Tulkas - Sistema de Publicação/Subscrição Tolerante a Faltas

Edgar Pascoal, André Luís, Ivo Guimarães
{TFD04}
Departamento de Informática - FCUL
Universidade de Lisboa
{edgar.pascoal,andr3.pt,ivouva}@gmail.com

Abstract

Com o aumento da importância do papel desempenhado pelos sistemas computacionais, nomeadamente sistemas distribuídos, revela-se cada vez mais uma necessidade a consideração de técnicas de tolerância a faltas para que um dado serviço não sofra rupturas e continue a funcionar mesmo que um elemento, dito crucial, falhe. Uma vez que há imensas aplicações para sistemas baseados em eventos e um elemento principal destes são sistemas de Publicação/Subscrição vamos explorar ao longo deste artigo uma solução possível para tornar estes sistemas Pub/Sub tolerantes a faltas.

1. Introdução

Os sistemas baseados em eventos têm crescido em diversas áreas, desde sistemas time-critical, passando por sistemas de controlo e gestão, até e-commerce. Estes, são sistemas em que as notificações entre participantes são feitas por eventos. Os pontos fortes dos mesmos estão associados à independência entre módulos, à possibilidade de integrar componentes incompatíveis e à escalabilidade inerente. São geralmente usados para dar suporte a: sistemas de streaming; vigilância e sistemas de monitorização; continuous query systems [3].
Um dos paradigmas mais usados para conceber este tipo de sistemas é denominado "Publish/Subscribe" [2]. Este paradigma consiste em Produtores e Subscritores. Os consumidores têm como seu papel manifestar interesse em receber determinado tipo de eventos e os produtores em publicar eventos para os consumidores interessados. O papel correcto de um sistema destes é garantir a coerência entre todos os participantes, nomeadamente na disseminação de eventos, isto é, se um "Produtor" produz uma mensagem, esta é entregue atempadamente a todos os "Consumidores" interessados e pela mesma ordem.
Este paradigma, no entanto, não prevê quaisquer mecanismos de tolerância a faltas, o que poderia ser implementado tendo em vista a melhorar a robustez dum sistema deste tipo.
Neste artigo iremos descrever uma solução para a concepção de um sistema de eventos, tolerante a faltas seguindo o paradigma "publish/subscribe". Para conseguirmos implementar tal sistema, usaremos a plataforma APPIA [5] como ferramenta de composição de protocolos.

2. Descrição do Problema

O projecto que nos propomos a desenvolver consiste em concretizar um sistema de edição-subscrição simplificado. Este sistema vai comportar dois tipos de clientes, Editores e Subscritores. Aos editores deve ser permitido que publiquem informação na forma de eventos para que sejam recebidos em tempo real por todos os subscritores interessados. Nesta fase o sistema a desenvolver é baseado em assuntos, que não são mais do que eventos publicados e marcados com uma string que os indentifica. Quando um subscritor pretende receber eventos de um determinado assunto, tem de indicar a sua intenção (`subscribe`) para que o sistema lhe envie as mensagens publicadas com esse assunto. No entanto, sempre que um editor pretender publicar eventos com um determinado assunto deve-o anunciar (`anunciar`) especificando qual a qualidade de serviço (QoS) pretendida e caso o assunto já exista, essa QoS tem de coincidir com a do assunto já existente. No caso do subscritor, dá-se algo semelhante, pois apenas pode subcrever um assunto se escolher a qualidade de serviço associada. No nosso sistema, as qualidades de serviço disponíveis são *sincronia na vista* e *ordem total* [1].

2.1. Características do Problema

No contexto do problema descrito anteriormente supomos que:

- Um subscritor só subscreve um assunto.

- Um editor não pode ser ao mesmo tempo subscritor de um outro assunto.
- Um editor só pode publicar para um assunto.
- Podem existir vários editores para o mesmo assunto.
- Um assunto, uma vez criado, não deixa de existir.

3. Arquitectura

3.1. Descrição

Nesta fase do artigo vamos examinar em detalhe os vários componentes do sistema. Estes são servidores, editores e subscritores. Aos servidores caberá a função de gerir de forma robusta uma estrutura de assuntos e atender os pedidos dos editores / subscritores e até de outros servidores. Entre si serão capazes de replicar o estado do sistema de forma a que este não seja perdido na eventualidade de um deles falhar. Os editores, por sua vez, devem anunciar aos servidores, a intenção de publicar eventos sobre um dado assunto e posteriormente, enviar os respectivos eventos directamente para os seus subscritores. Subscritores, esses, que apenas serão capazes de subscrever assuntos e de receber eventos sobre esse assunto.
Appia é uma plataforma middleware, que possui um conjunto de protocolos que oferecem uma série de propriedades [5]. Nesta arquictura usaremos esta plataforma, como forma de garantir que o nosso sistema respeita determinadas propriedades nomeadamente comunicação em grupo, garantias de ordem, difusão fiável e sincronia na vista.

