Avaliação Prática de um Detector de Defeitos: teoria *versus* implementação

Luiz Angelo Barchet Estefanel[1] Ingrid Jansch-Pôrto
{angelo,ingrid}@inf.ufrgs.br
Programa de Pós-Graduação em Computação
Curso de Mestrado em Ciência da Computação
Instituto de Informática - UFRGS
Porto Alegre - RS - Brasil

Resumo

A transposição de um modelo teórico para uma implementação costuma apresentar diversos problemas inesperados. No caso dos detectores de defeitos, a carência de publicações que demonstrem tais experiências muitas vezes leva a esforços duplicados e atrasos na solução dos problemas. Este artigo apresenta a experiência e as observações realizadas durante o desenvolvimento de um detector de defeitos *Heartbeat*. ilustrando bem os problemas desta transposição e quais soluções práticas foram tomadas neste experimento.

Abstract

The implementation of a theoretical model usually brings about many unexpected problems. Specially in the case of the failure detectors, the lack of publications on these problems drives to duplicated efforts and bigger delays to solve the problems. This work presents the experience and observations collected during the development of a Heartbeat *failure detector, showing the problems involved to transpose the theoretical model and the practical solutions.*

1 Introdução

A grande maioria dos trabalhos que tratam sobre detectores de defeitos enfocam os aspectos teóricos de suas funcionalidades, e de fato quase não há avaliações sobre o funcionamento dos detectores. Sergent *et al.* [SER 99], por exemplo, realizam comparações entre alguns detectores, mas não entram em detalhes sobre a construção destes por terem utilizado simulações ao invés de implementações.

Este artigo apresenta a experiência e as observações realizadas, ao estudar e implementar um protótipo do detector de defeitos *Heartbeat* [AGU 97a]. Aqui procura-se transmitir tanto os desafios quanto as soluções encontradas durante a construção e a operação dos detectores, estabelecendo um paralelo entre o que o que está formalmente expresso através das definições dos detectores e o que se verifica na prática. Os diversos passos que embasam os resultados aqui relatados estão detalhados em [EST 00a].

2 Tipos de Detectores

Apesar de Chandra e Toueg terem afirmado que o uso de *timeouts* não se aplica bem ao contexto dos detectores (devido às próprias características dos sistemas assíncronos), na prática eles são utilizados para a implementação. baseando-se no argumento de que, em sistemas práticos. não é necessário esperar indefinidamente a resposta de um processo para ter certeza de sua falha, bastando considerar um tempo suficientemente longo.

Assim, a construção dos detectores de defeitos normalmente utiliza limites de tempo a fim de julgar se os demais processos são suspeitos de falha. Entre as possibilidades de detectores para

1 Bolsista CNPq.

ambientes de falhas por *crash* destacam-se os modelos simples mas eficientes *Pull* e *Push* (nomenclatura utilizada por Felber [FEL 98]). Ambos são utilizados em diversos tipos de aplicações, não somente nos detectores de defeitos (por exemplo, a área de gerência de redes utiliza-os sob os nomes de *polling* e *heartbeat*, respectivamente, para monitorar o estado das máquinas). O princípio básico de um detector *Pull* é o envio de uma mensagem do tipo "*Are you alive?*". A partir do momento deste envio, é acionado um temporizador, e se até o fim do tempo determinado não houver resposta dos processos (na forma de uma mensagem do tipo "*Yes, I am!*"), estes processos são adicionados à lista local de suspeitos do detector de defeitos.

Já o método *Push* prefere, ao invés deste modelo pergunta-resposta, a utilização de envios periódicos de mensagens do tipo "*I am alive*", como forma de indicar seu estado (devido a isso também é freqüentemente chamado *heartbeat*). Se, após o recebimento de uma mensagem, não há um novo recebimento dentro de um prazo estabelecido, tal processo é considerado suspeito. Nota-se, entretanto, que devido às próprias características dos detectores de defeitos, se uma mensagem nova chegar em qualquer um destes detectores, o processo será retirado da lista de suspeitos.

