Comunicação Não Confiável em Detectores de Defeitos com Falhas por *Crash*

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 definição dos detectores de defeitos para ambientes com falhas por *crash* apresenta restrições severas, mesmo nos mais fracos detectores. Ainda assim, a operação dos detectores em sistemas distribuídos assíncronos é possível, mediante certas adaptações como, por exemplo, a utilização de *timeouts*. Um fator importante para a construção dos detectores é a necessidade de comunicação confiável; mesmo que esta seja requerida pela definição dos detectores, experiências práticas demonstram que se pode utilizar comunicação não confiável para a sua implementação. Este artigo toma como base os protocolos TCP e UDP, respectivamente confiável e não confiável, que representam as opções nativas da maioria dos sistemas operacionais, para avaliar o impacto de seu uso no funcionamento dos detectores de defeitos.

Abstract

The definition of failure detectors to be used on environments with crash failure hypothesis has some severe restrictions even in its weaker implementations. Yet, failure detectors are possible in distributed asynchronous systems, if we use technics to circumvent these restrictions, like the timeouts. During the construction of failure detectors, the need of reliable communication is an important factor: even if the definition if the failure detectors requires reliable communication, practical experiences show that we can use unreliable communication in its construction. This paper focuses on the native choices for reliable and unreliable communication, TCP and UDP, examining their impact on failure detectors' behavior.

1 Introdução

Quanto mais genérico um sistema distribuído, mais ampla é a sua possibilidade de adaptação aos diversos ambientes de operação, e os sistemas distribuídos puramente assíncronos representam o modelo mais genérico utilizado. Há, no entanto, algumas restrições nesse modelo que aumentam a complexidade das operações nele realizadas, sobretudo na possibilidade de falhas. Por exemplo, é conhecida a incapacidade de garantir o fechamento de uma operação de consenso, uma das mais significativas operações de acordo entre processos, em ambientes assíncronos sujeitos a falhas. Isso se deve à impossibilidade de determinar se um processo encontra-se falho ou apenas mais lento que os demais. Essa restrição, conhecida como Impossibilidade FLP (em homenagem aos seus autores, Fischer, Lynch e Paterson), ocasiona que, devido à falta de conhecimento sobre o sistema, o algoritmo de consenso deve esperar a resposta dos demais processos, sob risco de tornar as decisões inconsistentes. No caso da falha de um integrante do consenso, o algoritmo irá esperar indefinidamente. Essa restrição não afeta somente o consenso; muitas outras operações freqüentemente utilizadas, como a votação e a difusão atômica, são tão ou mais complexas que o consenso, e ficam sujeitas à mesma restrição de operação.

[1] Bolsista CNPq.

Existem algumas técnicas que objetivam contornar tais restrições, como por exemplo o sincronismo parcial e o assincronismo temporizado [GUE 97]. Tais propostas, entretanto, servem apenas a situações específicas, não contemplando as diversas possibilidades do sistema. Chandra e Toueg [CHA 96] propuseram um modelo de assincronismo auxiliado por um detector de defeitos, que demonstrou ser mais abrangente e adaptável que os demais [GUE 97]. Tais detectores funcionam através da manutenção de uma lista local, contendo os processos suspeitos de falhas, baseada em mensagens trocadas entre os detectores. Esse controle dá-se de tal forma que, se um processo incorretamente considerado suspeito voltar a enviar mensagens, este será retirado da lista de suspeitos tantas vezes quanto necessário. Como pode-se perceber, os detectores não podem determinar com exatidão o estado dos processos do sistema, mas sua utilização aumenta o conhecimento sobre os demais elementos, auxiliando na finalização das operações de consenso em tempo hábil, sem que suposições incorretas sejam mantidas.

