Avaliação Prática do Desempenho dos Detectores de Defeitos

Luiz Angelo Barchet-Estefanel

Avaliação Prática do Desempenho dos Detectores de Defeitos

Luiz Angelo Barchet Estefanel[1¤]

[1] Mestrado em Ciência da Computação, PPGC, Instituto de Informática, UFRGS

Porto Alegre − RS − Brazil Cx. Postal: 15064 CEP 91501−970

Phone: +55 (51) 316−6159

angelo@inf.ufrgs.br

Abstract: Despite the great number of papers which broach the consensus agreement, just a few specifically discuss the structure or the implementation of failure detectors. Even these papers usually present only the structure of the detectors, avoiding to present a comparison between the suggested detector and other existing ones. Actually, to our best knowledge, there is only one work that compares different failure detector models. We have chosen to use that paper as basis, and decided to make a comparison using practical systems, where the implementation issues and the operational system could influence both the performance of the detectors and the consensus termination time. This paper presents the preliminary results from our work, that allow us to determine which one is the most efficient, but specially allow us to learn how these detectors work under different failure situations and under diverse system loads.

Palavras−chave: detectores de defeitos, terminação do consenso, desempenho dos detectores

1 Introdução

Desde a definição dos detectores de defeitos por Chandra e Toueg [1], foram sugeridos muitos algoritmos para o problema do consenso, incluindo variações do modelo falhas ou do ambiente de aplicação. Esta variedade de algoritmos de consenso, no entanto, não foi acompanhada pelos detectores de defeitos, e por isso são relativamente poucas as propostas encontradas na literatura. De fato, os algoritmos de consenso normalmente são propostos de forma que seu funcionamento seja independente dos detector de defeitos que o suporta. Embora essa flexibilidade permita que se escolha livremente qual o detector a ser utilizado, é comum desconsiderar este aspecto quando se implementa um sistema.

Quando se considera um sistema real, existem muitos fatores que podem influenciar o processo de detecção. Enquanto a ocorrência de suspeitas incorretas afeta diretamente o tempo de terminação do consenso, existem outros aspectos resultantes da interação entre o detector e o ambiente (sistema operacional, rede de comunicação) que são responsáveis pela perda de desempenho. Por exemplo, o modelo de comunicação utilizado pelo detector tem impacto direto no tráfego na rede; além disso, o intervalo entre as mensagens também pode acarretar a sobrecarga dos recursos do sistema, pois tanto o meio de comunicação quanto o processamento das mensagens são afetados. Como cada detector reage de forma diferente aos diversos cenários de falhas, a melhor forma de manter o desempenho do consenso é através da escolha do detector de defeitos e dos parâmetros que melhor se adaptem a cada situação.

Essa comparação entre os detectores, entretanto, não costuma ser realizada pelos autores que especificam os algoritmos. De fato, foi encontrado apenas um trabalho que propõe a comparação entre detectores de defeitos, publicado por Sergent *et al.* [2]. Naquele artigo, os autores utilizaram simulações para avaliar o impacto do modelo de comunicação de três detectores selecionados sobre o tempo de terminação do consenso. Enquanto o trabalho de Sergent é muito útil para compreender a influência do modelo de comunicação usado por cada detector, muitos aspectos práticos que podem vir a interferir no desempenho dos algoritmos não puderam ser incluídos na especificação das simulações.

¤ bolsista CNPq

Assim, devido à limitação dos modelos de simulação, este trabalho procura avaliar, sob situações de falhas semelhantes, o comportamento dos detectores usados por Sergent (e também alguns modelos encontrados na literatura) através de implementações práticas, que estivessem sujeitas à influência do sistema operacional e da rede de comunicação.

Na seção 2 serão apresentados os modelos de detectores avaliados no trabalho, identificando suas diferenças e semelhanças. Na seção 3 serão discutidos o ambiente dos testes práticos, os cenários de falhas e as métricas usadas para validar os resultados. Na seção 4 serão apresentados alguns dos resultados obtidos, e uma breve análise destes dados. Finalmente, na seção 5, serão expostas as conclusões preliminares da dissertação.

