PROPOSTA DE INTEGRAÇÃO DE UM SERVIÇO DETECTOR DE DEFEITOS ADAPTATIVO AO *KERNEL* DO *LINUX*

Gabriel P. Pedebos[1] e Rogério C. Turchetti[1]

[1]Centro Universitário Franciscano - Unifra

Santa Maria – Rio Grande do Sul - Brasil

{gpedebos@gmail.com, turchetti@unifra.br}

Abstract. *This paper describes a proposal of implementation of an adaptive failure detector service for the Linux Kernel. The failure detector assists the development of distributed systems using Linux operating system, because works as support tool for the activities of fail tolerance. Such service monitors the process using timeouts. In this work the timeout is adapted dynamically with base in the reply time (round trip time) of processes and making use of mathematics models known by predictors.*

Resumo. *Este trabalho descreve uma proposta de implementação de um serviço detector de defeitos adaptativo para o núcleo do sistema operacional Linux. O detector de defeitos proposto auxilia o desenvolvimento de sistemas distribuídos utilizando Linux, pois funciona como uma ferramenta de suporte a atividades de tolerância a falhas. Tal serviço monitora os processos participantes do sistema através da utilização de timeouts, ou seja, quando um processo monitorado não responde a um processo monitor, dentro de um tempo limite, este é considerado suspeito. Neste modelo o tempo limite de espera é adaptado dinamicamente com base no tempo de resposta de um determinado processo em rodadas de detecção anteriores.*

1 INTRODUÇÃO

Diversas operações em sistemas distribuídos exigem a utilização de protocolos de consenso, para atingir um acordo entre as réplicas. Porém em um sistema assíncrono, este protocolo não pode ser resolvido devido ao problema da impossibilidade descrito por Fisher [FISCHER 1985]. Assim, detectores de defeitos não confiáveis [CHANDRA 1996] são utilizados como uma alternativa a este problema, pois permitem que operações de consenso tenham continuidade mesmo na ocorrência de falhas em um dos processos do sistema.

Neste sentido o presente trabalho propõe a integração de um serviço detector de defeitos ao *Kernel* do *Linux*, servindo como ferramenta de auxílio a tolerância a falhas em sistemas distribuídos que necessitam o máximo de confiabilidade e por isso fazem uso de técnicas de tolerância a falhas. As técnicas utilizadas atualmente, como por exemplo, a Redundância Modular Tripla, *checkpoint* coordenado e comunicação de grupo confiável são implementadas como aplicações que rodam sobre o sistema operacional. Como alternativa, este trabalho propõem que a atividade de detecção de um defeito, utilizada por várias técnicas de tolerância a falhas, seja adicionada ao núcleo do sistema operacional, melhorando o desempenho da detecção.

O detector de defeitos monitora os processos através da utilização de *timeouts*, ou seja, caso um processo não responda em um determinado limite de tempo, este é considerado suspeito. A adaptação dinâmica do *timeout* se faz necessária, pois ajusta o *timeout* de acordo com a carga dos processos e do canal de comunicação em um determinado instante de tempo, diminuindo as chances de falsas suspeitas. Como resultado têm-se melhorias na *QoS* (*Quality of Service*) do detector. Os cálculos para predizer os *timeouts* serão baseados em quatro preditores que serão descritos no decorrer deste artigo.

O restante deste trabalho está estruturado da seguinte forma: na seção 2 é definido o modelo de sistema; na seção 3, apresenta-se o objetivo dos detectores de defeitos e os modelos de detecção implementados; na seção 4, descrevem-se os modelos matemáticos para previsão do *timeout*, juntamente com a métrica de comparação entre eles que foi utilizada; a seção 5 apresenta os aspectos de implementação e por fim, os resultados dos testes de detecção e predição, bem como a análise de cada um; as considerações finais e os trabalhos futuros são apresentados na seção 6.

