PUBLISHER/SUBSCRIBER: UMA ARQUITETURA DE SISTEMAS DISTRIBUÍDOS TOLERANTE A FALHAS

Pedro Paulo Dias dos Santos[1]

RESUMO

A disciplina de Sistemas Distribuídos é muito importante na formação do egresso do curso de Sistemas de Informação, este texto objetiva apresentar uma arquitetura de sistemas distribuídos que lida com falhas por meio da redundância de componentes e replicação de dados. Mostraremos o projeto que pretende resultar em um sistema que atinge suficientemente a distribuição de recursos, confiabilidade e escalabilidade.

Palavras chave: Sistemas distribuídos. Publisher/Subscriber. Replicação

1 INTRODUÇÃO

Sistemas distribuídos são sistemas de computador em rede no qual os processos e recursos são suficientemente distribuídos por vários computadores (M. van Steen e A.S Tanenbaum, 2024)[2]. Note o uso da palavra “suficientemente”, ela é que desenha a linha divisória entre sistemas distribuídos e sistemas descentralizados. Para satisfazer essa definição, precisamos de uma arquitetura capaz de dividir as tarefas de um sistema em processos pequenos que se comunicam e coordenam a fim de produzir o mesmo resultado que um único processo seria capaz, mas com os benéficos da distribuição – em destaque a escalabilidade. Na aplicação que será apresentada utilizaremos a arquitetura Publisher/Subscriber que, simplificando, divide o sistema em três principais componentes (1) publisher, (2) subscriber, (3) intermediário que gerencia a comunicação entre o primeiro e o segundo (chamaremos esse terceiro de broker).

2 ARQUITETURA

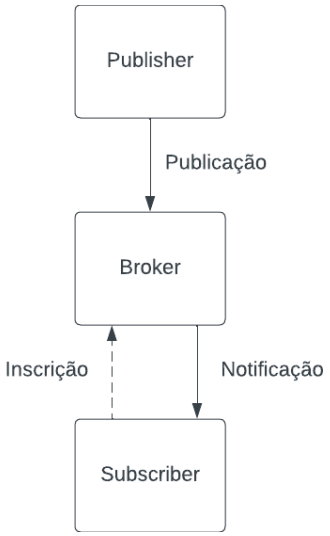
Cada um dos 3 componentes tem suas respectivas funções dentro do sistema. Do publisher se espera a capacidade de fazer publicações, do subscriber se espera a possibilidade de se

[1] Graduando em Sistemas de Informação pela Universidade Federal de Goiás/ UFG. E-mail: paulowskidias@gmail.com

[2] VAN STEEN, Maarten; TANENBAUM, Andrew S. Distributed Systems. 4. ed. v. 02. [s.l.]: [s.n.], fev. 2024. ISBN 978-90-815406-4-3 (digital). Disponível em: www.distributed-systems.net. Acesso em: 13 dez. 2024.

inscrever e desinscrever em um tópico de sua preferência; e o broker deve ser capaz de gravar as inscrições e transmitir as publicações para os subscribers que as desejam.

Figura 1 - Arquitetura Publisher/Subscriber



Fonte: Elaboração do autor a partir do livro Distributed Systems 4th edition (2024)