Estes dois métodos, representam quase uma unanimidade entre as opções disponíveis para detectores dedicados (existem detectores que em vez de mensagens próprias controlam as mensagens trocadas pela aplicação, mas seu uso depende muito das características desta aplicação). Uma alternativa a esses métodos foi apresentada por Aguilera [AGU 97a], e batizado de *Heartbeat* (apesar do método *Push* também ser chamado *heartbeat*, o detector de Aguilera apresenta características bem diferentes).

3 O Detector *Heartbeat*

O *Heartbeat* proposto por Aguilera [AGU 97a] apresenta algumas soluções interessantes para a construção de um detector. A primeira refere-se à ausência de controles do tipo *timeout* no recebimento das mensagens (embora ainda sejam utilizados limites de tempo para a realização do julgamento do estado dos processos). Quando uma mensagem chega, um contador referente ao processo que a enviou é incrementado. Se no momento da amostragem dos estados um contador foi incrementado (em relação ao valor da amostragem anterior), não importa em quantas unidades, isso indicará que o processo esteve ativo naquele período. Além disso, essa característica mascara o atraso de algumas mensagens, permitindo que processos mais lentos não sejam considerados suspeitos erroneamente.

Outra solução apresentada é a que tenta reduzir o número de mensagens transmitidas entre os processos (em situações normais, a comunicação de todos para todos tende a elevar exponencialmente o número de mensagens). Esta solução envolve o conceito de vizinhos, de forma a restringir a abrangência da comunicação. Para evitar que esta restrição atrapalhe o conhecimento sobre os demais processos do sistema, também foram realizadas alterações no conteúdo das mensagens. Ao invés de um simples "*I am alive*" do detector *Push*, a uma mensagem do *Heartbeat* é anexado o identificador do processo que a recebeu, e esta mensagem é repassada a todos os vizinhos que ainda não constam da rota (*path*) por onde esta mensagem passou. Isso faz com que a própria mensagem, ao conter informações sobre quais os processos ativos onde ela passou, permita deduzir o estado destes processos sem ter que se comunicar diretamente com eles. Além disso, esse formato de mensagens facilita a expansão do detector para sistemas com possibilidade de partição na rede [AGU 97b], pois as mensagens contêm informações sobre as possíveis rotas de intercomunicação dentro do sistema.

Foram estes aspectos diferentes e curiosos que levaram à adoção do modelo *Heartbeat* para a implementação de um protótipo de detector. A seguir, estão as observações resultantes da implementação e dos testes de funcionamento, que constituem uma comparação entre os problemas encontrados no momento de transportar o modelo teórico para uma implementação.

3.1 Vizinhos

O detector *Heartbeat* proposto por Aguilera considera como vizinhos apenas as máquinas diretamente conectadas, comunicando-se apenas com estas. As demais máquinas da rede poderiam ser

monitoradas através do *path* associado às mensagens trocadas entre os detectores. No entanto, o *Heartbeat* não faz isso e apenas os vizinhos são observados, porque a descrição dos detectores *Heartbeat* levou em consideração algoritmos de consenso e difusão confiável construídos especificamente para as facilidades de difusão desse ambiente. A não ser que sejam utilizados esses algoritmos especialmente desenvolvidos, não é possível operar um consenso "tradicional" com tal detector, originando a necessidade de modificar tal comportamento do detector para poder operar com protocolos "genéricos".

As modificações introduzidas na implementação mantêm a proposta do conjunto de vizinhos, realizando a verificação de estado não somente nestes, mas em todos os processos do sistema, através das informações transmitidas na variável *path* do próprio *Heartbeat*. Essa alteração não modifica o modo de funcionamento do detector, mas como apresenta dados sobre vários processos do sistema, facilita a operação do detector junto aos demais módulos de um sistema.

Quanto à granulosidade do monitoramento, embora o ambiente de objetos distribuídos possibilite um monitoramento individual sobre cada objeto, optou-se por implementar um sistema de monitoramento menos abrangente que o apresentado por Felber para o OGS [FEL 98]. No OGS, o monitoramento é feito através da comunicação entre objetos e detectores de todo sistema, gerando perdas na transparência; como o modelo de falhas adotado restringe-se ao *crash* de toda a máquina, ficaria redundante o controle individual por objeto. Já a notificação da lista de suspeitos foi feita de forma semelhante ao OGS, uma vez que não há perda da transparência (somente o detector local sabe quais são os objetos a notificar), e um único detector por máquina pode fornecer suas suspeitas a diversos receptores. Assim, o modelo de monitoramento-notificação assemelha-se ao exposto na figura 1, onde os detectores de defeitos trocam mensagens entre si, notificando os objetos interessados nas suas listas de suspeitos.

