Sistemas de Software Distribuídos Tolerantes a Falhas

Projeto e Implementação

Versão 1.0 - 11 de março de 2016

Aristofânio Garcia

1 Introdução

Ao se falar em sistemas de software distribuído pelo menos dois requisitos de qualidade devem estar presentes: confiabilidade e disponibilidade. Estes requisitos estão relacionados, respectivamente, a garantia de que o sistema executará suas funcionalidades de acordo com o que foi especificado e de que estas funcionalidades estarão disponíveis sempre que forem solicitadas.

Muitos usuários e clientes acabam nem tratado desses requisitos quando estão descrevendo o que querem. É fácil entender isto, pois eles acreditam que de nada adianta ter um sistema se não é possível confiar nele e de nada adianta ter um sistema confiável se não é possível ter acesso a ele. Talvez, quando o assunto é sistema de software centralizado, esta forma de pensar e agir possa ser aceitável, mas quando o assunto é sistema de sofware distribuído não há nada de óbvio ou trivial nestes requisitos, a começar pela sua definição. Isto porque definir que um sistema de software distribuído possui alta confiabilidade e alta disponibilidade vai além de avaliar a corretude das suas funcionalidades e de saber quanto tempo o seu hospedeiro fica disponível. É necessário avaliar também o que ocorre em situações adversas como as situações de falhas tais como: falta de conexão, comunicação lenta, limitação de memória para processamento, interrupção repentina da comunicação, etc.

Existem diversos cenários de adversidade que podem simulados e estudados antes de se desenvolver um sistema distribuído, mas projetar um sistema que atenda a todos os cenários é muito pouco provável que isto seja realizável e caso seja poderá tornar o sistema inviável, seja pelo alto custo operacional ou pela longa jornada necessária para implementar e testar todos os cenários.

Sendo assim, a solucão é criar sistemas que sejam tolerantes a falhas, ou seja, sistemas que consigam detectar e recuperar-se mesmo diante de falhas. Para que isto seja possível, um sistema distribuido precisa ser capazes de: gerar dados que apresentem como está ocorrendo o seus funcionamento, possibilitando ações preventivas de manutenção ou correção de erros (prevenção de falhas); remover estados indesejados ou incorretos (remoção de falhas); previnir ocorrências de falhas decidindo por estratégias previamente programadas com base em dados

fornecidos por indicadores (previsão de falhas) ou suportar e tratar as falhas (tolerânca a falha).

Como existem situações que podem não ser previstas pelo mais experiênte desenvolvedor, não é sufiente desenvolver sistemas que sejam capazes apenas de prevenir e remover as falhas. Além disto, situações inesperadas podem gerar cenários que não estejam contemplados dentre as estratégias de previsão de falhas e por isto o sistema não saberá responder a este cenário criando passando assim para o estado de falha. Quando isto ocorre, o sistema precisa ser robusto o suficiente para detectar, confinar, recuperar e tratar os erros.

Para tanto é preciso que os desenvolvedores entendam a arquitetura do sistema como um todo e o papel de todos os elementos que compões este sistema. Só assim será possível avaliar, sob a ótima de um determinado cenário, quais componentes podem ser afetados, qual a criticidade de cada componentes e como eles deverão agir ou reagir diante de uma provavel falha ou depois que ela ocorre.

Como pode-se imaginar, o projeto e a implementação de um sistema distribuído com tais características é complexo e pode demandar mais tempo do que o suficiente para finalizar este curso. Assim sendo, para que seja possível entender algumas das técnicas de tolerância a falha, serão estudadas apenas alguns problemas e soluções, os quais são:

- garantia da entrega de mensagens;
- redução do tempo de espera por parte do cliente/consumidor e
- aumentar a disponibilidade sobre algumas funcionalidades de um sistema.

2 Garantia da entrega de mensagens

Entendendo que para o usuário final um sistema distribuido aparenta ser apenas um componente, então ao realizar qualquer requisição este usuário aguardará por uma resposta que esteja de acordo com o que foi especificado no seu desenvolvimento e quando isto não ocorre o sistema como um todo poderá ser considerado não confiável. O que ele não imagina é que este erro pode ter sido originado simplesmente por um falha de conexão causada por uma falta de energia do lado do componente servidor (o que, na verdade, nem lhe interessa).

Sendo assim, o grau de confiabilidade de um sistema distribuido pode ser medido de acordo com a corretudo do seu processamento de dados em relação ao que foi especificado, entendendo que somente poderá haver um processamento correto se os dados forem entregues ao componente correto e se, após a execução do processamento destes dados, o resultado for entregue ao solicitante em conformidade com o que foi especificado.

