Arquitectura de um Sistema de Chamadas a Procedimentos Remotos a Servidores Replicados

Pedro Vicente
pedrofrv@di.fc.ul.pt

João Martins
jmartins@di.fc.ul.pt

Abstract

O paradigma das chamadas a procedimentos remotos é hoje em dia largamente utilizado na construção de aplicações distribuídas. Recentemente os sistemas de comunicação em grupo têm vindo a facilitar a construção de sistemas de alta disponibilidade baseados numa arquitectura de grupos de processos cooperantes. Uma possível aplicação destes sistemas é a construção de um servidor para chamadas a procedimentos remotos replicado, dotando este modelo tradicional com uma maior tolerância a faltas. Este artigo descreve uma possível concretização de tal sistema.

1. Introdução

As chamadas a procedimentos remotos (remote procedure calls ou RPC) são uma das formas mais utilizadas de construír aplicações distribuídas. As RPC's funcionam enviando um pedido pela rede a um servidor, que o executa e envia a resposta ao cliente. Este modelo é conhecido como cliente / servidor.

Embora largamente utilizado, este modelo tem uma falha do ponto de vista da tolerância a faltas: se o servidor falha, então toda a computação pára.

Recentemente têm vindo a ser desenvolvidos protocolos de comunicação em grupo [4, 3] que facilitam a construção de sistemas que fornecem serviços com elevado grau de disponibilidade recorrendo ao uso de servidores replicados.

Usando o paradigma de comunicação em grupo podem-se construir sistemas de servidores replicados que fornecem um interface igual ao das chamadas a procedimentos remotos, mas com a vantagem de o servidor não ser apenas um único processo mas sim um conjunto de processos replicados, guarantindo uma maior disponibilidade do sistema.

Este relatório está organizado da seguinte forma: na secção 2 são descritos os dois modelos de replicação que irão ser considerados. Na secção 3 é descrita a arquitectura do sistema. Na secção 4 é descrita a estratégia de concretização da replicação activa. Na secção 5 é descrita a implementação da replicação passiva. Na secção 6 são descritas as aproximações possíveis para o problema da reintegração de servidores.A secção 7 conclui este artigo.

2. Modelos

Um dos modos de construir sistemas que fornecem continuidade de serviço é através de redundância espacial sob a forma de replicação.

Usando este modelo, e assumindo a existência de um sistema de comunicação em grupo, existem várias técnicas para gerir as várias réplicas. Neste artigo iremos estudar e propor uma concretização de duas destas técnicas, a replicação activa e a replicação passiva. As primitivas de comunicação utilizadas são brevemente descritas na secção a seguir, seguidas pela descrição das técnicas de replicação referidas.

2.1. Primitivas de comunicação

Comunicação ponto-a-ponto Permite a comunicação entre os clientes e os servidores usando UDP em conjunto com uma camada que oferece FIFO [1].

Comunicação em Grupo Permite a comunicação entre os elementos do grupo usando Broadcast ou Multicast.

Sincronia Virtual Uma vista representa o conjunto de processos que fazem parte de um grupo.Quando existe uma mudança na filiação do grupo é entregue uma nova vista a todos os elementos. A sincronia virtual garante que numa certa vista todos os processos "correctos" recebem o mesmo conjunto de mensagens.

Difusão Atómica ou Ordem Total Propriedade que garante a entrega das mensagens a todos os membros "correctos" do grupo pela mesma ordem.

[1] As mensagens de um processo são recebidas pela ordem que são enviadas

2.2. Replicação Activa

É importante referir para este modelo irão ser considerados apenas servidores deterministas, isto é, cada servidor se comporta como uma máquina de estados.

Este modo de replicação pode ser aplicado facilmente a servidores deterministas. Consiste em ter várias réplicas do mesmo servidor a serem executadas em diferentes processos. Para assegurar a consistência do estado destas réplicas, cada um dos comandos é disseminado por difusão atómica para todas as máquinas. Isto assegura que todas as réplicas recebem os mesmos comandos na mesma ordem, o que implica que produzem os mesmos resultados.

Estender este modelo para as chamadas a procedimentos remotos é conceptualmente simples. O cliente contacta um dos servidores do grupo com o seu pedido, que se encarrega de o disseminar usando a ordem total. Cada um dos servidores, depois de executar o pedido, responde ao cliente. A camada de RPC do cliente encarrega-se de descartar as respostas duplicadas e retornar uma única resposta à aplicação cliente.

2.3. Replicação Passiva

Este modelo é diferente do modelo anterior principalmente na forma como são actualizadas as réplicas. Enquanto que no modelo de replicação activo a operação é realizada em todas as réplicas neste modelo apenas existe uma cópia do estado.

