejam replicados conforme a necessidade. Isto é visível em sistemas como o da Amazon, onde serviços críticos têm réplicas adicionais e utilizam cache nos edge nodes para acelerar a resposta

(Newman, 2015). Com isto, o sistema responde mais rapidamente aos pedidos mais frequentes sem sobrecarregar a infraestrutura.

No entanto, os microserviços também introduzem complexidade na comunicação. Quando um pedido depende de múltiplos serviços encadeados, o tempo de resposta global depende da latência cumulativa e da resiliência da cadeia de chamadas. Em sistemas como o da Netflix, esta questão é mitigada através de circuit breakers como o Hystrix, que isolam falhas e impedem que um serviço lento afete o sistema inteiro (Dragoni et al., 2017).

A comunicação entre microserviços, baseada frequentemente em chamadas HTTP/gRPC, pode ser menos previsível do que chamadas locais em sistemas monolíticos. Para compensar essa variabilidade, utilizam-se técnicas como balanceamento de carga local (ex: Netflix, que utiliza ferramentas como Zuul e Ribbon) e retry com timeout ajustado, permitindo responder com fiabilidade mesmo sob falha parcial (Coulouris et al., 2011).

A observabilidade distribuída também desempenha um papel importante. Ferramentas como Zipkin e Prometheus monitorizam tempos de resposta por serviço, permitindo identificar rapidamente gargalos e ajustar as configurações de acordo com as métricas reais (Newman, 2015). Isto contribui para maior previsibilidade do desempenho numa arquitetura naturalmente dinâmica.

Em síntese, a arquitetura de microserviços pode melhorar a eficiência global e tornar os tempos de resposta mais previsíveis, desde que sejam adotadas boas práticas de comunicação, isolamento e monitorização. Com base em estratégias como escalabilidade seletiva, cache localizada e tratamento preventivo de falhas, os microserviços tornam-se uma abordagem poderosa para a construção de sistemas distribuídos robustos e adaptáveis (Dragoni et al., 2017; Coulouris et al., 2011).

Assim, as arquiteturas baseadas em microserviços, quando bem implementadas, contribuem não só para maior eficiência dos sistemas distribuídos, mas também para uma previsibilidade mais controlada dos tempos de resposta, alinhando-se com os objetivos operacionais e de experiência do utilizador.

Referências Bibliográficas B4:

Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2011). Distributed systems: Concepts and design (5th ed.). Addison-Wesley.

Newman, S. (2015). Building microservices: Designing fine-grained systems. O'Reilly Media.

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: Yesterday, Today, and Tomorrow. In M. Mazzara & B. Meyer (Eds.), Present and Ulterior Software Engineering (pp. 195–216). Springer.