Sabendo que os dados enviados são chamados de mensagens em Sistemas Distribuidos e considerando o entendimento exposto no parágrafo anterior, pode-se afirmar que para ter um sistema distribuido confiável é necessário que as mensagens, seja de requisição ou de

resposta, sejam entregues ao componentes certos, ou seja, que haja **garantia da entrega de mensagens**.

Por si só, um sistema distribuido que garante a entrega de mensagens não pode ser considerado confiável, afinal de contas se não for entregue no tempo certo ou aceitável a mensagem pode ser considerada inválida ou inútil. Além disto, é necessário saber se a mensagem que foi enviada para processamento é a mensagem que o componente cliente enviou e se a mensagem que retornou como resposta é mesma que foi enviada pelo componente servidor.

Para entender melhor como dar maior confiabilidade a um sistema distribuido neste curso será tomado como base o cenário a seguir.

2.1 Cenário

Um sistema de software distribuido composto de componentes foi projetado inicialmente para atender uma especificação arquitetural de cliente-servidor, conforme apresentado na figura 2. No entanto alguns problemas começaram a se apresentar, tais como:

- o provedor de internet possui banda compartilhada e há certos horários do dia que a velocidade fica muito lenta, chegando inclusive a faltar comunicação;
- o número de usuários que usam o sistema ao mesmo tempo é maior que o previsto causando falha de conexão ou demora superior ao desejado na resposta;
- o computador responsável pela hospedagem do componente servidor tem recursos compartilhados com outros sistemas fazendo com que o processamento de algumas das funcionalidade mais desejada (relatórios) variem de 10 minutos a 20 minutos para serem realizadas.

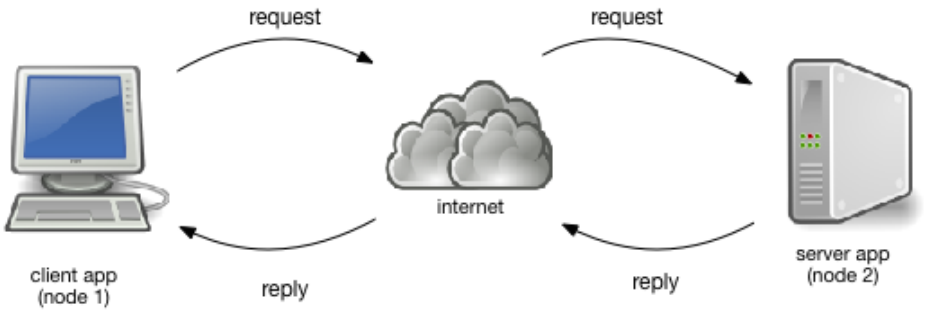

Figura 1: *Visão geral da arquitetura do sistema.*

O gerente de TI, responsável pelo projeto, agora precisa tomar atitudes para evitar que o sistema seja rejeitado pelos usuários. Para tanto as suas soluções se limitam a negociar com fornecedores, solicitar mais recurso financeiro e buscar soluções junto aos desenvolvedores.

Em negociação com o provedor de internet ficou acordado que eles deverão garantir, no mínimo, de 56kbps (para upload e download) para os horários de pico e de 90% de disponibilidade de internet por dia, limitadas a uma interrupção de 8640s ou, no máximo, 10 interrupções de 864s cada.

Analisando o problema de conexões, o gerente de TI constatou que a quantidade de usuário em conexão simultânea foi estimada inicialmente para o máximo de 50, de acordo com o número máximo de funcionários na empresa, mas que no primeiro dia de produção o sistema alcançou um número superior a 150 conexões simultâneas, impactando no tempo de processamento e no consumo de banda da internet. Como esta infraestrutura é compartilhada com outros sistemas e sua capacidade já encontra-se no máximo de uso, ampliar os recursos implica em adquirir um servidor exclusivamente para este sistema, solução descartada pelo deparamento financeiro quando solicitado aporte para este projeto de ampliação.

No caso dos relatório, o gerente de TI, constatou que o algoritmo desenvolvido para o processamento destes relatórios poderia ser melhorado, mas o tempo necessário para analisar, projetar, implementar e testar o novo algorítimo poderia ser maior do que o aceitável pelos usuários do sistema.

Diante deste cenário, o gerente de TI solicitou aos desenvolvedores do sistema que analisassem e projetassem uma solução que atendessem a limitações financeira e de infraestrutura com o objetivo de, pelo menos: garantir que cada mensagem seja enviada, processada e respondida pelo servidor, que o tempo de espera dos clientes para processar os relatórios seja reduzido e que seja ampliado a sensação de disponibilidade deste sistema.