2 MODELO DE SISTEMA E DEFINIÇÕES

O sistema proposto é formado por um conjunto de processos, os quais comunicam-se entre si através de troca de mensagens utilizando um canal de comunicação. Neste trabalho considera-se o termo processo como cada um dos computadores que fazem parte do sistema. Caso ocorra uma falha em um destes processos e esta faça com que o sistema não envie mensagens ao detector de defeitos, assume-se que o processo como um todo falhou. Neste trabalho utiliza-se o modelo assíncrono em um ambiente sujeito a falhas por colapso. Um processo pode possui dois estados: *operacional* ou *suspeito*. O processo operacional é aquele que envia periodicamente uma mensagem de vida dentro do tempo especificado ao detector, já um processo suspeito é caracterizado pela ausência das mensagens de vida, fazendo com que o mesmo seja considerado suspeito de ter falhado.

3 DETECTORES DE DEFEITO

Com a finalidade de resolver o problema do consenso em ambientes assíncronos, Chandra e Toueg (1996), propõem detectores de defeito não confiáveis, pois podem cometer enganos. Estes detectores têm o objetivo de monitorar e verificar um estado para um determinado processo do sistema.

O serviço detector de defeitos proposto monitora constantemente os processos do sistema, através da utilização de *timeouts* (tempo limite de resposta). Quando um processo não responde dentro deste limite de tempo o mesmo é considerado suspeito, até que volte a responder ao detector. O detector de defeitos possui uma lista de processos suspeitos. Tal lista consiste na visão momentânea que um processo monitor possui dos processos monitorados. Assim quando um processo não responde dentro do tempo esperado o detector adiciona-o em sua lista. Cada processo possui seu próprio detector de defeitos monitorando os demais, ou seja, todos monitoram todos. Nas seções seguintes serão apresentados os dois modelos de detectores de defeitos implementados neste trabalho, sendo eles: *push* e *pull*.

3.1 Detector estilo *push*

No algoritmo de detecção *push* [FELBER 1998], as mensagens de controle geradas pelos detectores seguem o mesmo sentido do fluxo das informações, desta forma, o processo monitorado envia periodicamente uma mensagem de vida ("*I'm alive*") ao detector de defeitos, indicando que o mesmo está operacional. A figura 1 exemplifica o modelo *push* através do fluxo de mensagens entre o processo monitor *q* e o processo monitorado *p*, bem como seu comportamento diante o atraso no recebimento de uma mensagem.

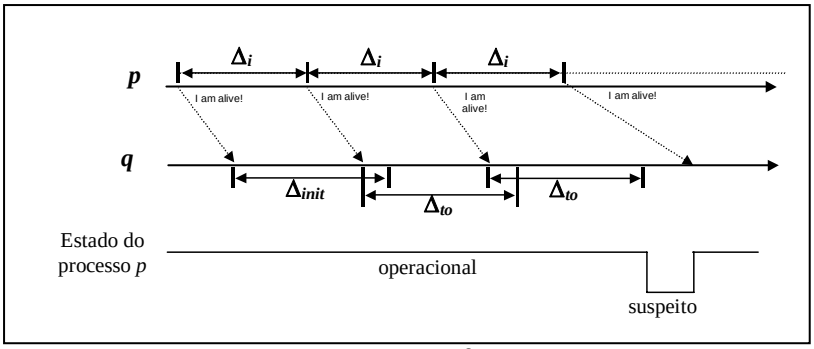


Figura 1. Detector de defeitos modelo *push.*

3.2 Detector estilo *pull*

No algoritmo de detecção *pull* [FELBER 1998], o processo monitor envia mensagens de requisição de vida (*Liveness Request*) aos processos monitorados, estes por sua vez, devem responder ao processo monitor dentro de um tempo limite, como mostra a figura 2. Assim, os processos monitorados não precisam estar ativos nem ter conhecimento sobre a freqüência de envio de requisições de vida por parte processo monitor. Esta característica facilita o desenvolvimento da aplicação, visto que toda configuração é centralizada no processo monitor. Por outro lado, a desvantagem deste modelo está relacionada ao número de mensagens trocadas entre as duas partes, pois para cada requisição enviada uma nova mensagem de resposta é gerada.