2 Detectores Presentes na Literatura

Para a avaliação prática, foram utilizados alguns modelos de detectores encontrados na literatura. Uma das principais dificuldades nesta pesquisa deve−se à forma como os autores classificam os detectores. Aparentemente, não há nenhuma padronização, e por isso, é comum um detector ser referenciado por diversos nomes, especialmente os detectores mais tradicionais. Isso também se reflete nos nomes que os autores dão aos seus algoritmos, causando muitas confusões (por exemplo, o detector *Heartbeat* de Aguilera [3] tem um mecanismo muito diferente dos detectores comumente chamados *heartbeat*). Optou−se assim por classificar os detectores tradicionais segundo Felber [4], e manter a nomenclatura dos autores para os demais detectores.

Como o objetivo principal do trabalho é comparar o impacto de diferentes mecanismo de detecção e de comunicação, foram escolhidos exemplares de quatro grupos de detectores. No grupo dos detectores tradicionais, foram avaliados os detectores *Push* e *Pull*[1]. O grupo dos detectores especializados é representado pelos detectores *ad−hoc "no message"* e *ad−hoc "heart−beat"*, usados no trabalho de Sergent. Também foram considerados os detectores adaptativos[1], construídos a partir dos detectores *Push* e *Pull* e que permitem a modificação dinâmica do seu *time−out*. Por fim, foi analisado o detector *Heartbeat* [3], que usa uma variação do modelo de comunicação *gossip* [5]. Estes detectores trocam mensagens apenas com um subconjunto dos processos do sistema, chamados "vizinhos", procurando−se assim reduzir o tráfego de mensagens na rede de comunicação.

A detecção nos modelos tradicionais é apresentada nas Fig. 1 e 2. Eles diferem principalmente na forma como é controlado o *time−out*. Por exemplo, um detector *Pull* pode fazer suas requisições com um grande intervalo, porque como ele próprio controla o *time−out* das respostas, estas independem do intervalo entre as requisições. Já um detector *Push* deve enviar suas mensagens periodicamente, porque quem controla o *time−out* são os processos que recebem as mensagens. Estes detectores, com suas vantagens e desvantagens, são os mais comumente referenciados na literatura.

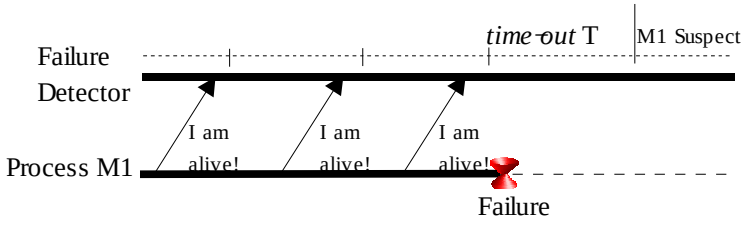

Fig. 1. Detecção de defeitos: o modelo *Push*

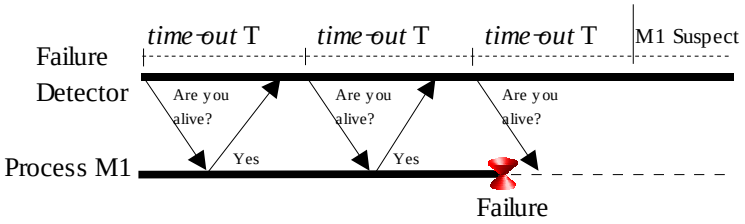

Fig. 2. Detecção de defeitos: o modelo *Pull*

1 Nomenclatura usada por Felber [4] para designar os detectores que enviam, respectivamente, mensagens do tipo "*I am alive*" (*heartbeat*) e *"Are you alive?*" (*interrogation*).

