Uma Estratégia Adaptativa para Melhorar a Precisão do Timeout de Detectores de Falhas na Internet

Rogério C. Turchetti[2], **Elias P. Duarte Jr.**[1]

[1] Departamento de Informática – Universidade Federal do Paraná (UFPR)
Caixa Postal 19.081 – 81.531-980 – Curitiba – PR – Brasil

[2]CTISM - Universidade Federal de Santa Maria (UFSM)
Avenida Roraima, 1000 – 97105-900 – Santa Maria – RS – Brasil

turchetti@redes.ufsm.br, elias@inf.ufpr.br

Abstract. *Timeouts are frequently used by failure detectors to monitor processes of distributed systems. To avoid both false failure suspicions and long delays to detect failures that have actually occurred, it is important to compute the timeout accurately. In this work, a strategy called $tuning_\phi$ is proposed which adapts the timeout according to the communication patterns observed. In this way the computed timeouts are able to better reflect the varying behavior of the underlying communication channel. In particular, we adapted the strategy proposed by Jacobson by replacing constants by automatically adjusted weights. The proposed strategy was evaluated in the context of detecting process failures in the Internet. Experiments show that $tuning_\phi$ nearly eliminates false suspicions, presenting less than 1% of the number of false suspicions raised by Jacobson's strategy. In addition, $tuning_\phi$ was also able to reduce the time to detect failures keeping a low average time to correct false suspicions.*

Resumo. *O timeout é um mecanismo regularmente utilizado por detectores de falhas para o monitoramento de processos de sistemas distribuídos. Para evitar falsas suspeitas e a espera prolongada para detectar falhas que efetivamente ocorreram, é importante que o timeout seja preciso. Neste trabalho é proposta uma estratégia denominada de $tuning_\phi$ que reajusta o valor do timeout de acordo com os tempos de comunicação obtidos, buscando refletir o comportamento real da rede. Em especial, adaptamos o cálculo proposto por Jacobson ajustando automaticamente pesos para constante que, no algoritmo original, são valores fixos. A estratégia proposta foi avaliada no contexto de detecção de falhas de processos na Internet. Os experimentos demonstraram que $tuning_\phi$ reduz expressivamente o número de falsas suspeitas, não atingindo 1% do número de falsas suspeitas cometidas pelo algoritmo original. Além disso, $tuning_\phi$ apresentou uma redução no tempo de detecção de falhas mantendo um bom desempenho no tempo médio para correção de falsas suspeitas.*

1. Introdução

Detectores de falhas são blocos fundamentais para auxiliar no desenvolvimento de aplicações distribuídas tolerantes a falhas. Um detector provê informações a respeito dos estados dos processos de um sistema. Os detectores de falhas foram propostos por Chandra e

Toueg [Chandra and Toueg 1996] e sua abstração possibilita encapsular as premissas temporais dos sistemas assíncronos e, dependendo das propriedades que oferecem, permitem resolver problemas que não poderiam ser solucionados de forma determinística em tais ambientes [Fischer et al. 1985]. Um detector de falhas é não confiável pois pode cometer erros, suspeitando erroneamente de um processo que não está falho. Porém, ao perceber seu erro, corrige-o imediatamente, deixando de suspeitar do respectivo processo. Em geral, o monitoramento realizado pelos detectores de falhas é baseado em troca de mensagens. O processo monitorado envia periodicamente mensagens de *heartbeat*. Caso o detector de falhas não receba um *heartbeat* dentro de um limite de tempo especificado, conhecido como *timeout*, o respectivo processo monitorado passa a ser considerado suspeito de ter falhado.

Do ponto de vista da precisão das informações fornecidas por um detector de falhas, podemos dizer que uma parcela dos erros cometidos pelos detectores está estreitamente relacionada à estratégia adotada para o cômputo do *timeout*. Em outras palavras, uma característica importante dos detectores de falhas é a sua capacidade em prever o instante preciso de tempo em que a próxima mensagem de monitoramento será recebida. Em geral, as abordagens são baseadas em estimativas que visam corrigir o tempo de espera de acordo com o comportamento do ambiente de execução. Dentre as diversas abordagens utilizadas na literatura para o cômputo do *timeout* [Chen et al. 2000, Nunes and Jansch-Porto 2004, Falai and Bondavalli , Dixit and Casimiro 2010, Tomsic et al. 2015] a que se destaca, por ser simples e adaptativa é o algoritmo proposto por Jacobson (1988), originalmente implementado no protocolo TCP (*Transmission Control Protocol*). Neste algoritmo, Jacobson indica pesos para algumas constantes que influenciam diretamente no valor final do *timeout*. Entretanto, alguns autores utilizam, além dos pesos padrões para as constantes, outros valores que possibilitam obter resultados diferentes dos originais. Por exemplo, nos trabalhos em [Bertier et al. 2003, Dixit and Casimiro 2010, de Sá and de Araújo Macêdo 2010, Moraes and Duarte Jr. 2011] o cômputo do *timeout* é auxiliado por outros valores que objetivam melhores resultados para casos específicos, e.g., reduzir o tempo para a detecção de uma falha.