2 Detectores de Defeitos

Chandra e Toueg definiram os detectores de defeitos através de duas propriedades a serem respeitadas: *completeness* (abrangência, completeza) e *accuracy* (precisão). A propriedade *completeness* refere-se à capacidade do detector identificar todos os processos que estão falhos, enquanto *accuracy* determina a precisão desta suspeita, a fim de evitar a inclusão de processos corretos nas listas de suspeitos. Ao respeitar essas duas propriedades, um detector de defeitos garante que os algoritmos que o utilizem não perderão a consistência das decisões, nem ficarão indefinidamente bloqueados (preservando as propriedades *safety* e *liveness*, respectivamente). Segundo essas propriedades, um detector que continuamente inclua e remova os processos da sua lista de suspeitos não é capaz de garantir as condições mínimas ao consenso, pois tal comportamento constitui uma violação da propriedade de *accuracy* dos detectores (por exemplo, *accuracy* em um detector considerado mais "fraco", *Eventually Weak* - $\Diamond\mathcal{W}$, diz que "há um instante de tempo após o qual um processo correto não é considerado suspeito por nenhum processo correto"). A indeterminação que acompanha um ambiente assíncrono impediria a definição de um *timeout* adequado ao sistema, uma vez que sempre poderia haver um processo mais lento. Os autores afirmam, entretanto, que em situações práticas tais detectores poderão considerar *timeouts* "suficientemente longos", de modo que todos os processos corretos consigam responder a tempo. Outra opção, quando não é possível determinar o máximo *timeout* suportável, é o uso de *timeouts* adaptativos, onde o atraso de uma mensagem faz com que o tempo de espera pela próxima seja incrementado em algumas unidades. Apesar dessa solução, a observação dos modelos de detectores mais utilizados (*Push, Pull*) indica que comumente utiliza-se *timeouts* fixos, ao invés de adaptativos.

A opção majoritária por soluções que utilizam *timeout* fixo leva a crer que mesmo nesses casos as suspeitas incorretas são "aceitáveis" nos sistemas. De fato, tais situações devem-se ao próprio comportamento dos detectores e ao ambiente de utilização mais comum deles. Quando um detector comete um engano, suas características fazem com que um processo correto seja suspeito de falhas, e não o contrário, onde um processo falho é considerado correto. Como o exemplo mais usado do emprego dos detectores envolve os algoritmos de consenso, onde os detectores são usados para acusar quando o coordenador está falho, uma detecção incorreta leva ao cancelamento da operação, preservando as propriedades de *safety* e *liveness* da aplicação.

O trabalho que definiu os detectores de defeitos teve por objetivo demonstrar quais as situações onde o consenso é solúvel, e por conseguinte, determinou o ambiente de falhas (*crash*) e o modelo de comunicação (confiável) onde estas demonstrações eram possíveis. Este artigo propõe uma análise específica do funcionamento dos detectores, pois estes apresentam exigências diferentes dos demais componentes do sistema, de tal forma que algumas opções de implementação (no caso deste estudo, a comunicação) poderiam ser aplicadas nos detectores de modo diferente dos demais subsistemas.

3 Comunicação Confiável *versus* Comunicação Não Confiável

Antes de mais nada, é necessário definir comunicação confiável. Este trabalho considera a definição de Aguilera [AGU 97]. Ele afirma que, quando um processo s invoca uma chamada de envio *send* ,(m) e retorna à execução, passando a tarefa do envio aos níveis mais baixos do sistema operacional, costuma-se dizer que o processo s "completou o envio da mensagem m para r, sendo r o processo destino. Entretanto, o simples envio da mensagem não garante o recebimento, pois pode ocorrer que, logo após s retornar da invocação, este processo entre em colapso (*crash*), antes que a mensagem seja realmente enviada para r. Nesse instante, pode ocorrer uma inconsistência entre os dois processos. Para evitar isso, primitivas de comunicação confiável estabelecem que se s enviou a mensagem m, e r é um processo não falho (também chamado de correto), então r irá receber a mensagem, mesmo após o *crash* de s.

As propriedades que determinam essas características da comunicação confiável são as seguintes [AGU 97]:

- Integridade: para todo $k \geq 1$, se r recebe a mensagem m de s k vezes, então s enviou anteriormente a mensagem pelo menos k vezes;
- *No Loss* (Sem Perdas): para todo $k \geq 1$, se r é um processo correto e s completou o envio da mensagem m para r exatamente k vezes, então r receberá m de s ao menos k vezes.

A maneira mais comum de construir tais primitivas de envio confiável envolve a retransmissão sistemática da mensagem m até que o processo r devolva um ACK, confirmando o recebimento da mensagem. Só então, com a confirmação de r, é que a invocação de envio retorna o processamento para s. Sob tais condições, s somente considerará a mensagem entregue após a resposta de r. Caso s falhe após a resposta de r mas antes de retornar da invocação, quando s se recuperar, irá apenas enviar uma mensagem repetida, que é facilmente descartada por r.