3.2. Opções Tomadas

No início tinhamos presente duas versões possíveis para a aquitectura dos servidores: a centralizada e a descentralizada.
Tanto na versão centralizada como na descentralizada, todas as mensagens/pedidos enviadas para os servidores são disseminados entre eles. Desta forma todos mantêm um estado igual ao processarem as mesmas mensagens (pressupondo ordem total e difusão fiável). A ideia da versão centralizada consistia na eleição de um servidor coordenador, responsável por indicar quem responde ao pedido recebido por todos e transmitir o estado actual do sistema para um novo servidor.
As divergências deixaram de existir após detectarmos as desvantagens que podiam estar associadas a esta arquitectura, nomeadamente: O coordenador é um único ponto de falha (*single-point of failure*[1]), o que implica que caso falhe todo o sistema pára até eleição de um novo coordenador; Em situações de carga, o congestionamento é mais provável de acontecer, sendo que todos têm de esperar pela decisão do coordenador;
Assim, surge a ideia de voltar a desenvolver a versão descentralizada, por forma a resolver os problemas encontrados. Nesta versão ao contrário da anterior, não existe o conceito de coordenador, mas sim um "*leader*" responsável por responder a um pedido. Este é eleito por concenso através de um algoritmo determinista (sem usar comunicação entre servidores). Detalhes sobre a concretização deste algoritmo serão descritos mais tarde neste artigo (secção "4.3 Servidores: Replicação").

4. Concretização

4.1. Relação Entre Componentes

4.1.1 Relação Editor/Subcritor com Servidor

Os clientes (Editores/Subscritores) têm, como já foi referido anteriormente, função de publicar/subscrever um determinado assunto. Para tal, estes têm de previamente contactar os servidores para que lhes indiquem o nome de um grupo onde difundir/receber mensagens. O contacto com o servidor apenas existe quando um cliente pretende publicar/subscrever um assunto, indicando aos servidores qual o assunto e a qualidade de serviço (QoS) desejada. O grupo de servidores, que gere todos os assuntos, verifica se esse assunto já existe e se a qualidade de serviço corresponde. No caso de sucesso, um dos servidores responde com o nome de grupo para esse assunto. No entanto, o servidor pode ainda responder que a qualidade de serviço requerida pelo cliente não corresponde à existente. Nesta situação, apenas poderá publicar/subscrever nesse assunto se escolher a QoS indicada pelo servidor. As figuras que se seguem mostram-nos o modo de interacção entre clientes e grupo de servidores.

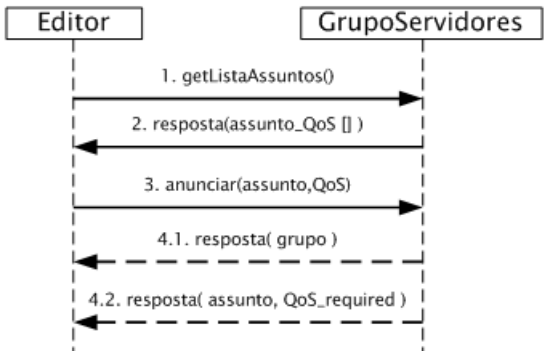

Figure 1: Modo de interacção entre Editor e o Grupo de Servidores

Como podemos verificar no diagrama da fig.1, o Editor vai implementar a função `anunciar`, onde passa como

parâmetros o assunto que quer publicar e a QoS. Se o assunto não existir, o servidor cria-o. Nesse caso este responde com o nome do grupo onde o assunto foi criado (passo 4.1). No caso em que o assunto já existe temos de verificar se a QoS requerida pelo Editor corresponde à QoS do assunto já existente. Essa indicação será fornecida na resposta 4.2. No caso favorável o editor junta-se ao grupo indicado pelos servidores para assim poder publicar mensagens desse assunto.