Segundo Dixit e Casimiro [Dixit and Casimiro 2010], dependendo dos pesos utilizados para as constantes do algoritmo do Jacobson, o cálculo pode transformar o *timeout*, mais ou menos agressivo. Além disso, um *timeout* mais agressivo possibilita um menor tempo de detecção de falhas. Por outro lado, um valor menos agressivo, pode evitar falsas suspeitas. Em outras palavras, se for considerado que a rede se comporta sem muitas oscilações ou, percebe-se uma redução nos atrasos de comunicação, nestas condições definir um *timeout* grande não trará benefícios, pois o tempo para a detecção de uma falha será maior. Por outro lado, se o comportamento da rede variar bastante ou, percebe-se que os atrasos aumentam linearmente, definir um *timeout* pequeno aumenta as chances do algoritmo de detecção gerar falsas suspeitas. De acordo com a estratégia original proposta por Jacobson e utilizada em diversos trabalhos da literatura, uma vez estabelecidos os pesos das constantes utilizadas para o cômputo do *timeout*, os valores permanecem inalterados ao longo do tempo, mesmo diante de variações no comportamento do sistema.

Com base no exposto, o objetivo do presente trabalho é propor uma estratégia (denominada $tuning_{\phi}$) que reajusta o valor do *timeout* de acordo com os tempos de comu-

nicação calculados por $tuning_\phi$. Em especial, ajusta-se o valor da constante responsável por multiplicar o valor obtido através das variações de atraso de comunicação. O ajuste é baseado em uma análise de tendência que retrata o comportamento de uma série temporal. Uma série temporal é uma sequência de observações coletadas em intervalos determinados ao longo do tempo [Shumway and Stoffer 2006] que representa, no contexto deste trabalho, os tempos de comunicação entre o transmissor e o receptor. Com a estratégia proposta, os cálculos previstos para o *timeout* seguem mais próximos do comportamento do ambiente de execução. Além disso, a própria estratégia é responsável por indicar um peso para a constante, sem a necessidade de uma indicação prévia.

Portanto, o objetivo é avaliar o comportamento do ambiente em execução, considerando os tempos de comunicação nas trocas de mensagens entre um processo transmissor e um receptor, para utilizá-los em futuras previsões. Em outras palavras, a ideia é avaliar o comportamento atual do ambiente com base em uma análise de séries temporais, para prever a tendência do comportamento futuro. A tendência pode ser calculada com base nos valores do coeficiente angular (se positivo indica uma tendência crescente, se negativo indica tendência decrescente) e do coeficiente linear, obtidos através dos valores da série temporal [Shumway and Stoffer 2006].

A estratégia $tuning_\phi$ é implementada em um algoritmo para detecção de falhas responsável por monitorar os processos através de trocas de mensagens e respeitando um limite de tempo previamente estabelecido. Os experimentos são conduzidos com base no monitoramento de processos na Internet. Os resultados permitem verificar que a estratégia $tuning_\phi$ reduz de forma expressiva o número de falsas suspeitas, não chegando a atingir 1% do número de falsas suspeitas cometidas pelo algoritmo original, que mantêm o peso da constante. Além disso, $tuning_\phi$ apresentou uma redução no tempo de detecção de falhas, mantendo um bom desempenho no tempo médio para correção de falsas suspeitas.

O restante do trabalho está organizado da seguinte forma. A Seção 2 apresenta uma breve descrição sobre o controle do *timeout* e métricas que possibilitam avaliar o seu cômputo. Na Seção 3 é inicialmente apresentada a estratégia proposta por Jacobson, na sequência é introduzida a estratégia $tuning_\phi$ que permite calcular um *timeout* adaptando valores para a constante ϕ. Por fim, os experimentos e as conclusões são descritos na Seção 4 e na Seção 5, respectivamente.

2. Definição e Utilização de *Timeouts*

O *timeout* é um mecanismo bastante útil em sistemas onde se pode utilizá-lo para impor limites temporais tanto para a comunicação entre processos quanto para a execução de tarefas. O *timeout* é um evento que é escalonado usando um temporizador o qual expira após um limite de tempo estabelecido. Se nenhuma informação for recebida dentro deste intervalo de tempo, uma ação é executada como parte integrante deste controle [Kebarighotbi and Cassandras 2011]. Por exemplo, no contexto das redes de computadores, uma importante tarefa é garantir que os dados enviados por um transmissor foram, de fato, recebidos pelo receptor remoto. Uma possível estratégia para garantir tal tarefa é, para cada pacote transmitido, utilizar um algoritmo que aguarda uma confirmação de retorno enviada pelo receptor remoto. Na ausência de qualquer confirmação, o algoritmo utiliza um tempo de retransmissão que permite 'garantir' que os dados não foram recebidos pelo receptor remoto. Dessa maneira, pacotes perdidos podem ser recuperados

utilizando o mecanismo de retransmissão de pacotes [Kesselman and Mansour 2005], tal como implementado pela camada de transporte [Braden 2017].

Note que os mecanismos baseados em *timeout* são susceptíveis a variações no atraso, sendo um grande desafio prever, de forma precisa, o instante de tempo em que a informação será recebida. Determinar um valor preciso para o *timeout* não é uma tarefa trivial, pois o tempo de comunicação pode variar de acordo com influências de atrasos de processamento ou sobrecarga nos canais de comunicação. Escolher um *timeout* adaptativo é uma alternativa, pois ele é baseado em estimativas que visam corrigir seu valor de acordo com o comportamento do ambiente ou com base em outras informações, como exemplo, considerar as necessidades de cada aplicação. O cálculo proposto por Jacobson [Jacobson 1988] é um exemplo de uma estratégia de controle de tempo adaptativa.