O grupo dos detectores especializados [2] conta com implementações feitas diretamente sobre o algoritmo de consenso, utilizando sua estrutura básica e mensagens como forma de otimizar o desempenho. Além disso, estes detectores são ativados somente em momentos específicos, quando o consenso necessita. O primeiro detector deste grupo, que também é o mais simples, é chamado ***ad−hoc "no message"***. Nele, não há envio de nenhuma mensagem de detecção, e o controle é feito sobre o *time−out* de uma mensagem específica da aplicação. No caso do consenso, o detector *ad−hoc "no message"* monitora a transmissão e recepção das mensagens `estimate` e `propose` do consenso. Este conjunto específico de mensagens monitoradas pelo detector são chamadas *mensagens críticas*; esta caracterização as difere das demais mensagens trocadas pelo algoritmo. Como não há nenhuma troca de mensagens de detecção, este modelo pode cometer falsas suspeitas quando um dos processos está bloqueado. No consenso, esta situação pode ocorrer quando o coordenador ainda não recebeu mensagens da maioria dos processos, resultando na suspeita mostrada na Fig. 3. Para resolver esse problema, há uma variação chamada ***ad−hoc "heart−beat"***, onde o coordenador envia mensagens "*I am alive"* enquanto a resposta crítica não é enviada (Fig. 4).

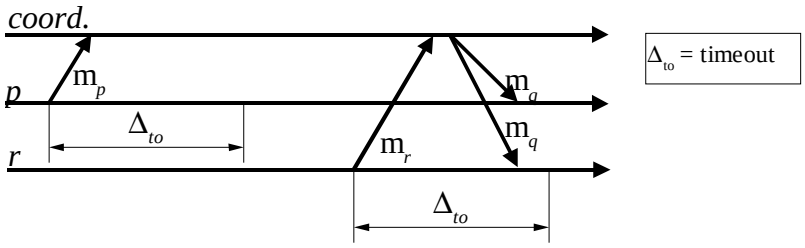

Fig. 3. Suspeitas incorretas com o detector *ad−hoc "no message"* [2]

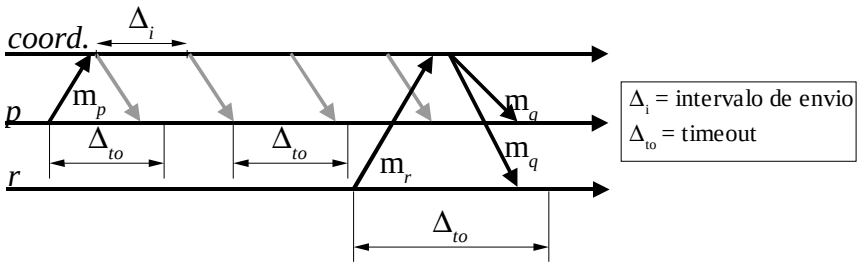

Fig. 4. Detecção com o detector *ad−hoc "heart−beat"* [2]

3 Premissas para o Experimento Prático

Para o experimento prático, optou−se por realizar os testes um ambiente onde os processos pudessem estar sujeitos à interação com o sistema operacional e a rede. Assim, empregou−se nos experimentos cinco máquinas com o sistema operacional Linux (Pentium II 233Mhz, 64MB, kernel 2.2.16), interconectadas por uma rede *Ethernet* 10−BaseTX não isolada. Para assegurar a avaliação dos resultados, e reduzir possíveis influências da interação com os usuários da rede, os testes foram realizados em horários pouco sociais, especialmente à noite e durante os fins de semana. Os algoritmos dos detectores e do consenso foram construidos utilizando apenas comunicação UDP. Os valores obtidos para cada combinação de situação de falhas e de parâmetros correspondem à média de no mínimo 1000 operações de consenso.

Além disso, como a comparação direta com os valores obtidos por Sergent é dificultada pela própria natureza do ambiente de testes, optou−se por analisar a curva de desempenho dos detectores, identificando seus comportamentos frente às diversas situações. Como essa análise provavelmente é independente da forma como se implementa os algoritmos, foi utilizada a linguagem *Java*, que agilizou o desenvolvimento dos diversos detectores.