2.2 Análise e Solução

Neste cenário, garantir que uma mensagem seja enviada, processada e respondida significa que o sistema deverá ser tolerante a falta de conexão e que as interrupções afetem o mínimo possível esta garantia.

Considerando que o tempo de processamento mínimo para os relatórios é de 10 minutos, uma interrupção, no pior caso, de 8640s (144 minutos), poderia fazer com que o usuário aguardasse, no mínimo, cerca de duas horas e meia por uma resposta.

Como não é possível controlar o pior cenário de interrupção, dentre outras, as soluções possíveis são:

- aceitar e tentar reduzir apenas o tempo de processamento ao máximo;
- estocar dados parcialmente em um cache do lado do cliente para providenciar simplesmente relatórios;
- processar os relatórios previamente do lado do servidor para que as solicitações sejam rapidamente atendidas.

A primeira solução não elimina o pior caso, ou seja, os efeitos de uma interrupção ainda sim seriam sentidas pelo sistema e este ficaria indisponível para processar ou obter resposta

por um longo período de tempo. A última solução, que poderá quase que eliminar o tempo de processamento dos relatório, seria excelente se não fosse pelo problema de interrupção, que o anula caso isto ocorra com frequência. A melhor solução, dentre estas, é a de armazenar apenas os dados necessários para produção do relatório em um local temporário e mantê-lo sincronizado sempre que houver conexão com o servidor.

Esta solução pode ser aproveitada para garantir que uma mensagem seja realmente entregue, processada e respondida pelo servidor mesmo quando estiver offline, aumentando ainda mais a sensação de que o sistema está disponível. Isto pode ser feito adotando um gerenciador de envio e resposta, que ficará responsável por manter a comunicação com o servidor e isolar o cliente de todas as falhas de rede que possam existir.

Para evitar que o sistema fique lento, o número máximo de conexões permitidas pelo servidor será limitada a 50, logo as demais conexões deverão ser rejeitadas. Para evitar que os componentes clientes sintam o efeito dessa rejeição, as conexões serão divididas da seguinte forma:

- pelo menos 1 conexão exclusiva para sincronizar dados de tempos em tempos;
- pelo menos 1 conexão exclusiva para realizar tarefas de background com o servidor e
- no máximo de tempo de 1s para cada conexão aberta entre um cliente e o servidor, podendo cada cliente repetir a solicitação de uma requisição somente depois de 5 segundos se a anterior foi realizada com sucesso ou imediatamente se for a primeira ou se a anterior foi rejeitada.

2.3 Projeto

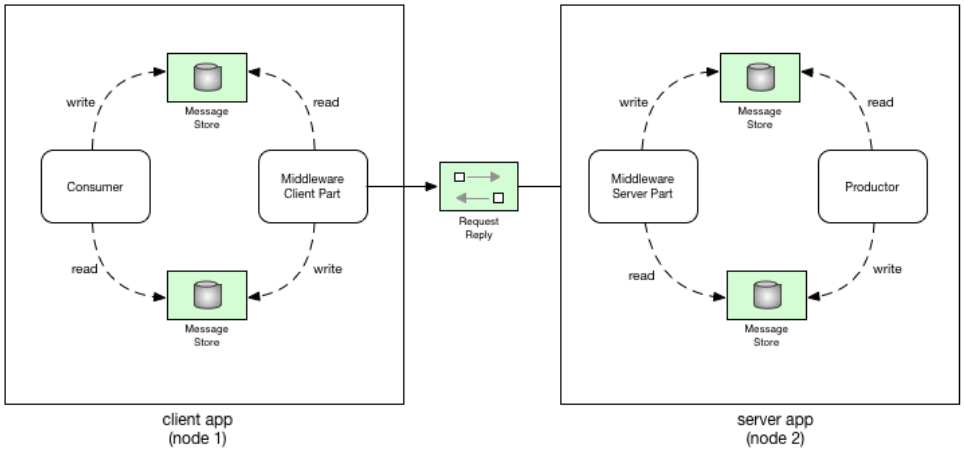

Figura 2: *Projeto inicial da solução de garantia da entrega de mensagens.*

Os alunos deverão implementar e resolver este cenário, descrevendo a sua solução

Referências

Android Developers. Api guide: Supporting multiple screens. https://developer.android.com/guide/practices/screens_support.html, 2015. Acessado em 01/fev/2015.

Statista. Cumulative number of apps downloaded from the google play android app store as of july 2013 (in billions). http://www.statista.com/statistics/281106/number-of-android-app-downloads-from-google-play/, julho 2013. Acessado em 01/fev/2015.