Outro contexto em que o mecanismo de *timeout* é empregado e que é de fundamental importância, é nos algoritmos para detecção de falhas em sistemas distribuídos [Chandra and Toueg 1996]. Um detector de falhas monitora os estados dos processos para indicar se um processo está ou não suspeito de ter falhado. Se a mensagem chegar dentro do intervalo de tempo estabelecido, o respectivo processo não será suspeito pelo detector. Por outro lado, se a mensagem não for recebida dentro do intervalo de tempo previamente estipulado, o respectivo processo terá seu estado alterado para suspeito. Neste contexto, diversos trabalhos foram propostos em busca de alternativas que possibilitam calcular um *timeout* que possa ser adaptado de acordo com mudanças no ambiente [Chen et al. 2000, Bertier et al. 2003, Dixit and Casimiro 2010, Xiong et al. 2012, Tomsic et al. 2015, Turchetti et al. 2016]. Vale ressaltar também que, dentre as diversas formas empregadas para o cômputo do *timeout* nestes trabalhos, a própria estratégia de Jacobson já foi também utilizada em serviços para detecção de falhas.

Uma maneira de analisar a precisão do cálculo do *timeout* é utilizar métricas clássicas do contexto de detectores de falhas. Isto é, para realizar experimentos comparativos entre diferentes estratégias, são utilizadas métricas propostas por Chen, Toueg e Aguilera [Chen et al. 2000]. As métricas permitem avaliar a qualidade de serviço de cada algoritmo considerando a velocidade com que conseguem suspeitar de processos falhos e a exatidão destas suspeitas. Note que a velocidade permite avaliar quão rápida uma falha pode ser detectada, enquanto que a exatidão avalia quão bem o algoritmo não comete enganos. Em ambos os casos o resultado é influenciado pelo valor obtido no cálculo do *timeout*.

A Figura 1 mostra as transições dos estados de um processo monitorado que falha no instante t_4, bem como a visão dos estados apresentada pelo processo monitor (detector de falhas). Para medir a qualidade do serviço oferecido pelos algoritmos que implementam um mecanismo de *timeout*, as seguintes métricas primárias são definidas e ilustradas na Figura 1:

- **Tempo de detecção (***Detection Time* **-** T_D**).** Mede o tempo decorrido desde o instante em que o processo monitorado falha até o instante em que o detector de falhas suspeita permanentemente do processo ($T_D = t_5 - t_4$).
- **Duração de um erro (***Mistake Duration* **-** T_M**).** Representa o tempo que o processo monitor leva para corrigir uma suspeita incorreta ($T_M = t_2 - t_1$).
- **Tempo para recorrência ao erro (***Mistake Recurrence Time* **-** T_{MR}**).** Determina o tempo entre dois erros consecutivos cometidos pelo processo monitor. O T_{MR} é

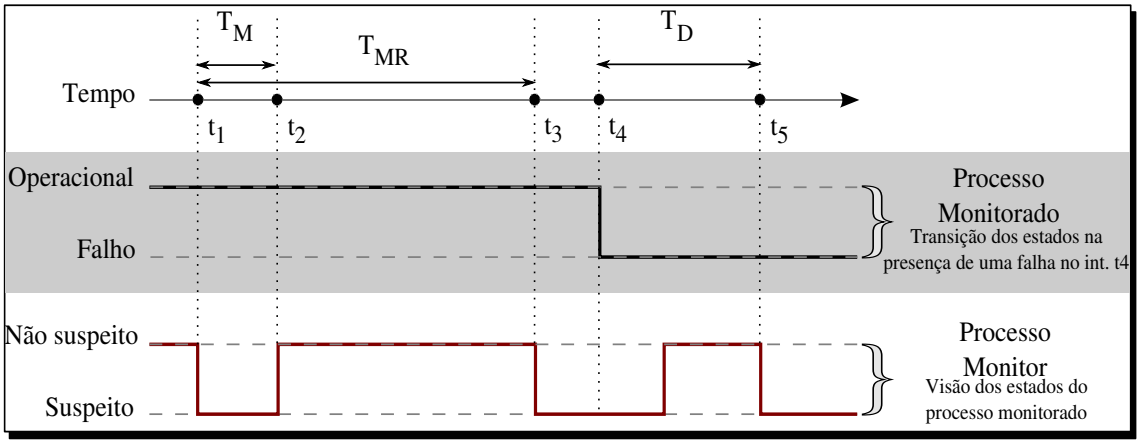


Figura 1. Métricas para avaliar a velocidade e a exatidão.

variado e inicia no instante em que ocorre a primeira suspeita errada até a seguinte ($T_{MR} = t_3 - t_1$).

Com a finalidade de avaliar a estratégia proposta neste trabalho, as métricas T_D e T_M, são utilizadas nos experimentos apresentados na Seção 4.