Um dos elementos do grupo de servidores é eleito para ser o primário. Este processo fica encarregue de efectuar as operações e transmitir o novo estado a todas as réplicas.

As réplicas secundárias apenas terão a responsabilidade de alterar o estado quando recebem as mensagens do servidor primário.

A cópia de estado tem a vantagem de permitir a execução de código não determinista, isto é, código que pode dar resultados diferentes dependendo do local onde é executado[2].

No caso do estado da aplicação ser mais extenso que os parâmetros dos procedimentos então esta aproximação terá mais necessidades em termos de largura de banda do que o primeiro modelo.

3. Arquitectura

O projecto proposto vai ser desenvolvido sobre o Appia[2][3], um sistema de comunicação baseado en microprotocolos desenvolvido na Faculdade de Ciências. Para o desenvolvimento do sistema é necessário podermos contar com as primitivas descritas atrás, fornecidas por protocolos já construidos para o Appia.

[2]código que depende de relógios, números aleatórios, etc

[3]`http://appia.di.fc.ul.pt`

O sistema pode ser decomposto em duas partes: clientes e servidores. Os clientes comunicam com os servidores usando comunicação ponto-a-ponto com um dos elementos do grupo. Este elemento envia os pedidos totalmente ordenados para os restantes elementos usando comunicação em grupo. Os clientes possuem duas camadas que implementam a parte genérica do sistema (RRPC) e a parte específica da aplicação cliente (Client). Estas camadas ficam em cima de uma pilha de protocolos que fornece FIFO e comunicação UDP. Os servidores também possuem uma camada genérica (Coordination) e uma camada específica da aplicação (Server). Estas camadas ficam em cima de uma pilha de protocolos que oferece: ordem total, comunicaçao em grupo, FIFO , UDP e comunicação ponto-a-ponto com os clientes. Outro ponto importante do sistema é a existência de servidores de gossip. Estes são responsáveis por manter a filiação dos grupos, permitindo tanto a entrada de novos elementos bem como o regresso de membros antigos.

4. Concretização da Replicação Activa

Nesta secção é descrita a estratégia de concretização do modelo de replicação activa aplicado às invocações remotas. Dado que este modo de replicação é transparente para as camadas Cliente e Servidor, descrevemos apenas o funcionamento das camadas RRPC e Coordination, que gerem a replicação do lado do cliente e do servidor, respectivamente.

4.1. Camada RRPC

A camada RRPC irá ser responsável por procurar um servidor no grupo ao qual entregar o pedido do cliente, usando o evento `RemoteViewEvent`. Esta camada terá também de atribuir um identificador único a cada pedido, usando para esse efeito o endereço do cliente e um número de sequência (i.e., um identificador de pedido é um tuplo $< end, nseq >$).

Será também nesta camada que será feita a triagem das respostas. Dado que na replicação activa todos os servidores respondem ao pedido, o cliente receberá até n respostas, sendo n o número de servidores. O que é feito neste caso é o cliente devolver a primeira resposta que recebe, e descartar as seguintes.

As respostas repetidas são descartadas da seguinte forma: é mantida uma lista dos pedidos efectuados. De cada vez que é feito um pedido novo é acrescentado a esta lista. Quando é recebida uma resposta, se o pedido correspondente existe na lista, a resposta é passada para cima, e o pedido é restirado da lista. Se o pedido correspondente não existe na lista, a resposta é descartada.

Esta camada tem também a responsabilidade de reenviar pedidos a outros servidores caso o primeiro servidor con-

tactado não responder. É mantida uma lista com os pedidos enviados. Periódicamente (2 em 2 segundos), a lista é percorrida, e os pedidos que nela constam são reenviados para um servidor diferente do último reenvio. Quando chega uma resposta, o pedido correspondente é retirado desta lista.

4.2. Camada Coordination

Esta camada será responsável por ordenar os pedidos dos clientes. Para tal, cada pedido não ordenado recebido de um cliente é enviado pela ordem total para o grupo de servidores. A camada servidor atende apenas pedidos ordenados.

Para a camada servidor saber a quem é que deve responder, a camada de coordenação acrescenta o endereço do cliente ao "payload" do pedido antes de o enviar pela ordem total.

A camada de coordenação acrescenta o endereço do cliente ao corpo da mensagem. Quando o pedido retorna já ordenado, a camada de coordenação gera um pedido para o servidor colocando como origem o endereço do cliente. Deste modo a replicação é completamente transparente para o servidor, porque o pedido é idêntico a um pedido não replicado.