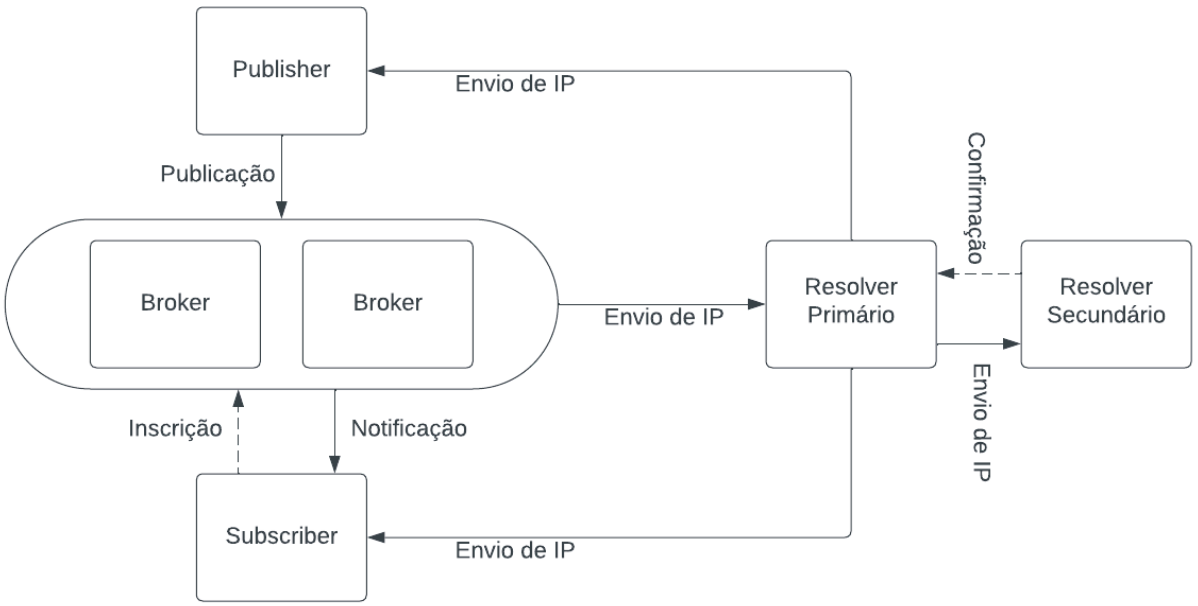
Como pode ser observado na figura 1, nessa arquitetura o broker é um ponto único de falha e isso é um problema para atingir os objetivos de confiabilidade e escalabilidade. Não nos permite atingir confiabilidade pois apenas uma única falha será capaz de alterar o comportamento desejado do sistema, pior ainda nesse caso que é o broker é um componente centralizador.

A primeira solução que vem à nossa mente é duplicar o broker. Agora, caso o primeiro falhe, ainda haverá um segundo para substituí-lo. Essa solução nos apresenta algumas novas dificuldades. Precisamos replicar as inscrições dos subscribers nos dois brokers? Se sim, como fazer? Com qual dos dois brokers os publishers e os subscribers devem se conectar?

[Guerraoui e Schiper][3] mostram duas formas de replicação; (1) Primária-backup e (2) Replicação ativa. Usaremos as duas no desenvolvimento de nossa arquitetura. Primeiro, vamos criar mais um componente que chamaremos de Resolver e sua primeira função será resolver um endereço IP de um broker funcional para os publishers e subscribers. Além disso, vamos replicar esse novo componente utilizando a técnica Primária-backup, como mostrado na figura 2.

[3]GUERRAOUI, Rachid; SCHIPER, André. Software-Based Replication for Fault Tolerance. Computer, v. 30, n. 4, p. 68-74, maio 1997. DOI: 10.1109/2.585156. Disponível em: ResearchGate. Acesso em: 13 dez. 2024.

Figura 2 - Arquitetura Publisher/Subscriber com Resolver de IP redundante



Fonte: Elaboração do autor

Para lidar com a replicação da informação, o primeiro resolver vai salvar a informação dos brokers funcionais no momento *t*, cada intervalo de tempo *i* é esperado que o próprio broker atualize seu estado; antes de enviar a informação (endereço de IP) para o cliente (publisher ou subscriber), o resolver primário enviará a atualização para o resolver secundário. Após o resolver secundário comunicar ao primário que finalizou a atualização, o primário enviará a informação ao cliente. É importante notar que estamos omitindo a coordenação entre os resolvers, na figura 3 temos um algoritmo simplificado da coordenação entre os dois.