3. Uma Estratégia para Ajustar o *Timeout* de Acordo com Previsões de Comportamento da Rede

Nesta seção é detalhado o cálculo proposto por Jacobson, o qual será identificado pela abreviação de Jac (seção 3.1), e na sequência (seção 3.2) é descrita a estratégia $tuning_\phi$ proposta neste trabalho.

3.1. O Algoritmo Jac

A seguir é detalhado o cálculo proposto por Jacobson que originalmente foi utilizado para determinar o tempo de duração para a retransmissão de uma mensagem. Este tempo é compreendido como sendo o tempo de transmissão até a confirmação de uma mensagem, também chamado de RTT (*Round Trip Time*). Vale ressaltar que neste trabalho, a falta de uma resposta de um processo dentro do intervalo de tempo calculado, não implica na retransmissão da mensagem novamente, mas sim indica uma falha do processo que não respondeu dentro do intervalo de tempo esperado. Em especial, utilizamos o algoritmo Jac para prever o valor do próximo *timeout* com a finalidade de detectar os estados dos processos, indicando se o processo monitorado está ou não suspeito de ter falhado. O valor do *timeout* é recalculado a cada troca de mensagem de monitoramento.

O cálculo apresentado a seguir utiliza como referência os instantes de tempo em que as mensagens de confirmação das requisições de vida (*liveness request*) são recebidas, isto é, $diff_{(k)}$ apresentado na Expressão 1 representa a diferença entre a '*k-ésima*' e a '*k-ésima*-1' mensagens:

$$diff_{(k)} = msg_k - msg_{k-1} \quad (1)$$

Na expressão (2), $delay_{(k+1)}$ é o valor do atraso que é calculado em termos da diferença $diff_{(k)}$, e γ representa o peso das novas amostras:

$$delay_{(k+1)} = (1 - \gamma) \cdot delay_{(k)} + \gamma \cdot diff_{(k)} \quad (2)$$

$var_{(k+1)}$ representa a variação da diferença:

$$var_{(k+1)} = (1 - \gamma) \cdot var_{(k)} + \gamma \cdot (|diff_{(k)} - delay_{(k+1)}|) \quad (3)$$

Por fim, $\alpha_{(k+1)}$ é obtido através da expressão 4, onde β, ϕ e γ são constantes que recebem os seguintes valores: $\beta = 1$, $\phi = 4$ e $\gamma = 0.1$ [Jacobson 1988].

$$\alpha_{(k+1)} = \beta \cdot delay_{(k+1)} + \phi \cdot var_{(k+1)} \quad (4)$$

De acordo com os cálculos do Jac é possível observar que α está relacionado à periodicidade das mensagens de *liveness request* transmitidas, com sua respectiva resposta ($diff$), bem como às variações de tempo durante o tráfego das mensagens entre o detector de falhas e os processos monitorados ($delay$ e var). Por fim, com o cálculo apresentado acima, o objetivo é utilizá-lo para determinar o *timeout* que será empregado pelos detectores de falhas no monitoramento de processos.

3.2. A Estratégia $tuning_{\phi}$

No contexto das estratégias implementadas para determinar o valor do intervalo do *timeout*, uma importante característica é configurá-lo de maneira adaptativa. A ideia é buscar funcionalidades que permitam, com precisão, prever o instante de tempo em que a próxima mensagem de monitoramento será recebida pelo processo. Quando uma estimativa é calculada de maneira errada, em geral busca-se corrigir o valor previsto de acordo com o novo comportamento observado no ambiente.

Neste trabalho é utilizado um algoritmo para detecção de falhas que determina o intervalo de *timeout* conforme atrasos na comunicação. Isto é, o *timeout* representado por α é adaptável a cada mensagem recebida de acordo com a Expressão 4. Como apresentado na seção anterior, no algoritmo Jac o autor indica pesos para constantes, como exemplo ϕ=4. Entretanto, percebe-se que alguns autores utilizam outros valores, como exemplo, em seus experimentos Bertier [Bertier et al. 2003] utiliza ϕ=2. Dixit e Casimiro [Dixit and Casimiro 2010] propõem utilizar ϕ com valores que podem ser pré-fixados com 1, 2 e 4, sendo que quanto menor o valor, como exemplo $\phi = 1$, transforma o cálculo em um *timeout* mais agressivo, possibilitando um menor tempo de detecção de falhas. Por outro lado, um valor menos agressivo, como $\phi = 4$, ou maior, pode evitar falsas suspeitas.

Além disso, se o comportamento da rede permanece sem oscilações, ou percebe-se uma redução nos atrasos de comunicação, definir um valor maior para ϕ, nestas condições, não trará benefícios pois o tempo para a detecção de uma falha será maior. Por outro

lado, se o comportamento da rede variar bastante ou, percebe-se que os atrasos aumentam linearmente, definir um valor pequeno para ϕ aumenta as chances do detector de falhas gerar falsas suspeitas.

Com base no exposto, a ideia é propor uma estratégia (denominada $tuning_{\phi}$) que ajusta o valor de ϕ de acordo com os tempos de comunicação percebidos através das ações de monitoramento, variando o valor entre ϕ_{min} e ϕ_{max}. O ajuste é auxiliado por uma análise de tendência que retrata o comportamento de uma série temporal representada pelos tempos de comunicação. Uma série temporal é uma sequência de observações coletadas em intervalos determinados ao longo do tempo [Shumway and Stoffer 2006].