Numa primeira análise, pode-se observar no funcionamento dos detectores de defeitos que eles não necessitam garantir a consistência com outros detectores durante sua operação. De fato, é explícito que os detectores trabalham apenas sobre suas percepções locais, podendo ter listas de suspeitos completamente diferentes em um mesmo instante de tempo [CHA 96]. A formação desta "visão" local de cada detector deve ocorrer de modo instantâneo, ou seja, a cada momento a lista de suspeitos pode mudar, expressando a percepção das novas mensagens recebidas. Essa possibilidade de corrigir os enganos é uma característica essencial para que os detectores sejam aptos ao auxílio das operações em sistemas distribuídos puramente assíncronos, e em sistemas práticos podem ser considerados detectores que mudem suas listas de suspeitos tantas vezes quantas forem necessárias.

Quando se utiliza *timeout* para detectar os defeitos, esse período de tempo deve ser estabelecido de forma que a detecção ocorra com o menor atraso possível, porém sem induzir um número excessivo de falsas suspeitas (como, por exemplo, na presença de processos mais lentos). De fato, utilizando-se apenas *timeouts* para as mensagens, é impossível distinguir os casos de processos lentos ou falhos dos casos de mensagens extraviadas.

Nos ambientes que suportam falhas por perdas de mensagens, os protocolos que utilizam os detectores (o consenso, especialmente) costumam agregar técnicas para mascarar essas perdas, tal como a retransmissão de mensagens perdidas. Chandra e Toueg, quando analisaram um detector para sistemas parcialmente síncronos com perda de mensagens, verificaram que, embora o detector de defeitos continuasse operacional, já não era possível implementar o algoritmo de consenso sem que houvesse esse mascaramento das mensagens perdidas. Assim, mesmo que o algoritmo de consenso necessite a retransmissão de mensagens, sobre os detectores o efeito da retransmissão de mensagens terá o mesmo impacto do recebimento de novas mensagens, de forma que o uso de retransmissão pode vir a sobrecarregar a rede sem que a qualidade das mensagens influencie na detecção. Além disso, do ponto de vista temporal, somente as mensagens novas poderão efetivamente auxiliar na precisão da detecção.

Se além do uso de retransmissão de mensagens for realizado um ordenamento, comum em transmissões confiáveis, pode-se tornar o detector inadequado à operação com processos de velocidades relativas muito diferentes. Um exemplo claro desta inadequação pode ser encontrado

numa das formas de implementar o detector *Pull*. Neste modelo, um processo p inicialmente questiona um processo q sobre seu estado, enviando mensagens do tipo "*Are you alive?*". Quando q recebe tais mensagens e encontra-se operacional, envia uma resposta contendo algo como "*Yes, I am!*". Imaginando um cenário onde um dos processos (no caso q) é muito lento, como mostra a figura 1, verifica-se que a lista de processos suspeitos é corrigida apenas quando chegam as primeiras respostas, no momento t_n. Considerando que o processo q, a partir deste momento, irá responder todas as mensagens em intervalos determinados por sua velocidade, este será retirado da lista de suspeitos, pois estará respondendo dentro de um intervalo de tempo esperado [SCH 00]. Se fosse utilizado um controle sobre o número de seqüência, à resposta da primeira mensagem seria atribuído um atraso muito superior ao limite de tempo considerado, o que levaria à suspeita do processo q mesmo que todas as suas mensagens chegassem em ordem a partir desse momento. Além disso, devido à menor velocidade de processamento de q, torna-se interessante prover o detector com *timeout* adaptativo, sob o risco de causar um *overhead* nas operações com a constante inclusão e remoção deste processo da lista de suspeitos.

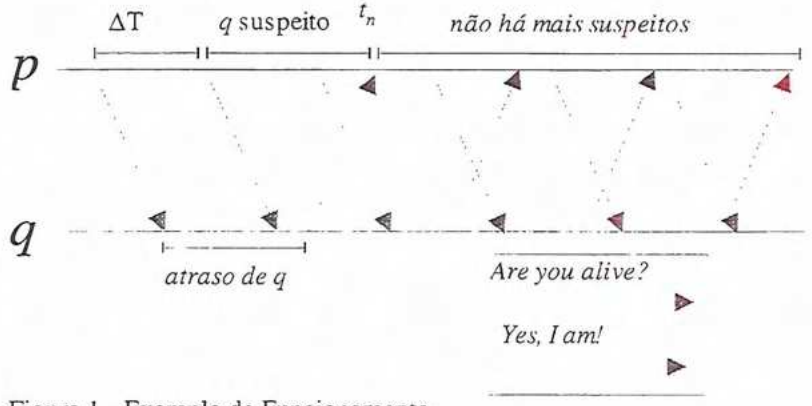

Figura 1 - Exemplo de Funcionamento

4 Alternativas Exploradas pelos Sistemas Práticos

Os sistemas práticos estudados costumam utilizar comunicação confiável, sobretudo para mascarar possíveis falhas no meio de comunicação. A não ser que o ambiente de falhas permita a perda de mensagens, subentende-se que a mensagem deve chegar ao seu destino. Alguns exemplos de sistemas que se enquadram neste ambiente (comunicação confiável e meio não confiável) são:

- detector *HeartBeat* [AGU 97] - subentende uma conexão "quase confiável", onde algumas mensagens podem ser perdidas, sem no entanto constituir uma partição na rede;
- o sistema de comunicação de grupo Bast [GAR 98] - considera o uso de *fair lossy links*, que também podem sofrer perdas eventuais de mensagens.
- detector para ambientes com falhas por *Muteness* [DOU 99] - requer comunicação confiável para distinguir o comportamento dos processos;
- sistema de comunicação de grupo OGS [FEL 98] - utiliza TCP devido ao ambiente CORBA para que foi projetado (embora possa escolher entre chamadas RPC ou *one-way*);
- sistema de comunicação de grupo TOTEM [MOS 96] - mesmo utilizando *multicast* UDP, obriga o envio confiável das mensagens, para controlar os *timeouts* e disparar as suspeitas sobre os processos.

De modo geral, verifica-se que a comunicação confiável é utilizada indistintamente para todos os protocolos do sistema, devido às restrições que o ambiente de falhas impõe. De fato, em certos protocolos há a necessidade do mascaramento das falhas do meio de comunicação, sobretudo porque a consistência das decisões está em jogo. Este artigo questiona somente a necessidade deste

mascaramento no nível da detecção de defeitos, já que não há contribuição para uma melhor detecção, nem a perda de mensagens constitui uma fonte de informações incorretas que levem à inconsistência dos protocolos superiores. Um detector cujas mensagens podem ser extraviadas ainda consegue ser suficientemente correto para que não induza a inconsistência do sistema, sem que no entanto sua implementação exija um grau de complexidade mais elevado.

5 Impacto da Comunicação Não Confiável no Desempenho do Sistema

Um aspecto adicional do uso de comunicação não confiável refere-se ao desempenho do detector (ou mais especificamente, o impacto do detector no desempenho global do sistema). Na prática, muitos sistemas distribuídos são construídos utilizando as primitivas de comunicação fornecidas pelos sistemas operacionais, sobretudo os protocolos TCP/IP e UDP/IP, confiável e não confiável, respectivamente. Embora não sejam ideais para a utilização junto aos detectores de defeitos (é possível, por exemplo, construir protocolos confiáveis tão eficientes ou mais que o UDP), tais protocolos ilustram bem as diferenças entre o uso de comunicação não confiável, com seu envio não bloqueante, mas sem garantia de entrega, e o uso de comunicação confiável, com garantia de entrega e retransmissão de mensagens perdidas, mas com características de envio bloqueante. Utilizando-se uma comparação entre os protocolos TCP e UDP pode-se extrair diversos pontos onde o uso do TCP torna o sistema menos eficiente, podendo até comprometer a precisão da detecção:

- estabelecimento da conexão - se cada transmissão entre os detectores exigir um novo estabelecimento de conexão, o protocolo TCP irá executar diversos *hand-shakings* antes de enviar as mensagens. Se for decidido criar uma conexão permanente, há a necessidade de uma estrutura específica para sua persistência. O UDP apenas envia o datagrama, e confia que os níveis inferiores da rede irão transmitir a mensagem até seu destino;
- envio bloqueante - somente quando a confirmação de recebimento das mensagens chegar, o protocolo TCP irá retornar o controle à aplicação. Isso pode acarretar um grande atraso no envio da mensagem para outros processos (especialmente no caso de falhas de transmissão), podendo causar suspeitas incorretas devido à falta de envio no prazo esperado;
- entrega ordenada - o TCP faz uso de números de seqüência para garantir a entrega ordenada das mensagens, e quando uma mensagem é perdida ou está atrasada, as outras somente serão entregues à aplicação quando esta mensagem for recebida. Apesar do protocolo TCP utilizar técnicas de *slide windows* para minimizar este problema, pode ocorrer um atraso na entrega das mensagens, atrapalhando a instantaneidade da formação da lista de suspeitos.