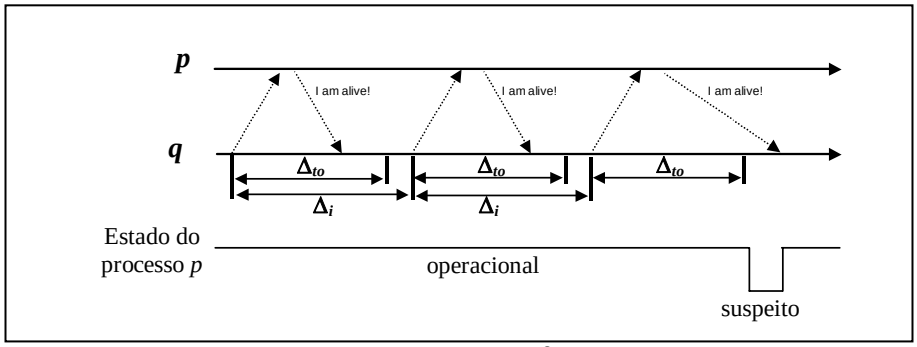


Figura 2. Detector de defeitos modelo *pull*.

4 MODELOS DE PREDIÇÃO DO *TIMEOUT*

Através da análise de séries temporais é possível fazer previsões de valores futuros de uma série [MORETIN 2004]. Baseando-se no trabalho de NUNES (2003), apresenta-se neste trabalho a utilização de séries temporais como método de cálculo previsão para o próximo *timeout* em detectores de defeitos adaptativos. Foram

implementados quatro preditores ou métodos de previsão que serão apresentados na seqüência desta seção.

O preditor *LAST* é o primeiro método de previsão do *timeout* apresentado neste trabalho, sendo que preditor assume o último tempo de resposta do processo monitorado como estimativa para o próximo *timeout*. O preditor *MEAN* baseia-se em todos os tempos de resposta de um determinado processo. Este preditor realiza uma média aritmética de todos os tempos de resposta, assumindo tal valor como próximo *timeout*.

Assim como o preditor *MEAN*, o preditor *WINMEAN* realiza o cômputo de sua estimativa baseado na média dos tempos de resposta de rodadas anteriores, porém este preditor considera apenas uma parte destes de valores, conhecida por janela. Esta janela compreende os *n* últimos valores amostrados. O preditor *LPF* (*Low Pass Filter*) emprega a técnica de um filtro passa baixa para estimar valores futuros do *timeout*, ou seja, filtra o comportamento transiente dando um peso maior ao valor médio dos dados considerados não transientes ao computar a estimativa para o próximo valor [NUNES 2003]. A função para cálculo do próximo *timeout* (tr_{n+1}) utilizando o preditor *LPF* é a seguinte:

$$tr_{n+1} = \alpha * tr + (1 - \alpha) * \overline{tr}$$

O termo *tr* da equação representa o último tempo de resposta amostrado, enquanto que $\overline{tr}$ representa o último valor estimado pelo preditor *LPF*.

4.1 Métrica de comparação

Para verificar qual dos preditores apresenta os melhores resultados foi definido um critério de comparação entre eles. Para tanto, considera-se o erro quadrático médio como métrica de comparação entre os preditores, desta forma o preditor que apresentar o menor erro quadrático médio acumulado após um intervalo de monitoramento, será considerado o melhor entre os preditores avaliados. A equação para o cálculo do erro quadrático (*eqm*) médio é a seguinte:

$$eqm = (tr - \overline{tr})^2$$

onde, o termo *tr* representa o tempo que o processo levou para responder enquanto $\overline{tr}$ é último valor estimado pelo preditor.