O objetivo é avaliar o comportamento nos tempos de comunicação da troca de mensagens durante o monitoramento dos processos, para utilizá-lo em futuras previsões. Em outras palavras, a ideia é avaliar o comportamento atual do ambiente com base em uma análise de séries temporais, para prever a tendência do comportamento futuro. A tendência pode ser calculada com base nos valores do coeficiente angular (se positivo indica uma tendência crescente, se negativo indica tendência decrescente) e do coeficiente linear obtidos através dos cálculos que consideram os valores da série temporal. Então, a equação de tendência é calculada conforme expressão apresentada a seguir:

$$T = \ell + \partial.t \tag{5}$$

Na Expressão (5), T é o valor da tendência, ∂ representa o coeficiente angular, ℓ é o coeficiente linear e t é o valor do tempo que representa o período na amostra a ser calculado.

A seguir é descrita a equação que permite obter o valor do coeficiente linear apresentado pela Expressão (6) e para o cômputo do coeficiente angular apresentado na Expressão (7):

$$\ell = \frac{n.\sum_{i=1}^{n}(t_i.y_i) - \sum_{i=1}^{n} t_i.\sum_{i=1}^{n} y_i}{n.\sum_{i=1}^{n}(t_i^2) - (\sum_{i=1}^{n} t_i)^2} \tag{6}$$

$$\partial = \frac{\sum_{i=1}^{n} y_i - \ell.\sum_{i=1}^{n} t_i}{n} \tag{7}$$

n é o número de elementos que corresponde ao tamanho da série temporal (usamos n = 5 nos experimentos), y_i representa cada elemento registrado na série temporal, como por exemplo a duração para a resposta de uma mensagem de monitoramento e, t_i é o período que está associado ao elemento y_i, que nada mais é do que a sequência das mensagens registradas na série. Com os valores de ℓ e ∂ obtidos é possível então aplicar os valores na equação de tendência apresentada na Expressão (5), para prever o próximo valor de y_i conforme o seu respectivo período (t_i).

Portanto, utilizando o cálculo de tendência busca-se prever o próximo período da série e, consequentemente, calcular o valor apropriado para ϕ. Em outras palavras,

basta definir o valor mínimo e máximo (ϕ_{min} e ϕ_{max}) e a própria estratégia $tuning_\phi$ é responsável por indicar um peso para a constante ϕ, tornando a constante com valores que são adaptados de acordo com as previsões obtidas através da Expressão (5). Além disso, $tuning_\phi$ isenta a responsabilidade da indicação de um valor prévio e fixo para a constante ϕ.

Para ilustrar o funcionamento da estratégia $tuning_\phi$, considere o Algoritmo 1 onde são apresentadas, resumidamente, as funcionalidades de um detector de falhas que realiza o monitoramento dos processos utilizando a estratégia $tuning_\phi$ para calcular o valor do próximo *timeout*. De acordo com o algoritmo, há um processo monitor denominado de p_j e um processo monitorado chamado de p_i. Inicialmente, todo processo monitorado p_i (linha 1) está incluso na lista de suspeitos ($suspect$) e nenhum processo correto é conhecido (linhas 3 e 4).

Algorithm 1: Algoritmo que implementa a estratégia $tuning_\phi$.

```
1   Every process p_i ∈ Π:
2   upon event ⟨ _Init ⟩ do
3     suspect := Π;
4     correct := ∅;
5     φ_min := 1;                          ▷ φ_min e φ_max são definidos a priori
6     φ_max := 4;
7     {∃ φ ∈ ℤ+ : φ_min ≤ φ ≤ φ_max};      ▷ indica que φ={1, 2, 3, 4}
8   upon event ⟨ _Send | p_j, type, p_i⟩ do
9     run ⟨ send | self, ARE YOU ALIVE, p_i⟩;   ▷ m{src, type, dst}
10    correct := correct - {p_i};
11    wait for α
12    if (p_i ∉ correct) then
13        suspect := suspect ∪ {p_i};
14  upon event ⟨ _Receive | p_i, YES I AM, p_j⟩ do
15  correct := correct ∪ {p_i};
16  suspect := suspect - {p_i};
17  T := ∂ + ℓ .t;                  ▷ calcula a tendência para o próximo período
18  φ := ⌈|((T + var) − delay)/var|⌉;   ▷ result. do arredondamento p/ cima
19  if (φ > φ_max) then
20      φ := φ_max;
21  α := delay + φ * var;
```

Periodicamente, p_j executa o evento $\_Send$ para enviar uma mensagem de requisição de vida ao processo p_i, aguardando por um intervalo de tempo α. Se p_i responder a requisição de vida, p_j atualiza o estado de p_i e, com base em uma análise de tendência, p_j tenta prever o valor do próximo instante de tempo (linha 17). Com o valor previsto para o tempo de chegada da próxima mensagem, calcula-se o valor de ϕ que deve pertencer aos naturais positivos e possuir um valor entre ϕ_{min} e ϕ_{max} indicados nas linhas 5 e 6 respectivamente. Vale ressaltar que a decisão por utilizar os valo-

res apresentados nas linhas 5 e 6 do algoritmo foi feita com base na literatura estudada [Jacobson 1988, Bertier et al. 2003, Moraes and Duarte Jr. 2011].

