ESCRITAS E LEITURAS EM REGISTO DISTRIBUIDO TOLERANTE A FALTAS

Irina Clington 27976
27976@di.fc.ul.pt

Daniel Mendes 29376
i29376@di.fc.ul.pt

Moisés Xavier 25533
25533@di.fc.ul.pt

Abstract

Vemos, nos dias de hoje, uma grande dificuldade em aceder a um registo que está replicado em várias máquinas. Há que disseminar eficazmente as escritas por todos os registos para que as leituras não sejam incoerentes. Neste artigo iremos apresentar uma solução difusão best-effort e uma solução difusão atómica.

1 Introdução

Com a evolução da informática e o seu aumento de utilização, os modelo cliente-servidor, em que só existia um servidor central, onde os serviços estavam concentrados, começaram a ficar obsoletos, uma vez que o uso de um só servidor para vários clientes começava a implicar aperda de determinadas propriedades imprescindíveis neste tipo de arquitecturas.

Referimo-nos por exemplo à disponibilidade e a salvaguarda de informação. Sendo um serviço que deve estar sempre disponível, num único servidor onde estão centralizados todos os serviços isso nem sempre é possível de oferecer. Os motivos podem ser os mais variados, falha do sistema, sobrecarga de pedidos a tratar ou congestionamento da rede. A salvaguarda da informação também era um problema crítico, pois era necessário parar o sistema para fazer uma cópia de segurança, caso acontecesse algo à informação do servidor num certo instante, não havia uma cópia actual para a substituir.

Tendo presentes estes pressupostos e a problemática envolvente, apareceu então a solução dos servidores replicados, havendo assim várias réplicas a prestar o mesmo serviço e a manter a informação coerente nas várias máquinas.

Como em qualquer sistema distribuído, o nosso sistema pretende ser fiável, transparente, com bom desempenho e tolerante a faltas, ou seja, prever que algo possa acontecer a uma máquina (uma falha) e mesmo assim o sistema continuar a funcionar, fornecendo os mesmos serviços aos seus utilizadores. O sistema, neste caso em concreto, o servidor replicado deve ser tolerante a faltas.

Com este artigo pretendemos mostrar o desenvolvimento de um servidor replicado que forneça registos ao seus clientes, replicados em várias máquinas, mantendo um estado coerente. Pretende-se que seja tolerante a faltas e que ofereça dois tipos de funcionamento, propagação melhor-esforço e propagação atómica.

Propagação melhor-esforço - a diferença de tempo que pode existir entre a actualização das várias réplicas pode ser visível, ou seja, dois clientes que contactam replicas diferentes podem obter valores dispares do mesmo registo.

Propagação atómica - após uma escrita, se alguém obter o novo valor do registo, mais ninguém a partir desse momento vai obter o valor antigo.

Para esta concretização vamos utilizar uma arquitectura de comunicação configurável capaz de suportar reutilização e composição de componentes.

Referimo-nos ao APPIA [1] que é um núcleo de protocolo que tenta balancear a flexibilidade na composição de protocolos com a eficácia em tempo de execução. O APPIA [2] acaba por ser uma ferramenta em camadas de suporte à comunicação que nos vai facilitar a comunicação em grupo e a difusão fiável de mensagens entre eles.

Este artigo encontra-se estruturado da seguinte forma; secção na secção 2 está descrita a arquitectura do sistema, a secção 3 aborda a noção de ordenação total, a secção 4 descreve o sincronismo virtual, a secção 5 descreve a pilha protocolar que vai ser usada na implementação do servidor replicado. A secção 6 aborda o modelo de faltas, descrevendo os tipos, o seu enquadramento com o nosso trabalho e as soluções propostas pelo grupo. Na secção 7 retratamos o funcionamento do algoritmo que pretendemos desenvolver para implementar a propagação best-effort e a propagação atómica.

Por ultimo mas não mais importante temos na secção 8 a bibliografia usada.

2. Arquitectura

Considerámos 2 possibilidades para a arquitectura em questão. Uma em que os clientes comunicam com os servidores usando comunicação ponto-a-ponto com um dos el-

ementos do grupo de réplicas do servidor. Este elemento envia os pedidos totalmente ordenados para os restantes elementos usando comunicação em grupo, ou seja, usando RPC. Acabámos por descartar esta solução, apresentada pelos nossos colegas [3], porque o cliente teria que guardar a localização de vários servidores e teria que ir tentando até que um destes lhe desse resposta.

Optámos por uma solução em que o cliente é ao mesmo tempo cliente e servidor. Assim sendo existe um grupo de servidores, quando um cliente pretende aceder ao registo junta-se ao grupo. Passa a pertencer ao grupo de servidores o que lhe permite manter o seu registo actualizado, e é também um cliente pois faz pedidos de leitura e escrita. Neste caso a falha do servidor é a falha do seu próprio computador e só terá que ser tratada pelos outros elementos do grupo de forma a que esta não provoque nenhuma incoerência.