Figura 1 - Modelo de detecção-notificação

3.2 Amostragem dos suspeitos

O detector *Heartbeat* caracteriza-se por não empregar limites temporais para avaliar o recebimento de mensagens dos demais nós da rede. Ao invés de acusar suas suspeitas quando uma mensagem não chega em um tempo limite, o detector *Heartbeat* fornece apenas o número de mensagens que recebeu de cada nó, cabendo à camada de *software* imediatamente superior comparar os contadores obtidos em diferentes instantes, à procura de nós que não tiveram seus contadores incrementados. Isso torna a detecção pouco transparente, pois a aplicação também fica encarregada de realizar o processo de suspeita das máquinas.

O algoritmo modificado inclui uma camada adicional específica para esse fim, ou seja, um complemento ao detector *Heartbeat* que realiza periodicamente a verificação de quais nós podem ser considerados suspeitos. Dessa forma, a aplicação recebe uma lista de suspeitos, no mesmo formato empregado pelos demais detectores de defeitos, facilitando a troca de módulos contendo implementações diferentes de detectores.

3.3 Notificação da suspeita

Como o detector *Heartbeat* repassa a responsabilidade da detecção de suspeitos para a aplicação, não há nenhuma especificação quanto à forma com que a aplicação é notificada sobre os processos suspeitos. Nos modelos tradicionais, a notificação comumente é executada sem uma

requisição proveniente da aplicação, exigindo então a utilização de um mecanismo de *callback*.

Não obstante, para evitar restringir as formas de interação com o detector de defeitos, foram incluídos outros mecanismos para a obtenção da lista de suspeitos, além do *callback*. Um destes métodos corresponde à requisição ao detector da lista de suspeitos, por parte de um objeto específico. Somente o objeto que realizou esta chamada irá receber a lista de suspeitos atualizada, enquanto os demais objetos deverão aguardar até a nova amostragem, realizada periodicamente. Esse método pode ser utilizado também por elementos que não façam parte do conjunto de "notificáveis", e que ocasionalmente necessitam obter a lista de suspeitos.

Outro método disponível permite que seja disparada uma nova amostragem, fora do momento pré-definido, sob demanda. Este mecanismo pode ser utilizado por elementos que desconfiam de algum processo externo (o módulo de comunicação, por exemplo, pode detectar problemas na entrega de mensagens), e que desejam acelerar a troca da visão do sistema.

3.4 Comunicação não confiável

A utilização de um protocolo de transporte que não garante a entrega, como no caso do UDP, não teve nenhum efeito negativo sobre a precisão do detector implementado. De fato, o monitoramento das mensagens enviadas entre máquinas de uma rede local revelou que não ocorreram perdas de mensagens em número suficiente para comprometer a detecção. Ainda assim, espera-se que mesmo no caso de perdas de mensagens (devido ao tráfego da rede, por exemplo), estas não influenciarão tão dramaticamente a detecção [EST 00b]. O modelo do detector *Heartbeat* ainda contribui para minimizar a influência dessas perdas, pois cada mensagem que consegue chegar ao seu destino carrega informações sobre diversas máquinas por onde passou antes. O número de mensagens trocadas varia de acordo com o número de máquinas adicionadas conjunto de vizinhos, mas quanto mais interligados forem os processos, tende a ser mais alto o número de mensagens. Assim, é pouco provável que não ocorra a coleta de informações sobre todas as máquinas monitoradas, mesmo com poucas mensagens recebidas.

Como o número de mensagens que trafegam tende a ser grande, a utilização de um protocolo como o UDP reduz o custo do estabelecimento de cada transmissão, e as possíveis perdas de mensagens são suficientemente compensadas pelo grande número de informações enviadas.