Na linha 18 é determinado o valor de ϕ, que é obtido considerando a seguinte condição:

$$\alpha \geq T + var \tag{8}$$

onde var é um valor obtido da Expressão (3). Além disso, substitui-se α na Expressão (8) pela Expressão (4) e isolando-se a constante ϕ, obtendo-se assim o peso para ϕ de acordo com o valor previsto para o próximo período.

Por fim, utiliza-se o peso encontrado para ϕ na expressão apresentada na linha 21, obtendo-se então o valor corrente de α. Note que α, neste caso, representa o próprio valor do intervalo de *timeout*.

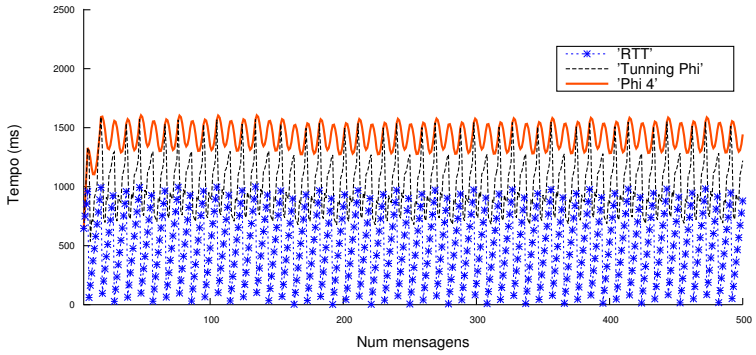
A seguir, são apresentados resultados experimentais que demonstram a eficiência da estratégia $tuning_\phi$ descrita nesta seção.

4. Resultados Experimentais

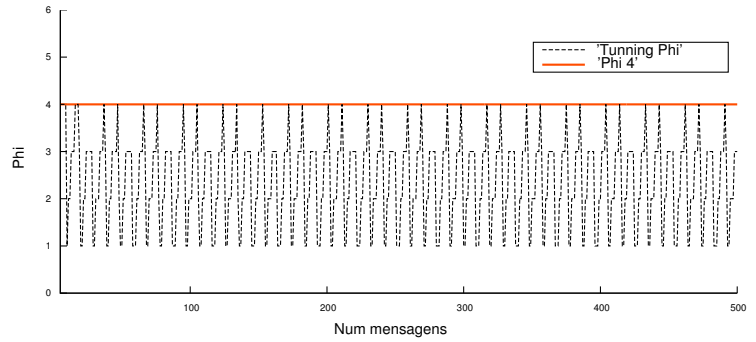
Para avaliar a estratégia $tuning_\phi$ proposta neste trabalho, utilizou-se traces reais disponibilizado em uma base de dados criada por Hayashibara et. al [Hayashibara et al. 2004]. Esta base armazena informações de tempos de comunicação entre dois computadores na Internet, posicionados em países diferentes, sendo um computador localizado na Suíça (no *Swiss Federal Institute of Technology in Lausanne*) e o outro localizado no Japão (no *Japan Advanced Institute of Science Technology*). A base descreve os tempos de comunicação entre os dois computadores desconsiderando mensagens perdidas.

Vale ressaltar que os experimentos apresentados nesta seção comparam o algoritmo originalmente proposto por Jacobson, isto é, algoritmo que não adapta o valor de ϕ, com o algoritmo da estratégia $tuning_\phi$ apresentado na Seção 3, ambos executando um serviço para detecção de falhas. O objetivo dos primeiros experimentos apresentados nas Figuras 2 e 3 é demonstrar como a estratégia $tuning_\phi$ trabalha, adaptando o valor de ϕ de acordo com o comportamento percebido no ambiente de execução. Para estes experimentos foram analisados os resultados das 500 primeiras mensagens armazenadas na base de dados, descrita no parágrafo anterior. Na Figura 2(a) é possível observar que o RTT possui um comportamento bastante variado. Verifica-se que, quase periodicamente, o RTT atinge um valor aproximado de 1000 ms e cai bruscamente para um valor próximo de 100 ms. Neste cenário, o *timeout* computado com o valor de ϕ fixo, isto é, $Jac_{\phi=4}$, possui variações que acompanham as oscilações nos tempos de comunicação. Podendo-se verificar que o algoritmo Jac é originalmente adaptativo.

Por outro lado, analisando os valores apresentados pela estratégia $tuning_\phi$ é possível perceber que os valores do *timeout* acompanham com mais fidelidade, as oscilações dos tempos de comunicação. A variação do valor de ϕ calculado pela estratégia $tuning_\phi$ é mais perceptível na Figura 2(b). Neste gráfico, o comportamento quase que periódico do RTT torna a troca de pesos para a adaptação do ϕ praticamente periódica também.



(a) Comparação do *timeout* usando $\phi = 4$ com a estratégia $tuning_\phi$.



(b) Peso de ϕ ao longo do tempo.

Figura 2. Comparação da estratégia $tuning_\phi$ com o ϕ fixo.

Por fim, é possível observar que a troca do peso da variável ϕ, realizada pela estratégia $tuning_\phi$, torna a função mais próxima do comportamento real da rede.