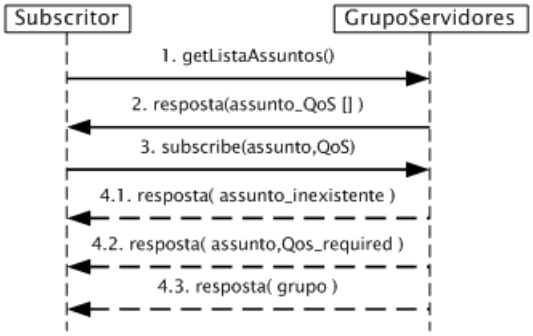

Figure 2: Modo de interacção entre Subscritor - Grupo Servidores

O funcionamento do Subscritor é análogo ao do Editor, como podemos ver na fig.2, no entanto existem diferenças relativamente aos passos 3 e 4.2. No `subscribe` (passo 3), a única diferença é relativa ao facto de o servidor apenas necessitar de verificar se o assunto já existe ou não. Caso não exista apenas informa o subscritor desse facto. O resto do funcionamento é análogo, como já foi indicado.

O primeiro passo (`getListaAssuntos()`) dos diagramas de sequência das fig.1 e 2, representa o pedido ao grupo de servidores para devolver a lista de assuntos existentes, que este gere. Esta funcionalidade apenas serve para que o cliente (Editor/Subscritor), não tenha de adivinhar os assuntos existentes e a respectiva QoS. De qualquer modo, um editor poderá inserir um assunto (com respectiva QoS) sem que este venha na lista.

4.1.2 Pilha Protocolar na relação entre o Editor/Subcritor com Servidor

As interacções existentes nesta aplicação entre os componentes, utilizaro as pilhas protocolares apresentadas na fig.3. As pilhas 1, 2 e 3 serão usadas pelos clientes. A pilha 4 será a usada pelos servidores.

A primeira e a segunda pilha representadas pelos números '1' e '2', são as pilhas usadas para a comunicação entre Editores e Subscritores de um determinado assunto com uma deteminada qualidade de serviço. Respectivamente a primeira pilha será usada para grupos com qualidade de serviço *ordem total* e a segunda pilha *sem ordem total*. As principais camadas desta primeira pilha a serem usadas estão apresentadas nesta figura, no entanto apenas as camadas 'Gestão de Assuntos' e 'Aplicação' serão implementadas por nós. Na 'Gestão de Assuntos' apenas existirá uma lista de assuntos, onde estarão registados aqueles sobre os quais os editores/subscritores pretendem publicar/receber as mensagens. Assim garantimos que, mesmo atinjindo o limite de grupos no sistema, este suportará um número arbitrário de assuntos. Esta pilha será usada no caso em que o cliente especifica como qualidade de serviço *ordem total*. A segunda pilha representada com o número '2', é em todo idêntica à anterior. A única diferença centra-se no facto de não existir ordem total. Esta será usada para a comunicação onde apenas existirá como QoS sincronia virtual. A terceira pilha representada pelo número '3', é usada para o anúncio de um assunto por parte do Editor ou para a subscrição por parte do subscritor. Essa comunicação vai ser efectuada com o grupo de servidores, que terá uma pilha igual representada pelo número '4'. A quarta pilha é usada pelo grupo de servidores para a sua comunicação. Para essa comunicação irá ser usada *ordem total*, como é explicado na [4.6 Servidores: Ordem Total]. Existirá ainda uma camada de Gestão de Mensagens que também se encontra explicada na secção [4.7. Servidores: Gestão de Mensagens].

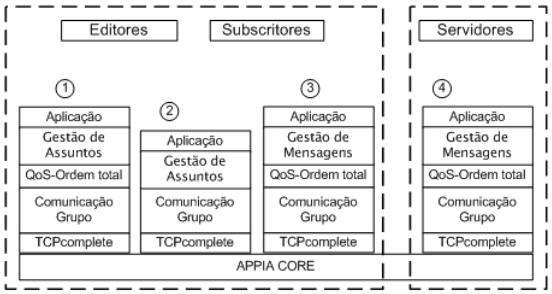

Figure 3: Conjunto de pilhas protoculares do Sistema.

4.2. Servidores: Estado do Sistema

Para representar o estado do sistema vamos utilizar duas tabelas, cada uma delas em contextos diferentes. Iremos precisar de representar todos os pedidos que já chegaram ao grupo mas que ainda não foram respondidos para a eventualidade duma falha do servidor que ficou encarregue de enviar resposta a esse pedido. Para isto basta manter uma tabela como a representada na fig.4.

Neste exemplo podemos ver que existem 4 servidores,

Servidores	Fila de Pedidos	Estado
A	{ anúncio1 }	activo
B	{ newbie1 }	activo
C	{ }	activo
D	{ }	inactivo

Figure 4: Tabela de Pedidos não respondidos (nos servidores)

3 activos e um inactivo. Pode-se concluir que o servidor D acabou de entrar para o grupo e ainda está a recolher informações para poder iniciar a actividade. Podemos também perceber que o servidor A está a processar o pedido `anúncio1` e o servidor B, o pedido `newbie1`. Caso algum destes falhe será possível recuperar o pedido e atribuí-lo a outro servidor para que o processe. Esta situação será descrita posteriormente em maior detalhe.