3.5 Quantidade de mensagens

Um detector de defeitos, apesar de ser um módulo dedicado, não deve representar um grande peso no desempenho do sistema em que será integrado, nem é desejável que a precisão de suas suspeitas seja condicionada à super-utilização da rede de comunicação.

Uma das possibilidades do modelo *Heartbeat* é a minimização do número de mensagens, pois todas as transmissões são baseadas no conjunto de vizinhos. Esse modelo, entretanto, não permite a integração com outros protocolos, pois considera que somente os contadores dos vizinhos serão incrementados. A implementação do detector estendeu o modelo do *Heartbeat* ao utilizar as informações contidas na própria estrutura *path* das mensagens, para monitorar todos os processos da rede.

A determinação do número ótimo de processos a ser considerado no conjunto de vizinhos é essencial na operação dos detectores, e para auxiliar nesse julgamento foram realizados alguns testes com o detector. Os testes realizados utilizaram cinco detectores comunicando-se em computadores diferentes, de forma a representar o monitoramento de cinco máquinas. Além disso, os mesmos testes foram executados entre cinco detectores rodando em uma única máquina, a fim de verificar a carga imposta ao sistema. A simulação das falhas dos processos foi criada através da finalização manual das aplicações que rodavam os detectores.

Em um primeiro momento, foi analisada a comunicação entre os detectores quando se utiliza todos os processos do sistema dentro do conjunto de vizinhos. Embora a detecção de defeitos seja amplamente favorecida pela quantidade de informações, um número excessivo de mensagens trocadas representa um possível problema quando forem utilizadas redes com muitos processos. A razão desse aumento excessivo é que, quando todos os objetos monitorados pertencem ao conjunto de vizinhos,

cada mensagem recebida será retransmitida para todos os outros detectores, até que não existam mais objetos vizinhos ausentes do *path*. O número de mensagens, embora não tenha causado nenhum problema nos testes, apresentou uma curva de crescimento muito acentuada (figura 2), o que pode exigir a definição de um limite prático ao número de objetos monitorados, a fim de manter os sistemas em boas condições de operação.

Figura 2 - Número de mensagens trocadas

No outro extremo, quando se considera apenas um dos outros processos como vizinho, a transmissão correta de informações somente ocorre quando todos são dispostos em um ciclo fechado. Qualquer falha em um dos processos do ciclo irá interromper a circulação das mensagens, causando suspeitas incorretas nos demais processos do sistema, como demonstra a figura 3.

Avaliando-se as situações médias, onde exista um número de processos vizinhos razoável, verifica-se que o número de mensagens é suficientemente controlado, não impondo sobrecarga ao sistema. Entretanto, mesmo nesses casos deve-se tomar extrema precaução quanto à topologia com que são interligados os processos. Deve-se evitar processos que somente enviem ou recebam mensagens, pois em tais casos estes não funcionarão ou serão considerados inadequadamente. Outro problema interessante refere-se à possibilidade de falhas em todos os vizinhos de um processo, como demonstra a figura 4. Nestes casos, um detector é isolado dos demais através de uma partição lógica incorreta, e a visão do sistema será afetada pelas diferentes suspeitas observadas entre os detectores, mesmo quando os protocolos superiores puderem se comunicar diretamente com os demais processos.

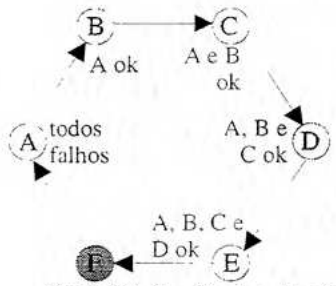

FIGURA 3 - Quebra do ciclo

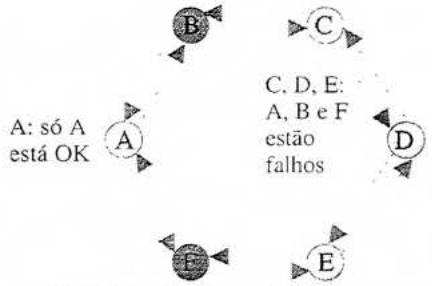

FIGURA 4 - Partição lógica incorreta

Com base nesses aspectos, recomenda-se que o número de processos diretamente conectados e a topologia dessas conexões sejam tomados de acordo com um objetivo mínimo de tolerância a falhas para o suporte ao sistema (por exemplo, número máximo de falhas suportadas).