Quanto às métricas utilizadas na avaliação, a mais importante foi o tempo de terminação do consenso, medida entre a requisição da operação de consenso e o retorno do valor acordado para a aplicação. Entretanto, foram consideradas também duas métricas adicionais, para validar estas observações. Através da ferramenta *top* do Linux, foi possível obter o tempo de CPU utilizado pela aplicação, bem

como a ocupação média de memória. Através do tempo de CPU, é possível avaliar a carga de processamento, descartando−se os momentos em espera, que normalmente são contabilizados no tempo de terminação do consenso. Já a ocupação da memória revela principalmente a sobrecarga da memória e dos *buffers* devido ao excesso de mensagens.

Para a avaliação, os detectores foram analisados sob três situações distintas de falhas, sendo que duas delas podem ser consideradas hipotéticas, e uma delas representa a operação normal dos detectores. As duas situações hipotéticas, representam respectivamente o melhor caso (***best case***) e o pior caso (***worst case***) que um algoritmo de consenso pode estar sujeito, em um *round*. No *best case*, não há nenhuma falha no sistema; isso permite analisar o impacto dos detectores no processamento global, uma vez que estes disputam recursos (rede, memória, tempo de CPU) com o algoritmo de consenso. Já o *worst case* trata sobre a ocorrência de falhas no coordenador de um *round*, onde pode ser avaliada a latência da detecção. Como os objetivos destes casos são bem específicos, os detectores são impedidos de realizar suspeitas errôneas, que poderiam influenciar nos resultados (por exemplo, estendendo o consenso por diversos *rounds*). Entretanto, não é possível ignorar a importância destas suspeitas incorretas, e por isso é feita a avaliação desta influência, em um ambiente semelhante à operação normal das aplicações (***normal case*),** onde não ocorrem falhas nos processos mas os detectores podem suspeitar erroneamente.

4 Análise de um dos detectores avaliados

Através da avaliação das três situações de operação foi possível obter os resultados que mostram o comportamento dos detectores sob as métricas consideradas. Com base nestes dados foi possível fazer uma avaliação do comportamento de cada detector, bem como compará−los. Tomando por exemplo um detector *Push*, pode−se observar seu comportamento através das três métricas. Na Fig. 5 são mostradas as curvas deste detector, avaliadas segundo o tempo de terminação do consenso.

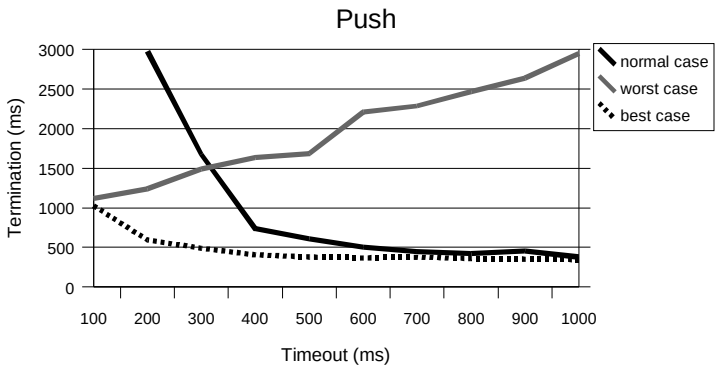

Fig. 5. Detector *Push*: tempo de terminação do consenso

Neste caso, pode−se verificar que o tempo de terminação da situação *best case* só é influenciado quando o *timeout* utilizado é menor do que 400 ms, uma vez que o envio mais mensagens gera mais *overhead*. Já a situação *worst case* é favorecida pela diminuição do *timeout*, uma vez que reduz a latência da detecção. Quando se analiza a situação *normal case*, pode−se verificar que para *timeouts* maiores do que 400 ms a curva segue muito próxima à da situação *best case*. Entretanto, quando foram usados *timeouts* menores, ocorrem mais suspeitas incorretas, levando o consenso a se estender por vários *rounds* (a situação *worst case* resolve o consenso em exatamente dois *rounds*).