3 Ordenação Total

Para o bom funcionamento do algoritmo é necessário garantir a entrega das mensagens a todos os membros correctos pela mesma ordem. Para este fim iremos utilizar a TotalAbstractLayer oferecida pelo APPIA [4]. Um exemplo de extrema utilidade da ordem total é o caso em que chegam duas mensagens de escrita. Optámos por apenas permitir uma delas(a primeira). Caso uns descartassem a primeira e outros a segunda seria impossível obter coerência.

4 Sincronismo Virtual

Para lidar com o facto de haver entradas no grupo de utilizadores que pretendam aceder aos registos e saídas por utilizadores que não estejam mais interessados em aceder aos registos e a falha de um membro do grupo, contaremos com o sistema APPIA [2].

O APPIA oferece-nos a funcionalidade de gestão de grupos com recurso a técnicas de sincronismo virtual. A sincronia virtual é um paradigma da comunicação em grupo que estipula que:

todos os participantes de um grupo vêm a sua constituição em forma de vistas;

os participantes de um grupo vêm as mesmas mudanças de vista pela mesma ordem;

todas as mensagens de uma vista são entregues nessa vista;

antes de uma mudança de vista, todas as mensagens entregues a um membro são entregues aos outros.

5 Pilha Protocolar

No Appia as propriedades suportadas pelo canal de comunicação são definidas com base nos protocolos utilizados e denomina-se Qualidade de Serviço (QoS).

Apresentamos em seguida a pilha protocolar:

ApplicationLayer
TotalAbcastLayer
VSyncLayer
StableLayer
HealLayer
InterLayer
IntraLayer
SuspectLayer
MergeOutLayer
GossipOutLayer
GroupBottomLayer
TCPCompleteLayer

Todas as camadas excepto a ApplicationLayer são fornecidas pelo Appia. A camada debaixo especifica o protocolo de transporte. Os 10 protocolos acima especificam comunicação em grupo com sincronismo virtual e ordem total. A camada de cima especifica a aplicação. A descrição dessas camadas pode ser encontrada em [6] [5].

6 Modelo de Faltas

Tratando de um servidor replicado e como tal um sistema distribuído, temos de ser tolerantes a faltas com intuito de dar fiabilidade ao sistema.

6.1 Tipos de faltas [7]

Existem dois grandes grupos de faltas que podem ocorrer em sistemas distribuídos, são elas as omissivas e as afirmativas.

As faltas omissivas existem no domíni-o do tempo e estão relacionadas com o facto de um componente não efectuar a acção que normalmente realiza. As faltas omissivas subdividem-se em faltas de omissão, faltas temporais e as faltas por paragem.

As faltas de omissão acontecem quando um componente omite uma acção, por exemplo, a recepção de uma mensagem. As faltas temporais surgem devido ao facto de um componente se atrasar na realização de uma acção.

As faltas por paragem estão relacionadas com a paragem do componente.

Por sua vez, as faltas afirmativas existem no domínio do valor e subdividem-se nas faltas semânticas e nas faltas sintácticas.

As faltas sintácticas são possíveis de detectar, pois nesse caso a mensagem trocada não está sintacticamente correcta, ou seja, as mensagens não respeitam as normas de criação e/ou os intervalos de valores válidos.

Por outro lado, as faltas semânticas estão relacionadas com o facto das mensagens estarem sintacticamente correctas, mas não semanticamente, ou seja, respeitam as normas de criação mas pode ter valores absurdos ou somente incorrectos.

6.2 Enquadramento

No sistema de servidor replicado que vamos desenvolver este tipo de faltas pode causar graves falhas no seu desempenho e como tal, consequências imprevisíveis.

Podemos dar como exemplo algumas situações em que estas faltas podem colocar em causa o nosso sistema, nas faltas por paragem temos o caso de uma máquina parar aquando de uma escrita, não sabemos então se ela chegou a escrever e no pior caso se já mostrou essa actualização a alguém. Nas faltas temporais temos que um servidor pode demorar muito tempo a enviar uma mensagem e assim atrasar toda a interacção do grupo. Nas faltas por omissão pode ocorrer que o componente omita a recepção de uma mensagem de escrita, podendo ficar num estado incoerente relativamente às restantes réplicas.

Relativamente às faltas no domínio do valor, temos no caso das faltas sintácticas estarmos esperando o número do registo (um inteiro) para escrita ou leitura e recebermos um caracter ou então recebermos um número superior ao número de registos. Por sua vez no caso das faltas semânticas, temos a situação do servidor responder que no registo x está o valor y e ser mostrado ao cliente um valor w, que é um valor válido, mas não o correcto. Temos de estar preparados para estas situações, de forma a assegurar as propriedades que os utilizadores esperam de um serviço deste género.

6.3 Soluções e Detecções

Comecemos pelas faltas omissivas. Não nos preocuparemos muito com as por omissão e paragem porque contamos com o sistema APPIA, através do mecânismo de sincronismo de vistas para tratar das saídas do grupo e que também garante que quando um membro do grupo recebe a mensagem todos os outros também a receberão. Não existe grande problema com as faltas temporais pois como temos uma camada APPIA que fornece ordenação total, independentemente de quando chegarem sabemos que chegarão a todos e pela mesma ordem.