Assunto	Nome do Grupo	QoS
appia	1	Sinc. Vista
opengl	2	Ordem Total
webapps	3	Ordem Total
xml	1	Sinc. Vista

Figure 5: Tabela de Assuntos (nos servidores)

Para que cada servidor/réplica consiga responder a todos os pedidos é importante que tenha acesso descentralizado à tabela dos assuntos existentes (fig.5. Assim, se algum servidor falhar não haverá custo nenhum para o sistema visto todos os servidores terem conhecimento dos assuntos actuais. Esta tabela é construída a partir das mensagens recebidas no grupo de servidores e/ou na integração dum servidor no grupo.

4.3. Servidores: Replicação

Quando ocorrem pedidos de, ou para servidores, estes são disseminados para todo o grupo através das camadas de comunicação em grupo do APPIA [5] e cada membro mantém um registo na forma das tabelas descritas anteriormente. O modelo usado é semelhante à replicação semi-activa, também conhecida por *leader-follower* [1], mas ao contrário do original não existe um líder permanente, este é periodicamente eleito para responder a um pedido. Essa eleição é feita através de um algoritmo determinista baseado no consenso (entre réplicas) que consiste em percorrer a tabela de pedidos (fig.4) de cima para baixo onde somente contam os activos, escolhendo o servidor com a menor ”fila de pedidos”, ou seja, o menos ocupado (balanceamento de carga).
Só é possivel garantir que todas as réplicas se mantêm coerentes devido a uma camada responsável por gerir a ”fila de pedidos” da fig.4 e porque existe a garantia de que todas as mensagens são entregues a todos os elementos do grupo de servidores (difusão fiável) e pela mesma ordem (a importância da ordem total será descrita na secção ”Servidor: Ordem Total”).

4.3.1 Replicação: Nova Réplica

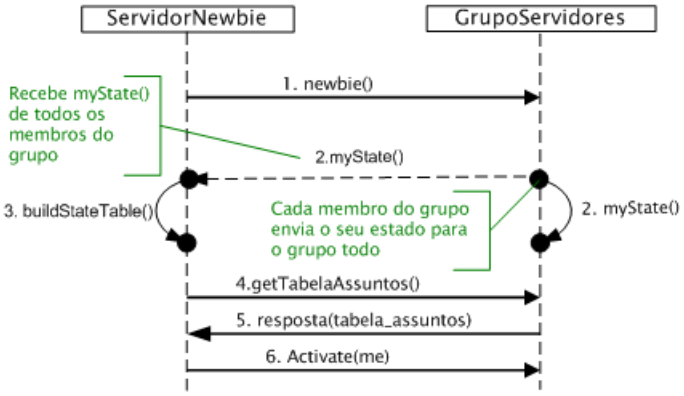

Figure 6: Surgimento dum novo servidor/réplica

Acrescentar uma nova réplica sem que o funcionamento do sistema seja interrompido é possível devido ao mecanismo de replicação a seguir descrito.
1. O newbie (novo servidor), envia uma mensagem para o grupo de servidores. Todos registam a sua entrada na sua *Tabela de Pedidos* (fig.4), mas colocam-no como inactivo (não conta para o algoritmo de eleição).
2. Todos os outros servidores enviam somente a sua ”fila de pedidos” da *Tabela* (fig.4) para o grupo.
3. O newbie constrói a sua tabela com as ”filas de pedidos” que recebe.
4. Quando verificar que já recebeu as ”filas de pedidos” de todos os elementos que pertencem ao seu grupo, efectua o pedido da *Tabela de Assuntos* (fig.5) que será atendido como se de outro pedido qualquer se tratasse.
5. O servidor eleito, responde com a *Tabela de Assuntos* (fig.5). A tabela manterá a coerência devido à ordem total, descrito na secção ”Servidor: Ordem Total”.
6. No fim, o newbie ficará com a réplica igual ao de todos os outros (porque ao longo do de todo o mecanismo, vai processando as mensagens recebidas pelo grupo e actualizando a *Tabela de Pedidos* (fig.4) e envia um pedido (`activateme`) para que possam contar com ele para

responder aos pedidos.

4.4. Servidores: Tolerância a faltas

Uma vez que todos os pedidos não atendidos estão replicados pelos membros do grupo de servidores, sempre que um servidor falhar a meio do processo dum pedido e não envie resposta para o grupo, será possível passar o(s) pedido(s) orfão(s) a um servidor livre.
Esta passagem deve ser feita com alguma precaução para não violar a ordem total. Para simplificar o processo, cada um selecciona o servidor com a fila mais pequena e une as filas desse servidor com a do servidor-defunto, não sem reordernar a fila resultante.