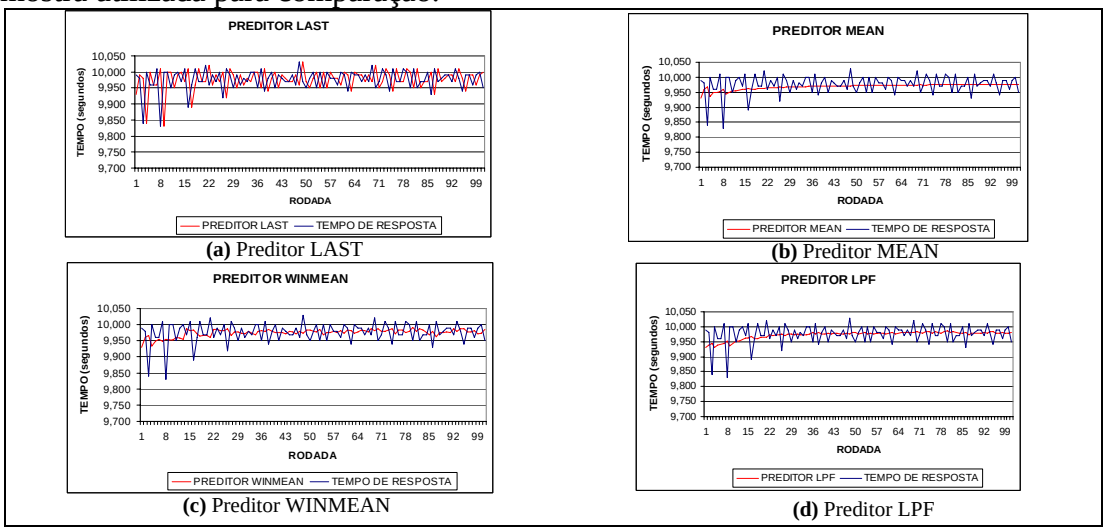
5 TESTES

Para a implementação do serviço detector de defeitos, foi utilizada a linguagem de programação *C*, sendo que esta linguagem é mais adequada para a integração com o *Kernel* do sistema operacional em questão. O processo de implementação do serviço detector de defeitos encontra-se concluído, restando a fase de modularização do mesmo junto ao *Kernel* do *Linux*. O serviço de detecção é executado em todos os computadores que compõem o sistema, funcionando como processo monitor e monitorado ao mesmo tempo. O ambiente de testes do serviço detector de defeitos constitui-se de um *cluster* de computadores (estações de trabalho) do tipo *Beowulf* [DANTAS 2005]. Para a realização dos testes cada um dos computadores do *cluster* possui seu próprio detector de defeitos executando e gravando localmente os resultados da detecção, sendo estes dados utilizados posteriormente para análise.

5.1 Análise dos resultados

Para realizar a comparação entre os preditores implementados, as informações de 100 rodadas de detecção foram coletadas, desta forma o serviço detector de defeitos foi adaptado de modo que este calculasse simultaneamente o próximo tempo de resposta baseado nos preditores *LAST*, *MEAN*, *WINMEAN* e *LPF*. Como critério de comparação foi utilizado o cálculo do erro quadrático de cada um dos preditores.

A figura 3 apresenta os gráficos dos valores para o próximo *timeout* estimados pelos quatro preditores implementados, sobrepostos ao gráfico do tempo de resposta da amostra utilizada para comparação.



(a) Preditor LAST **(b)** Preditor MEAN

(c) Preditor WINMEAN **(d)** Preditor LPF

Figura 3. Gráficos dos valores estimados pelos preditores.

Visto que o preditor *LAST* (Figura 3a) assume o último tempo de resposta como próximo valor amostrado, este preditor teria uma maior precisão em ambientes, nos quais, os tempos de resposta se mantivessem constantes, durantes períodos maiores de tempo, sendo que através do gráfico do tempo de resposta, é possível perceber que são raros os casos em que este se mantém constante, de uma rodada para a outra. Como esperado, o preditor *MEAN* mantém sua previsão entre os valores médios dos tempos de resposta, tal característica é notada na Figura 3b. Neste método de previsão do *timeout*, os valores passados possuem grande influência na previsão de valores futuros, fazendo com que o *timeout* demore em adaptar-se a velocidade atual do sistema. O preditor *WINMEAN* (Figura 3c) surge como opção a este problema, pois este realiza uma média aritmética sob um conjunto dos últimos tempos de resposta de um determinado processo, assim a previsão do *timeout* adapta-se mais rapidamente às variações no ambiente. Para realização dos testes foi utilizada uma janela de tamanho 5, porém sugere-se como trabalhos futuros testar a utilização de outros valores. O gráfico do preditor *LPF* (Figura 3d) demonstra que este preditor mantém sua estimativa entre os valores médios dos tempos de resposta, mas com maior variação que o preditor *MEAN*.