Em outros subsistemas que não os detectores de defeitos, o estabelecimento de conexões do TCP poderia compor uma grande vantagem, devido à quantidade de dados a enviar. Entretanto, o conteúdo das mensagens dos detectores costuma ser mínimo, não justificando os esforços para manter essas conexões abertas, ou para repará-las caso uma detecção se mostre incorreta.

A utilização de um protocolo de comunicação confiável específico para este problema é a solução ótima para os detectores, mas estes protocolos normalmente não são nativos nos sistemas operacionais e possivelmente teriam que ser implementados pelo próprio desenvolvedor do detector. Não obstante, nestes casos o protocolo especializado constitui uma solução bem mais adequada do que o uso dos protocolos TCP ou UDP. Quando o desenvolvedor for obrigado a escolher entre os protocolos nativos dos sistemas operacionais, o TCP apresenta características que praticamente inviabilizam uma boa operação de um detector.

6 Conclusões

A concepção de um sistema distribuído normalmente inicia pela definição do modelo de falhas a ser tratado. Enquanto a escolha de um modelo mais restritivo facilita a concepção, ao mesmo tempo aumenta a complexidade da implementação, pois as técnicas utilizadas para adequar um sistema real ao modelo de falhas escolhido exigem níveis de complexidade compatíveis com o que se deseja (um

exemplo é o uso de retransmissão para esconder perdas de mensagens). Essa diferença de complexidade usualmente faz com que cada modelo de falhas tenha implementações específicas, e se tornou comum rescrever todos os componentes do sistema, sem verificar o nível de complexidade exigido para cada componente.

Este artigo tentou demonstrar que, em sistemas práticos, os detectores de defeitos podem ser construídos utilizando comunicação não confiável para o suporte a sistemas com falhas por *crash*, uma vez que as perdas ocasionais de mensagens não levam a inconsistências do sistema, e a sua construção é bem menos complexa do que utilizando comunicação confiável. Os demais subsistemas continuariam com comunicação confiável segundo suas exigências, mas os detectores poderiam ser construídos com protocolos menos exigentes. O uso do protocolo de comunicação não confiável nativo da maioria dos sistemas operacionais (UDP), quando não existe nenhum protocolo especializado, pode tornar o impacto dos detectores de defeitos bem menor no desempenho final da aplicação do que o correspondente protocolo de comunicação confiável (TCP).

Referências

[AGU 97] 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)

[CHA 96] CHANDRA, Tushar Deepak.; TOUEG, SAM. Unreliable Failure Detectors for Reliable Distributed Systems. **Journal of the ACM**, v. 43, n. 2, pg 225-267, Março 1996. Também disponível por WWW em *http://cs-tr.cs.cornell.edu:80/Dienst/ UI/1.0/Display /ncstrl.cornell/TR95-1535* (12 Janeiro 2000)

[DOU 99] DOUDOU, A.; GARBINATO, Benoît; GUERRAOUI, Rachid; and SCHIPER, André. Muteness Failure Detectors: Specification & Implementation. In: Third European Dependable Computing Conference (EDCC-3), **Proceedings...**, Springer Verlag, Suíça, Setembro 1999. Também disponível por WWW em http://www.ubilab.com/ publications/print_versions/pdf/dou99-1.pdf (9 Maio 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)

[FIS 85] FISCHER, Michael J.; LYNCH, Nancy A.; PATERSON, Michael S. Impossibility of distributed consensus with one faulty process. **Journal of the ACM**, v. 32, n. 2, pg 374-382, 1985.

[GAR 98] GARBINATO, Benoît. **Protocol Objects and Patterns for Structuring Reliable Distributed Systems**, Tese de Doutorado, *École Polytechnique Fédérale de Lausanne*, Suíça, 1998. Disponível por WWW em *http://lsewww.epfl.ch/garbinato/PhD/* (11 Janeiro 2000).

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

[MOS 96] MOSER, L. E.; MELLIAR-SMITH, P. M.; AGARWAL, D. A.; BUDHIA, R. K., LINGLEY-PAPADOPOULUS, C. A. Totem: A Fault-Tolerant Multicast Group Communication System. **Communications of the ACM**, v. 39, n. 4, pg 54-63, Abril 1996.

[SCH 00] SCHIPER, André. **Implementação de um detector *Pull***, Março 2000. Comunicação pessoal por e-mail.