A Fig. 6 mostra o comportamento deste detector quando avaliado pelo seu consumo de tempo de CPU. Neste gráfico a situação *best case* segue a mesma tendência já observada para o tempo de terminação do consenso. Isso se deve ao seu grau de atividade elevado (afinal, se não há nenhuma suspeita, todo processamento ocorre no menor tempo possível, sem interrupções). Já a situação *worst case* apresenta uma curva diferente da observada anteriormente. A explicação para esse comportamento é de que na situação *worst case* os algoritmos são controlados pelo máximo *timeout* possível, pois só conseguem finalizar o consenso quando o *timeout* é ultrapassado, e a suspeita notificada. No gráfico do tempo de CPU esse tempo de espera até atingir *timeout* não é computado, pois o detector não está fazendo nenhum processamento. Isso também determina a tendência de crescimento observada nas curvas, pois

no *worst case* o consenso fica bloqueado boa parte do tempo, enquanto na situação *best case* ele está operando juntamente com o detector, o que aumenta o já elevado *overhead* causado pelo grande número de mensagens. O comportamento da curva *normal case* segue o da curva do tempo de terminação do consenso, ou seja, para valores de *timeout* mais elevados o custo é baixo, uma vez que quase todas as operações conseguem finalizar em um único *round*. Entretanto, abaixo de um certo valor de *timeout* (aproximadamente 350 ms) o custo dos múltiplos *rounds* aumenta muito o tempo de CPU utilizado pelos algoritmos.

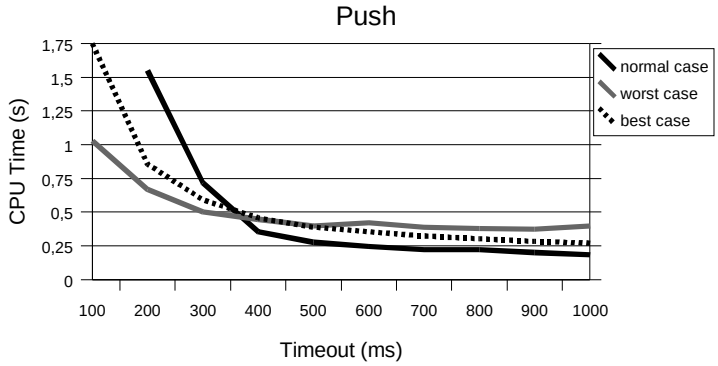

Fig. 6. Detector *Push*: tempo de CPU

Na Fig. 7 pode−se observar a média do consumo de memória deste detector, nas três situações de teste. Neste gráfico, o comportamento das situações *best case* e *worst case* foi semelhante; como estas situações enfocam propriedades características (*overhead* e latência da detecção), pode−se dizer que a pequena e constante diferença entre elas deve−se ao processamento extra que o *worst case* tem que fazer para tratar a detecção. O comportamento das curvas expressa o acúmulo de mensagens nos *buffers* e a necessidade de tratar mais mensagens devido à maior taxa de envio. Já o comportamento da situação *normal case* não segue essa tendência, ao menos em boa parte da faixa de testes. Provavelmente, o custo das suspeitas incorretas e do tratamento destas, mesmo quando raras, obriga o detector a manter uma ocupação mais constante da memória e apenas quando a sobrecarga causada pela maior taxa de mensagens se torna muito elevada, aumentando as suspeitas incorretas, é que o detector e o consenso consomem mais memória para realizar suas tarefas.