O preditor *LPF* apresentou a menor diferença entre o tempo de resposta previsto e o tempo em que o processo realmente respondeu. Porém, com uma pequena diferença o preditor *MEAN* apresenta o segundo menor erro quadrático médio seguido pelos preditores *WINMEAN* e *LAST* respectivamente.

Quanto aos modelos de detecção implementados, tanto o modelo *push* quanto o modelo *pull* cumprem integralmente com as suas especificações e que as variações no

tempo de detecção entre estes dois modelos são geradas devido à diferença no modo de detecção de defeitos utilizado. Desta forma, a escolha do modelo de detecção a ser utilizado deve levar em consideração outros aspectos como a facilidade de definição dos parâmetros de configuração e o número de mensagens trocadas entre os processos.

5 CONSIDERAÇÕES FINAIS

Este trabalho descreveu uma proposta de integração de um serviço detector de defeitos adaptativo e configurável ao *Kernel* do *Linux*. Com a adaptação dinâmica do *timeout* através de modelos matemáticos, o tempo de detecção é mais adequado e o número de falsas suspeitas são reduzidos, como resultado tem-se um detector com melhor qualidade de serviço. Todos os métodos de previsão do *timeout* e modelos detecção de defeitos estão implementados e funcionando corretamente. Desta forma, os usuários poderão escolher qual dos métodos de previsão do *timeout* e detecção de defeitos desejam utilizar, tornando-se assim, um serviço configurável.

Como trabalhos futuros pretende-se realizar a integração do serviço detector de defeito desenvolvido ao *Kernel*, bem como uma interface gráfica para facilitar a configuração do serviço. Implementação de outras técnicas de tolerância a falhas no kernel, e quem sabe o desenvolvimento de sistema específica para sistemas distribuídos e com alta disponibilidade.

REFERÊNCIAS BIBLIOGRÁFICAS

CHANDRA, T. D.; TOUEG, S. **Unreliable failure detectors for reliable distributed systems**. Journal of the ACM, [S.l.], v.43, n.2, p.225.267, jan 1996.

DANTAS, M. **Computação Distribuída de alto desempenho – Redes, Clusters e Grids Computacionais**. Axcel Books. Rio de Janeiro. 2005.

FELBER, P.; GUERRAOUI, R.; SCHIPER, A. The Implementation of a CORBA Object Group Service. **Theory and Practice of Object Systems**, 1998.

FISCHER, M.; LYNCH, N.; PATERSON, M. **Impossibility of Distributed Consensus with One Faulty Process**. Journal of the ACM, New York, v.32, p.374-382, Apr. 1985.

JALOTE, Pankaj. **Fault Tolerance in Distributed Systems**. Englewood Cliffs:Prentice Hall, 1994.

MORETTIN, P. A., TOLOI, C. M. C.. **Análise de séries temporais**. São Paulo, SP : Edgard Blücher , 2004.

NUNES, R. C.; JANSCH-PÔRTO, I. A Lightweight Interface to Predict Communication Delays Using Time Series. In: LADC, 2003.

RIBEIRO, U. E. **Sistemas Distribuídos – Desenvolvendo Aplicações de Alta Performance no Linux.** Rio de Janeiro – RJ: Axcel Books, 2ª ed., 2005.

TURCHETTI, R. **Uma Nova Abordagem Para Redução de Mensagens de Controle em Detectores de Defeitos**. Dissertação de mestrado. PPGEP-UFSM, 2006.