Figura 3 - Pseudocódigo da replicação Primária-backup

Algoritmo da Réplica Primária

Data: IP recebido de um broker (*IP_Funcional*)
Result: Atualização do banco interno, envio para publishers e subscribers
`// Receber e Atualizar IP Funcional`
IP_Funcional ← IP recebido de broker; Atualizar banco de dados interno com *IP_Funcional*;
`// Função Assíncrona para Atualizar Réplica`
Função Assíncrona Atualizar_Réplica(*IP_Funcional*, *IP_Réplica*)
begin
  Confirmação ← Esperar(Requisição(*IP_Réplica*, *IP_Funcional*));
  if *Confirmação = Verdadeiro* **then**
    Enviar *IP_Funcional* para os publishers e subscribers;
  else
    `// Caso a confirmação falhe`
    Tratamento de erro ou nova tentativa;
  end
end

Algoritmo da Réplica Secundária

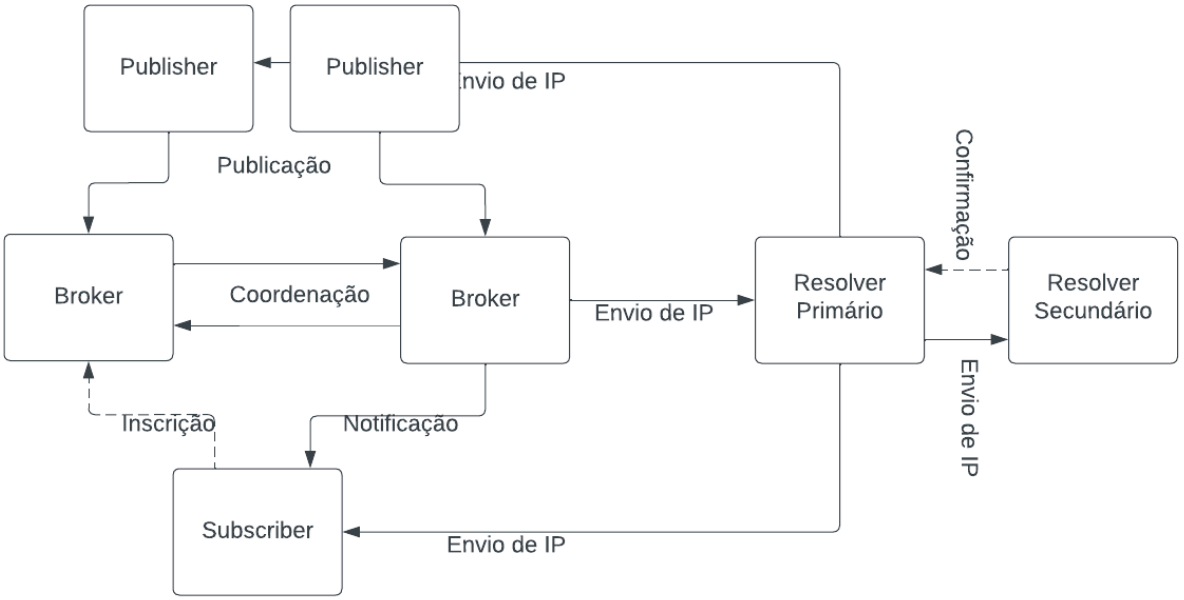
Data: IP recebido do Resolver Primário (*IP_Funcional*)
Result: Confirmação enviada à Réplica Primária
`// Receber e Atualizar IP Funcional`
IP_Funcional ← IP recebido do Resolver Primário;
Atualizar banco de dados interno com *IP_Funcional*;
Enviar Confirmação para a Réplica Primária;

Fonte: Elaboração do autor

3 REPLICAÇÃO

Se o resolver vai continuar oferecendo um endereço de qualquer um dos brokers funcionais, isso quer dizer que ambos brokers precisarão saber todas as inscrições. Poderíamos aplicar o modelo de replicação primária-backup também nos brokers, mas um ponto importante dessa replicação é que essencialmente apenas um dos processos está trabalhando ativamente, enquanto o outro processo está esperando sua falha para começar a trabalhar ativamente; em síntese: a réplica de backup está apenas repetindo ações que já foram feitas. Isso nos leva a olhar a característica de escalabilidade, se utilizarmos o modelo de réplica primária-backup não importa quantas réplicas tenhamos do broker, ainda assim apenas um estará trabalhando ativamente.