É também nesta camada que são mantidas as respostas dadas aos clientes. Dado que cada pedido tem um identificador único, se um cliente fizer o mesmo pedido duas vezes (porque o cliente desistiu de esperar pela resposta da primeira tentativa), então a resposta armazenada nesta camada é fornecida ao cliente, e o pedido não é reexecutado. A eliminação de elementos desta lista de respostas é feita associando um alarme com um certo tempo de disparo (40 segundos) a cada resposta, findo o qual essa resposta é descartada.

É de notar que o valor deste tempo de disparo tem um efeito directo no grau de tolerância a faltas que o sistema irá suportar: sendo d o tempo que demora o pedido do cliente a chegar ao servidor (e o tempo que demora a resposta a voltar ao cliente, assumindo uma rede simétrica), e t_c o tempo que o cliente espera até reenviar o pedido, se o timeout do servidor for de $n(t_c + 2d)$, podem ser toleradas até $n - 1$ faltas de omissão da rede. Na concretização consideramos $d = 1$, $t_c = 2$ como referido acima e $n = 2$, pelo que temos um tempo de disparo de $2(2 + 2(1)) = 8$ segundos.

5. Implementação da replicação passiva

O método de replicação passiva pode ser construido recorrendo apenas a alteração do código do lado do servidor, permitindo assim uma maior flexiblidade.

Se uma aplicação necessita de realizar operações não deterministas, basta colocar os servidores a funcionar em modo de replicação passiva.

É de salientar que a replicação passiva não necessita de ordem total.

A replicação passiva funciona do seguinte modo:

1. O cliente faz o pedido aos servidores, até obter uma resposta
2. Se o servidor obter o pedido pela primeira vez e não for primário envia o pedido para todos os elementos do grupo.
3. Se o pedido é repetido envia a resposta que já tem.
4. Quando o primário recebe um pedido processa-o, e envia a resposta e a actualização do estado a todos os membros do grupo.
5. O primário envia a resposta ao cliente.

Nas seccções seguintes são descritos os pormenores da implementação das camadas servidor e coordenação.

5.1. Camada Server

Com este método de replicação é necessário ter em conta que as operações não são efectuadas em todos os servidores. Para os servidores manterem a consistência são feitas cópias de estado entre o servidor primário e os servidores secundários. Como o estado da aplicação não é um aspecto genérico do sistema é necessário ser a própria camada da aplicação a fornecer esse informação. Este mecanismo tem a desvantagem de obrigar a criar, para cada aplicação, um método que produza uma actualização ao estado a pertir da execução de um pedido.

Além de copiar o estado o servidor primário tem também de enviar a resposta a todos os outros servidores, para que estes possam enviá-la ao cliente.

Para se reutilizar o código feito não foram criadas versões diferentes da camada servidor para a replicação passiva, apenas foi acrescentado um método para produzir uma actualização de estado. Quanto à camada RRPC é a mesma, como referido acima.

Assim decidiu-se criar um campo nos pedidos que caso venha preenchido indica a esta camada que deve criar uma actualização de estado. A camada além de calcular e enviar a resposta cria um objecto onde se encontra o estado actual da aplicação. Este objecto só é interpretado pela camada coordenação sendo totalmente transparente para as outras camadas.

5.2. Camada Coordination

A maioria das mudanças foi feita na camada Coordination. Tal como no modo de replicação activa todos os servidores guardam as respostas antigas dos clientes. Esta

prática evita a execução de pedidos repetidos bem como o envio imediato da resposta ao cliente, independente do tipo de servidor.

Os servidores funcionam de modo diferente caso estejam em modo primário ou secundário.

Se o servidor está a funcionar em modo secundário apenas tem de reencaminhar os pedidos novos para o primário e enviar os eventos de actualização de estado para a camada servidor.

Este eventos de actualização de estado têm de ser entregues à aplicação na ordem pela qual foram executados no servidor primário. Para isso estes eventos levam um número de sequência que permite a entrega pela ordem correcta. Além do estado actualizado estes eventos levam a resposta ao pedido para os servidores poderem continuar a responder aos clientes, já que em caso de falha do primário os pedidos são feitos a um servidor secundário.

Se o servidor está a funcionar como servidor primário pode receber pedidos dos clientes, pedidos dos outros servidores ou respostas da camada Server.

Os pedidos dos clientes e dos outros servidores, caso sejam novos, são passados para a camada cliente.

As respostas dos clientes são enviadas de volta ao cliente e é gerado o evento de mudança de estado para ser enviado para o grupo de servidores.