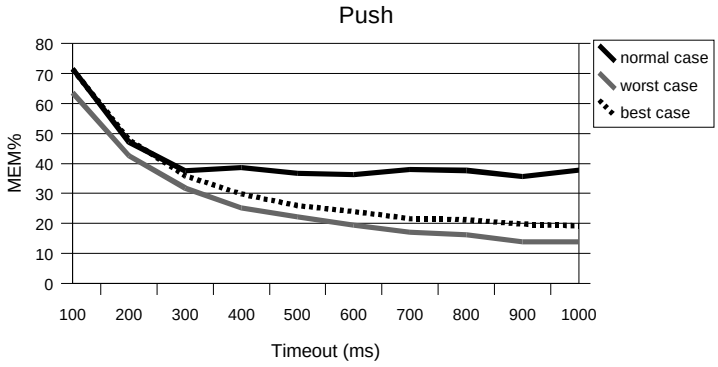

Fig. 7. Detector *Push*: consumo de memória

5 Conclusões

Este artigo apresentou os princípios fundamentais do trabalho de dissertação que está sendo elaborado, e uma amostra dos dados que estão sendo analisados. A primeira contribuição deste trabalho é realizar uma comparação entre os diversos modelos de detectores de defeitos, assim como comparar os resultados resultados obtidos através de simulações e de experiências práticas. De fato, pode−se inferir, em uma análise preliminar, que o modelo de comunicação (objetivo das simulações) é um dos grandes responsáveis pelo desempenho dos detectores, destacando−se sobretudo nas situações de maior sobrecarga. Entretanto, só otimizar a comunicação não basta, pois existem diversos fatores que também

influenciam no desempenho do consenso. Por exemplo, o tempo de terminação do consenso é intimamente relacionado ao custo de processamento das estruturas de controle dos detectores, que se reflete nas escolhas de implementação. Além disso, o impacto das suspeitas incorretas é bem notável, e deve−se procurar equilibrar uma baixa latência de detecção com uma boa precisão (*accuracy*).

Como resultados preliminares da dissertação, pode−se afirmar que de modo geral, os detectores *Push* parecem ser mais eficientes e custar menos ao sistema do que os detectores *Pull*. Além disso, se forem escolhidos parâmetros adequados ao ambiente, os detectores *Push* podem ser tão eficientes quanto o detector *ad−hoc "no message"*, sem apresentar suas desvantagens. O detector *ad−hoc "heart−beat"*, por outro lado, demonstrou ser muito caro ao sistema, por causa do seu mecanismo de ativação/desativação do detector a cada operação de consenso. Já os detectores adaptativos, apesar do desempenho um pouco menor, demostraram ser bem estáveis, independentemente da situação e da carga do sistema, com vantagem para o detector *AdaptivePush*. Por fim, o detector *Heartbeat* apresentou diversos problemas quanto à disseminação e ao processamento das mensagens, anulando assim as vantagens do modelo *gossip*, e deverá ser revisado para que possa ser utilizado em algum trabalho futuro.

Referências Bibliográficas

1. Chandra, T.D., Toueg S.: Unreliable Failure Detectors for Reliable Distributed Systems. Journal of the ACM, 43(2) (March 1996) 225−267
2. Sergent, N., Défago, X., Schiper, A.: Failure Detectors: implementation issues and impact on consensus performance. Technical Report, EPFL, Lausanne, Switzerland (1999)
3. Aguilera, M. K., Chen, W., Toueg, S.: Heartbeat: a timeout−free failure detector for quiescent reliable communication. Proceedings of the 11th International Workshop on Distributed Algorithms. (May 1997)
4. Felber, P.: The CORBA Object Group Service. PhD. Thesis. EPFL, Lausanne, Switzerland (1998)
5. Guo, K. H.: Scalable Message Stability Detection Protocols, PhD. Thesis, Cornell University (May 1998)
6. Chen, W., Toueg S., Aguilera M. K.: On the Quality of Service of Failure Detectors. Proceedings of the IEEE International Conference on Dependable Systems and Networks. New York.(June 2000)
7. Estefanel, L. A. B.: Detectores de Defeitos Não Confiáveis. Publicação Interna − UFRGS, 90 pg., UFRGS, Porto Alegre, Brasil. (Janeiro 2000). Disponivel em <*http://www.inf.ufrgs.br/~angelo/ publications/ TIDetectores.zip*>
8. Estefanel, L.A.B., Jansch−Pôrto, I.: Avaliação Prática de um Detector de Defeitos: teoria *versus* implementação. Anais do II Workshop de Testes e Tolerância a Falhas. Curitiba, Brasil. (Julho 2000)