Em relação as faltas afirmativas, para as sintácticas faremos as verificações necessárias para garantir que estas respeitam o formato desejado. No caso das semânticas e tendo consciência de serem as mais difíceis de detectar e consequentemente de tratar, propomos para a sua resolução o envio junto da mensagem do seu hash, desta forma saberemos se os dados estão ou não correctos.

Para além disto a forma como o algoritmo está escrito e a sincronia virtual não permitem que hajam incoerências na base de dados, quando uma maquina morre, dentro das duas formas de propagação das actualizações que nos propusemos a implementar.

7 Funcionamento do Algoritmo

Uma vez que o APPIA funciona por eventos explicaremos os eventos dos quais necessitaremos para pôr o algoritmo a funcionar e como se passará a interacção com o utilizador.

7.1 Eventos e Repostas

7.1.1 Propagação best-effort

Perante um evento de escrita W(valor, posição):

Caso quem lançou o evento pertença à vista ac tual actualiza-se o registo na posição posição com valor valor e envia-se um evento que corresponde a um acknoledge A();

Perante um evento A():

Adiciona-se aquele membro à lista de acks recebidos e quando os membros na lista de acks for igual à vista actual, responde-se ao utilizador que a operação foi bem sucedida

Perante um evento do APPIA com uma nova vista:

Actualiza-se a vista para que possamos saber sempre de quantos acks estamos à espera.

7.1.2 Propagação atómica

Perante um evento de escrita W(valor, posição):

Este evento é visto como uma proposta. Assim sendo caso a proposta esteja a -1, valor considerado inválido, altera-se a proposta para este valor e guarda-se, o membro que o propôs e a posição do registo. Caso o valor da proposta seja diferente de -1 deveremos primeiro ver com a vista actual se quem propôs o valor se encontra ainda entre nós. Caso se encontre, descartamos o pedido ou se formos nós quem o tivermos enviado respondemos ao utilizador que não é possivel escrever de momento naquele registo. caso não se encontre, alteramos a proposta para este novo valor. Sempre um membro receber a proposta que ele próprio colocou, e esta esteja em condições de ser actualizada, envia um evento WD().

Perante um evento de decisão de escrita WD():

Caso quem propôs o evento se encontre vivo actualiza-se o registo na posição posicao com valor valor, guardada anteriormente e envia-se um evento de que corresponde a um acknoledge A(); Caso contrário actualiza-se a proposta para -1.

Perante um evento do APPIA com uma nova vista:

Actualiza-se a vista para que possamos saber sempre de quantos acks estamos à espera.

Perante um evento A():

Adiciona-se aquele membro à lista de acks recebidos e quando os membros na lista de acks for igual à vista actual, responde-se ao utilizador que a operação foi bem sucedida

7.2 Interacção com o utilizador

7.2.1 Propagação best-effort

Caso o utilizador nos faça um pedido de escrita:

Lançamos um evento de escrita e tratamo-lo de acordo com o que está acima descrito e respondemos ao utilizador se a operação teve ou não sucesso.

A operação só não tem sucesso se este morrer. No segundo caso a escrita pode ou não ter sucesso dependendo de como o APPIA ordena as falhas do membro em relação à mensagem para a nova vista, de qualquer forma, não é muito relevante para o morto o sucesso ou insucesso da escrita.

Caso o utilizador nos faça um pedido de leitura:

Lemos o valor do registo na posição que o utilizador quer e devolvemos-lhe esse valor.

7.2.2 Propagação atómica

Caso o utilizador nos faça um pedido de escrita:

Lançamos um evento de escrita e tratamo-lo de acordo com o que está acima descrito e respondemos ao utilizador se a operação teve ou não sucesso.

A operação só não tem sucesso se estiver a ocorrer uma escrita ou se este morrer. No segundo caso a escrita pode ou não ter sucesso dependendo de como o APPIA ordena as falhas do membro em relação à mensagem para a nova vista.

Caso o utilizador nos faça um pedido de leitura:

```
If(proposta !=-1 && sender  E vistaActual){
Proposta = -1;
printf(o valor da posição %d é %d, posicao, registo[posicao]);
}
else if(proposta ==-1){
printf(o valor da posição %d é %d, posicao, registo[posicao]);
}
else{
printf(posicao do registo em actualização tente novamente
mais tarde!);
}
```

References

[1] H. Miranda, A. Pinto, L. Rodrigues, *Appia*, a flexible protocol kernel supporting multiple coordinatedchannels

[2] H. Miranda, A. Pinto, L. Rodrigues, Application Program Interface Specification of *Appia*, 2001

[3] M. Monteiro, S. Teixeira, Tolerância a faltas em jogos de computador multi-utilizador

[4] N. Carvalho, L. Rodrigues, Implementing Reliable Broadcast Protocols in *Appia*, 2003

[5] A. Pinto, "*Appia* Group Communication Manual", 2001

[6] http://appia.di.fc.ul.pt

[7] P. Verríssimo L. Rodrigues, "Distributed Systems for System Architects", Kluwer Academic Publishers ISBN 0-7923-7266-2