6. Reintegração

Nas secções acima foi ignorado o problema da reintegração de servidores. Embora a replicação dos servidores providencie um certo grau de tolerância a faltas, a reintegração de um servidor falhado e que entretanto recuperou deve ser contemplada, de modo a repor o grau de tolerância a faltas original.

Dado que apenas a aplicação sabe em que consiste o estado de um servidor, este problema é naturalmente específico da aplicação, pelo que qualquer protocolo terá de ter a intervenção desta. No entanto este processo pode ser facilitado se existir um método delineado.

Este problema já foi abordado no contexto de bases de dados replicadas que usam comunicação em grupo [1]. Nesse artigo os autores apresentam diversas formas de efectuar a integração de um servidor recuperado.

Uma dessas formas (a mais simples) foi adaptada e concretizada na pequena aplicação exemplo desenvolvida para demonstrar o funcionamento das RPC's sobre grupos.

De modo a simplificar o problema, não são consideradas partições. As partições são um caso problemático, porque implicam ou a existência de métodos de reconciliação de dados (caso duas partições progridam independentemente) ou o conhecimento do número total de servidores, de modo a permitir progresso apenas na partição maioritária.

Se descartarmos a hipótese de partições, um servidor pode estar num de três estados:

clean quando o servidor acabou de arrancar e não atendeu quaisquer pedidos.

dirty quando o servidor arrancou e permaneceu numa vista isolado a atender pedidos.

old quando o servidor pertencia à vista anterior.

Note-se que não é possível termos dois servidores dirty, por exemplo, já que se existirem dois servidores a funcionar, então estão na mesma vista. No caso *old*, é detectado o facto de o servidor provir ou não de uma vista inicial (a vista inicial é a primeira vista que um servidor recebe quando arranca, e que contem apenas o próprio servidor); se tal acontecer então o servidor está num dos outros dois estados.

Na descrição que se segue, consideramos que existe um novo membro S_n que se juntou à vista V, provocando a nova vista W.

1. Quando é entregue uma nova vista W, cada servidor manda ao coordenador da vista W o seu estado em relação à replicação: *clean, dirty ou old*. Um servidor é *clean* quando acabou de iniciar, é dirty quando acabou de iniciar mas já processou pedidos, e é *old* se pertencia à vista anterior V. Usando esta informação, o coordenador elege um membro S_p pertencente à antiga vista V para ser o *par* do servidor S_n que se juntou ao grupo.

2. O processamento de pedidos continua como normalmente em todos os servidores, excepto em S_p e S_n.

3. S_p transfere o seu estado para S_n, incluindo todas as modificações efectuadas por pedidos que foram entregues *antes* de ocorrer a mudança para a nova vista W.

4. Enquanto a transferência de estado ocorre, tanto S_n como S_p vão colocando todos os pedidos que recebem numa fila de espera.

5. Quando a transferência de estado é concluída, S_n e S_p executam os pedidos em fila de espera, pela ordem que foram entregues.

É de salientar que este esquema não contempla falhas durante o processo de reintegração, e que implica que dois nós (o reintegrado e o par) atrasem o processamento de pedidos até a transferência de estado estar concluída.

Os autores do artigo citado acima descrevem modos mais avançados de efectuar estas mudanças de estado, que, tal como o método acima, poderiam ser adaptados à aplicação que iria utilizar os RPC's sobre grupos.

7. Conclusão

Neste artigo descrevemos a arquitectura de um sistema de chamadas a procedimentos remotos a servidores replicados. Foram considerados dois modos de replicação, activa e passiva, e descritas as linhas gerais de implementação desses modos. Considerámos também o problema da reintegração de servidores falhados, e apontámos uma solução possível que se adapta aos modelos em causa.

Referências

[1] A. Bartoli, B. Kemme, and O. Babaoglu. Online reconfiguration in replicated databases based on group communication. Technical Report UBLCS-2000-17, University of Bologna, Dec. 2000.

[2] H. Miranda, A. Pinto, and L. Rodrigues. Appia, a flexible protocol kernel supporting multiple coordinated channels. In *Proceedings of the 21st International Conference on Distributed Computing Systems*, pages 707–710, Phoenix, Arizona, Apr. 2001.

[3] D. Powell et al. Group communication (special issue). *Communications of the ACM*, 39(4):50–97, Apr. 1996.

[4] R. van Renesse, K. P. Birman, and S. Maffeis. Horus, a flexible group communication system. *Communications of the ACM*, Apr. 1996.

[5] P. Veríssimo and L. Rodrigues. *Distributed Systems for Systems Architects*. Kluwer Academic Publishers, 2001.