O gráfico da Figura 3 mostra os valores dos tempos para a correção de falsas suspeitas (ver Seção 2). O experimento compara a estratégia $tuning_\phi$ com a utilização de ϕ fixo, isto é, $Jac_{\phi=1}$ e $Jac_{\phi=4}$. Para simular falsas suspeitas, a cada 30 mensagens recebidas foi duplicado o tempo de comunicação para a próxima mensagem. Conforme mostra o gráfico da Figura 3, o valor de $Jac_{\phi=4}$ apresentou 6 falsas suspeitas com um T_M médio de 297,9 ms, já com $Jac_{\phi=1}$ observou-se 145 falsas suspeitas mas com um T_M médio de 161,28 ms.

Utilizando a estratégia $tuning_\phi$, pode-se observar que ela apresentou valores de T_M para ϕ variando entre 1 a 3, não apresentando nenhuma ocorrência de falsa suspeita com o valor de $\phi = 4$. Entretanto, vale ressaltar que a estratégia $tuning_\phi$ atribuiu valores para $\phi = 4$, como mostrado na Figura 2(b). Além disso, foram cometidas 14 falsas suspeitas, com um T_M médio de 274,07 ms, levemente inferior ao apresentado por $Jac_{\phi=4}$.

É possível observar que com o cômputo do $Jac_{\phi=1}$ diminui-se o tempo médio para correção de falsas suspeitas, apresentando o menor valor obtido, $T_M = 0,38$ ms. Entretanto, aumenta-se consideravelmente o número de falsas suspeitas e apresenta também o

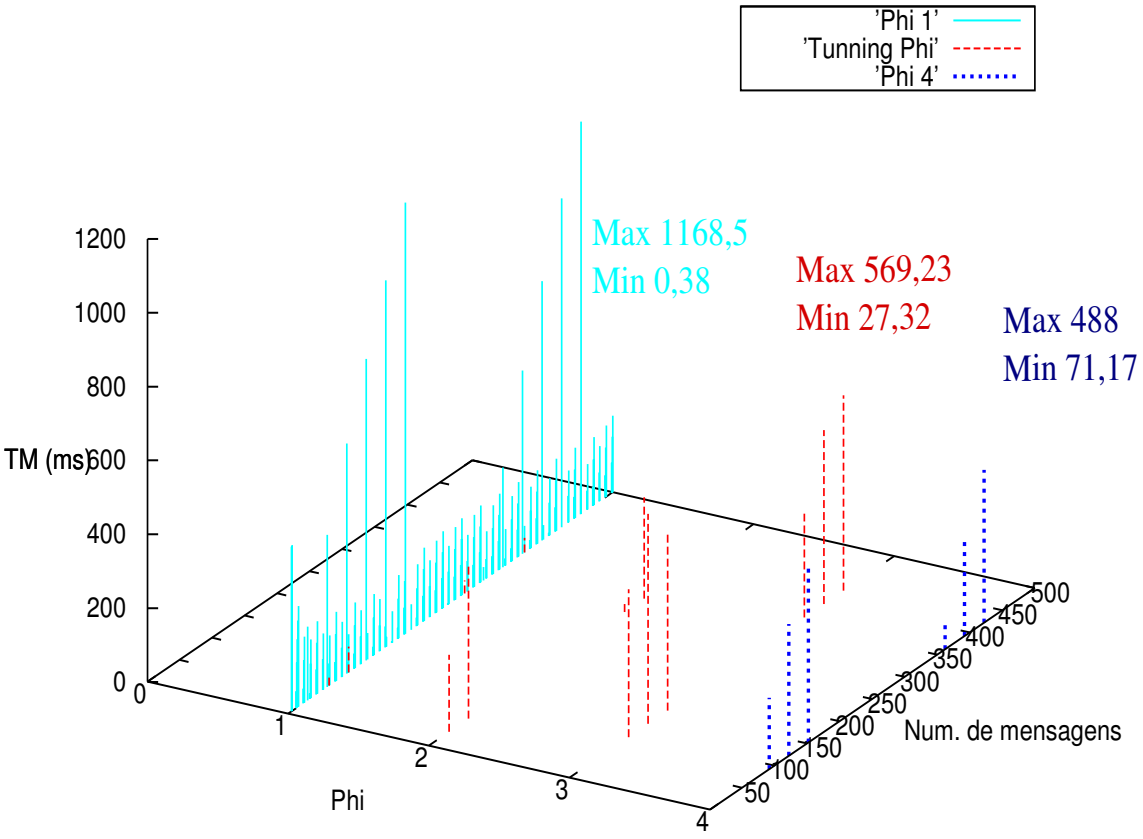


Figura 3. T_M para a estratégia $tuning_\phi$ e ϕ fixo.

maior T_M computado, atingindo a máxima de 1168,1 ms. A estratégia $tuning_\phi$ consegue reduzir o número de falsas suspeitas se comparada a $Jac_{\phi=1}$, mas não foi melhor do que $Jac_{\phi=4}$, embora o T_M tenha sido ligeiramente menor para a estratégia $tuning_\phi$.

Com base nestes experimentos é possível observar que a estratégia $tuning_\phi$ apresentou valores intermediários entre o número de falsas suspeitas e os tempos para as correções de falsas suspeitas. Os resultados apresentados na Tabela 1 permitem ver com clareza os resultados para as falsas suspeitas. Para estes resultados foi utilizada toda a base de dados, ou seja, informações coletadas em 7 dias de monitoramento, obtendo um total de 5.822.520 mensagens.