Figura 4 - Arquitetura Publisher/Subscriber com replicação ativa de Broker e Replicação Primária-backup de Resolver



Fonte: Elaboração do autor

Por outro lado, como mostra a figura 4 de maneira simplificada, o modelo de replicação ativa nos permite ter múltiplos processos idênticos trabalhando paralelamente. A dificuldade desse modelo de replicação em relação ao anterior é a necessidade de uma comunicação intensa entre os processos para coordenar as ações. Para complicar o sistema e exemplificar a necessidade e dificuldade de coordenar processos, adicionaremos dois bancos de dados que

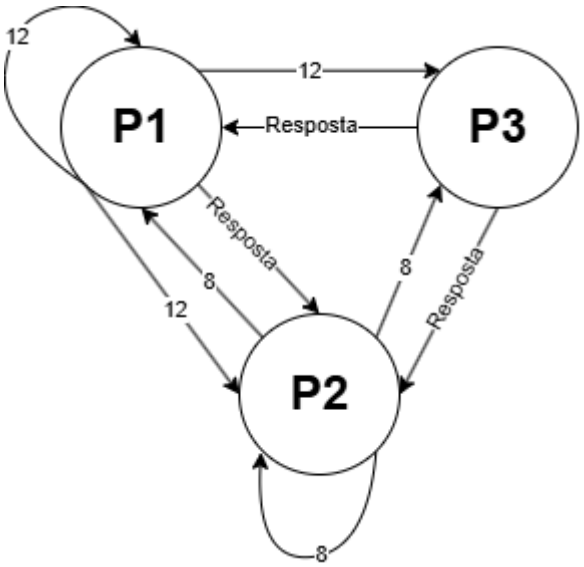
armazenarão todas as publicações e inscrições, além disso atribuiremos ao publisher capacidade de atualizar e deletar publicações.

Agora, para continuarmos em busca da confiabilidade, temos que garantir que os bancos de dados fiquem idênticos. Toda vez que uma transação for executada em um banco de dados, ela deve ser executada também no outro. A razão disso é simples, se a sequência dos registros importa e afeta no funcionamento do sistema, então é necessário que todas as réplicas sejam consistentemente iguais.

Vamos resolver o problema mais fácil primeiro, que é a execução de uma mesma transação nos dois bancos de dados. Para isso basta dar a cada um dos brokers a chave de acesso a cada um dos bancos de dados e sempre que o broker executar uma ação em um, esta só deve ser confirmada após a mesma ação ter sido executada no outro banco de dados.

Nosso problema agora é coordenar o acesso à região crítica (o banco de dados). Para isso existem muitas soluções, das mais simples às mais sofisticadas. No nosso sistema utilizaremos relógios de lógicos de Lamport[4]. A situação representada na figura 5 é muito comum para exemplificar como se dá a coordenação do acesso à região crítica utilizando relógios lógicos.

Figura 5 - Representação de coordenação com relógios lógicos entre processos distribuídos para acessar região crítica



Fonte: Elaboração do autor a partir do livro Distributed Systems 4th edition (2024)

[4] LAMPORT, Leslie. Time, Clocks, and the Ordering of Events in a Distributed System. Communications of the ACM, v. 21, n. 7, p. 558–565, jul. 1978.

O algoritmo distribuído fornecido por [Ricart e Agrawala][5] nos ajuda a lidar com a situação mostrada na figura 5. Para qualquer par de eventos *a* e *b*, deve existir, de forma inequívoca a ordem de acontecimento de *a* e *b.* Em resumo, em um sistema de comunicação confiável, quando um processo P2 solicita acesso à região crítica, ele envia uma mensagem para todos os outros processos que podem acessar o mesmo recurso, incluindo a si mesmo. Esta mensagem identifica o recurso a ser acessado, o processo que está solicitando e o tempo lógico do processo no momento da solicitação.
Para os processos destinatários que receberam a mensagem de solicitação, devemos considerar os seguintes casos:

(1) Se o destinatário não está acessando a região crítica, ele deve responder imediatamente;
(2) Se o destinatário está em posse da região crítica, ele não responde a solicitação de imediato, apenas armazena-a em uma fila;
(3) Se o destinatário também quer acessar a região crítica, mas ainda não solicitou, então os tempos lógicos dos dois processos serão comparados - o tempo lógico usado pelo destinatário é o do momento da chegada da mensagem de solicitação do processo P2. Se o tempo de P2 é menor, o destinatário manda uma mensagem de reconhecimento (*ACK*) para P2. Se o tempo do destinatário é menor, então ele armazenará a solicitação de P2 em uma fila e acessará a região crítica.