4.5. Servidores: Gestão de Grupos

Existem três grupos principais no Tulkas; o grupo de servidores, o grupo da QoS de Ordem Total e o grupo da QoS de Sincronia na Vista. Dentro destes últimos dois será criado o maior número de grupos possíveis, um por cada assunto. Quando se atingir o número máximo de grupos deverá começar-se a utilizar um grupo para cada dois assuntos e uma vez que todos os grupos tenham dois assuntos, passa-se a três por grupo e assim sucessivamente.

4.6. Servidores: Ordem Total

A utilização de uma camada que oferece ordem total nos servidores é necessária por várias razões:

- Como cada servidor mantém a sua ”tabela pedidos” (fig.4) coerente através das mensagens que recebe (tanto dos próprios servidores, como dos anunciantes subscritores), é importante que todas essas mensagens sejam recebidas por todos na mesma ordem.
- Quando um servidor se junta ao grupo existente, para criar mais uma réplica de dados, todos os servidores enviam a ”fila de pedidos” (fig.4) que os próprios têm para responder. Para que o novo servidor possa acompanhar o estado do sistema, a mensagem recebida com a ”fila de pedidos” (fig.4) terá que ser entregue antes de qualquer nova mensagem.
- O editor ao anunciar um assunto (cria assunto) a sua mensagem terá que ser atendida antes dos pedidos dos subscritores, caso contrário um subscritor não consegue subscrever um assunto já criado.

4.7. Servidores: Gestão de Mensagens

Esta camada é responsável pela criação da fila de pedidos por processar em cada servidor. Possibilita que um servidor possa receber pedidos sem os tratar imediatamente ficando ao critério da camada acima controlar quando é que pretende resolver. Esta camada torna-se particularmente importante no caso em que uma nova réplica é criada. Porque quando um novo servidor se junta ao grupo de servidores ainda inactivo é importante que vá guardando as mensagens enviadas ao grupo, de forma a quando estiver activo, poder começar a processar essas mesmas mensagens. [4.3.1 Servidores: Nova réplica] Considerando as capacidades de processamento dos sistemas de hoje em dia, é justo dizer que estas chamadas filas servem como protecção contra picos de utilização ou rajadas de pedidos mal intencionadas, mantendo sempre o serviço disponível, ainda a sua eficiência seja afectada.

5. Conclusão

Ao longo desto artigo, esperamos ter alcançado os objectivos a que nos proposemos, nomeadamente o de documentar uma forma de implementar um sistema baseado no paradigma Publish/Subscriber com capacidades de tolerância a faltas distribuídas. Concluímos que as ferramentas fornecidas pela plataforma APPIA facilita bastante a implementação deste sistema de forma fiável e robusta.

6. Trabalho Futuro

Ao longo do trabalho em torno deste artigo, identificámos algumas funcionalidades que poderiam ser adicionadas ao sistema de forma a proporcionar um melhor desempenho e uma maior liberdade de acção. Uma das funcionalidades a adicionar seria definir diferentes níveis de urgência para as mensagens trocadas dentro do grupo de servidores. Por exemplo, no caso em que um servidor quer juntar-se ao grupo, este tem de lhes enviar uma mensagem. Esta mensagem poderia ter uma maior prioridade sobre as restantes, para que esse elemento fosse tratado com a maior brevidade para mais rapidamente agir como um servidor replicado e tratar os pedidos. Outra funcionalidade seria detectar quando num assunto deixa de haver editores, se tal acontecer, poderíamos considerá-lo ’morto’ e desta forma eliminar o assunto. Implementar um ”editor/subscritor” em simultâneo e optimizar a forma de associar os assuntos aos grupos seria um dos próximos passos.

7. Referências

[1] Paulo Veríssimo, Luis Rodrigues. ”Distributed Systems for System Architects”, Kluwer Academic Publishers

[2] International Workshop on Distributed Event-Based Systems, http://www.cs.queensu.ca/ dingel/debs05/

[3] Jianjun Chen, David J. DeWitt, Feng Tian, Yuan Wang. NiagaraCQ: a scalable continuous query system for Internet databases

[4] Nuno Carvalho, Luís Rodrigues. Implementing Reliable Broadcast Protocols in Appia, http://www.di.fc.ul.pt/ ler/docencia/tfd0405/bib/RBTutorial.pdf

[5] Homepage do APPIA. URL: http://appia.di.fc.ul.pt