4 Conclusões

A experiência no desenvolvimento de um detector de defeitos, apresentada neste artigo, possibilita uma avaliação sobre as dificuldades e escolhas encontradas na transposição de um modelo teórico para um sistema real. Diversos aspectos tiveram que ser repensados e até mesmo modificados, para que fosse garantido o funcionamento do protótipo junto ao restante do sistema. Essas modificações não tiveram por objetivo descaracterizar o modelo original, mas apenas adaptá-lo às situações comumente encontradas na prática. Embora tais modificações não tenham sido validadas matematicamente, a observação prática do protótipo indica que estas foram bem sucedidas.

Outro aspecto que deve ser ressaltado das observações realizadas refere-se aos parâmetros de inicialização dos detectores. Cada ambiente exige uma avaliação específica, de tal forma que sejam passados parâmetros condizentes com a sua realidade, de forma a expressar os melhores valores de *timeout*, a melhor topologia de comunicação, entre outros.

Não obstante, a descrição da implementação e das características de funcionamento foram importantes, pois são de grande auxílio tanto na implementação de modelos de detectores já existentes quanto na elaboração de novos modelos que venham a solucionar os problemas aqui destacados.

Referências

[AGU 97a] AGUILERA, Marcos Kawazoe; CHEN, Wei; TOUEG, Sam. Heartbeat: a timeout-free failure detector for quiescent reliable communication. In: 11TH International Workshop on Distributed Algorithms, **Proceedings...**, Setembro 1997. Também publicado como Technical Report, Cornell University, Maio 1997. Disponível por WWW em *http://cs-tr.cs.cornell.edu:80/Dienst/UI/1.0/ Display/ncstrl.cornell/TR97-1631* (11 Janeiro 2000)

[AGU 97b] AGUILERA, Marcos Kawazoe; CHEN, Wei; TOUEG, Sam. **Quiescent Reliable Communication and Quiescent Consensus in Partitionable Networks**, Technical Report, Cornell University, Julho 1997. Disponível por WWW em *http://cs-tr.cs.cornell.edu:80/Dienst/UI/1.0/Display/ncstrl.cornell/TR97-1632* (11 Janeiro 2000)

[EST 00a] ESTEFANEL, Luiz Angelo Barchet. **Detectores de Defeitos Não Confiáveis**, Trabalho Individual No. 880 (publicação interna), PPGC, UFRGS:Porto Alegre, janeiro 2000. Também disponível em *http://www.inf.ufrgs.br/~angelo/* (8 Maio 2000)

[EST 00b] ESTEFANEL, Luiz Angelo Barchet; JANSCH-PÔRTO, Ingrid. Comunicação Não Confiável em Detectores de Defeitos com Falhas por *Crash*. In: II Workshop de Testes e Tolerância a Falhas (WTF'2), **Anais...**, Curitiba, Julho 2000.

[FEL 98] FELBER, Pascal. **The CORBA Object Group Service**, Tese de Doutorado, *École Polytechnique Fédérale de Lausanne*, Suíça, 1998. Disponível por WWW em *http://lsewww.epfl.ch/OGS/thesis/* (12 Janeiro 2000)

[GUE 97] GUERRAOUI, Rachid; SCHIPER, André. Consensus: The Big Misunderstanding. In: Proceedings of the IEEE International Workshop on Future Trends in Distributed Computing Systems (FTDCS'97), Outubro 1997. Também disponível por WWW em *http://lsewww.epfl.ch/~rachid/papers/ftdcs2-97.ps* (12 Janeiro 2000)

[SER 99] SERGENT, Nicole; DÉFAGO, Xavier; SCHIPER, André. **Failure Detectors: implementation issues and impact on consensus performance**. Technical Report, EPFL, Suíça, Maio 1999. Disponível por WWW em *http://lsewww.epfl.ch/Publications/techreps.shtml* (8 Maio 2000)