Se considerarmos que cada um dos processos é um broker, garantiremos a confiabilidade dos bancos de dados, uma vez que apenas um processo acessa-os por vez e as transações são feitas sempre em um uma ordem consistente.

4 FALHAS INDEPENDENTES

Como foi mostrado até agora, mesmo que algum um único componente falhe, ainda assim todo o sistema ficará comprometido. Porém, para resolver isso é mais simples nessa arquitetura do que na primeira apresentada. Como o tópico de falhas é muito extenso, nos limitaremos a tratar as falhas de uma forma pouco sofisticada. Uma das principais causas do tópico “Falhas” ser difícil de lidar é a dificuldade de saber se um processo está apenas lento

[5] RICART, Glenn; AGRAWALA, Ashok. An Optimal Algorithm for Mutual Exclusion in Computer Networks. Communications of the ACM, v. 24, n. 1, p. 9–17, jan. 1981.

ou se está em falha. Uma forma de resolver isso no caso dos brokers é adicionando um tempo esperado para as respostas e para atualização de informação, se este processo não responder ou se atualizar dentro do limite de tempo, então o sistema deve considerar que o broker não será capaz de desempenhar da forma correta. Ou seja, pode ser retirado do sistema até que ele volte ao normal e comunique ao sistema que está “de pé” novamente. Essa solução pode ser aplicada tanto aos resolvers quanto aos bancos de dados também. A comunicação em grupo é essencial para uma melhor identificação do atual funcionamento de cada um dos processos. Na nossa arquitetura é indispensável que cada um dos componentes armazene seu histórico de interações com os outros componentes para assim saber se o estado atual de um processo lento é na verdade uma falha ou não.

5 CONCLUSÃO

Projetar e desenvolver sistemas distribuídos não será sempre igual, é necessário avaliar os requisitos funcionais e verificar se cada uma das soluções apresentadas ou conhecidas são realmente necessárias na construção de um projeto específico. Ainda hoje há muitos sistemas que são monolitos, centralizados e altamente acoplados, mas isso não é necessariamente ruim, afinal cada problema exige uma solução. O projeto apresentado atinge o esperado de distribuição de recursos, confiabilidade e escalabilidade, mas com certeza há problemas que precisam de uma quantidade maior de processos e ainda outros novos que realizarão funcionalidades que aqui nem foram mencionadas. Entretanto, este é um pontapé inicial na discussão sobre sistemas distribuídos robustos.

REFERÊNCIAS

VAN STEEN, Maarten; TANENBAUM, Andrew S. Distributed Systems. 4. ed. v. 02. [s.l.]: [s.n.], fev. 2024. ISBN 978-90-815406-4-3 (digital). Disponível em: www.distributed-systems.net. Acesso em: 13 dez. 2024.

GUERRAOUI, Rachid; SCHIPER, André. Software-Based Replication for Fault Tolerance. Computer, v. 30, n. 4, p. 68-74, maio 1997. DOI: 10.1109/2.585156. Disponível em: <https://www.researchgate.net/publication/2954736_Software-Based_Replication_for_Fault_Tolerance>. Acesso em: 13 dez. 2024.

LAMPORT, Leslie. Time, Clocks, and the Ordering of Events in a Distributed System. Communications of the ACM, v. 21, n. 7, p. 558–565, jul. 1978.

RICART, G.; AGRAWALA, A. An Optimal Algorithm for Mutual Exclusion in Computer Networks. Communications of the ACM, v. 24, n. 1, p. 9–17, jan. 1981. Disponível em: <https://www.researchgate.net/publication/220419990_An_Optimal_Algorithm_for_Mutual_Exclusion_in_Computer_Networks>. Acesso em: 13 dez. 2024.