Tabela 1. Comparação do número de falsas suspeitas cometidas.

	Jac	$tuning_\phi$
ϕ	**Falsas Suspeitas**	**Falsas Suspeitas**
1	1805372	38
2	1777	25
3	58	3
4	47	38

Na Tabela 1 é realizada uma comparação entre $tuning_\phi$ e Jac considerando o número de falsas suspeitas cometidas. Nesta comparação é possível observar uma grande vantagem para a estratégia $tuning_\phi$. Vale ressaltar que as falsas suspeitas não foram força-

das como no caso do experimento da Figura 3. Aqui, as falsas suspeitas ocorrem quando o cálculo do próximo *timeout* é menor do que o tempo para a chegada da próxima mensagem. É possível observar na Tabela 1 que a estratégia $tuning_\phi$ é muito superior quando comparada com o valor de ϕ fixo (Jac). De fato, foram cometidas um total de 1.807.254 falsas suspeitas utilizando $Jac_{\phi=1}$, ao passo que, utilizando a estratégia $tuning_\phi$ foram observadas 104 falsas suspeitas. O que equivale a dizer que o número de falsas suspeitas cometidas pela estratégia $tuning_\phi$ foi menor do que 1%, mais precisamente 0,57% comparada ao ϕ fixo com valores de 1, 2, 3 e 4.

Os experimentos apresentados na Figura 4 mostram um comparativo considerando o tempo de detecção (T_D, ver Seção 2) para uma falha. O T_D é computado a cada falsa suspeita cometida pelo detector de falhas. Neste caso, para $\phi = 3$ a estratégia $tuning_\phi$ apresentou um T_D relativamente maior. A razão para este valor é que foram detectadas somente 3 falsas suspeitas (ver Tabela 1) sendo todas com valores altos, o que não possibilitou reduzir a média do T_D para este experimento.

Entretanto, observa-se que para os demais casos a estratégia $tuning_\phi$ apresentou um T_D menor. Para justificar estes valores, considere o seguinte caso: para uma mesma falha detectada, a estratégia $tuning_\phi$ pode estar usando um valor de ϕ menor do que utilizando ϕ fixo, quando isso ocorre, o valor do *timeout* é menor, por consequência a detecção da falha será em menor tempo. Já para $\phi = 1$ ocorrem casos em que uma falha computada pelo ϕ fixo não é calculada pela estratégia $tuning_\phi$, pois esta pode estar utilizando um valor de *timeout* maior não caracterizando uma suspeita e não computando o T_D. Em linhas gerais, o T_D total para Jac_ϕ foi de 5330,74 ms e usando $tuning_\phi$ foi de 4761,37 ms apresentando uma redução de aproximadamente 10%.

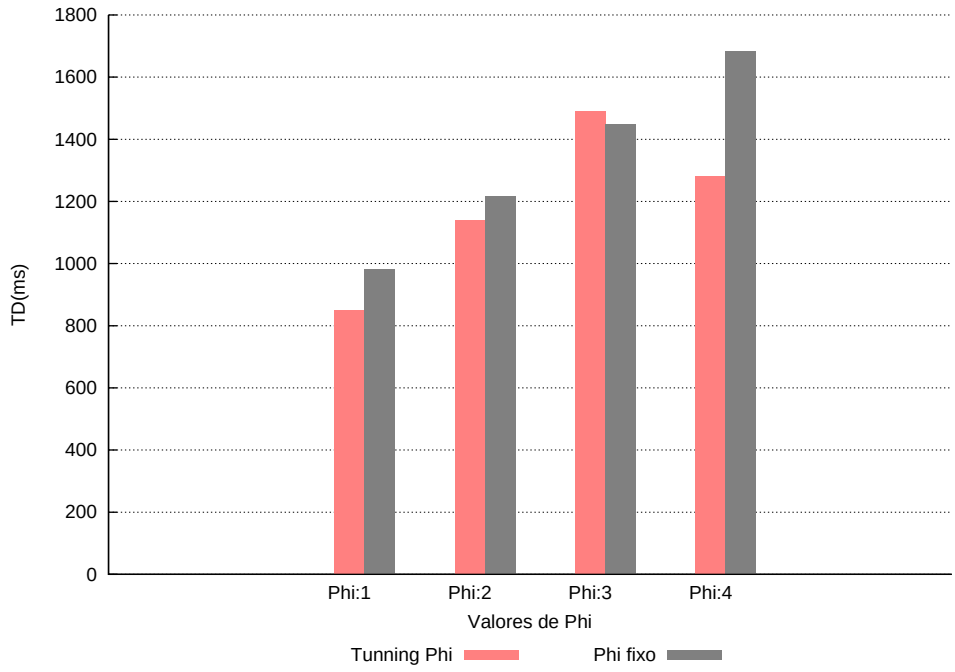


Figura 4. T_D para a estratégia $tuning_\phi$ e ϕ fixo.

Em linhas gerais, podemos afirmar que é uma boa escolha, considerando o número de falsas suspeitas, utilizar $Jac_{\phi=4}$, pois há um total de somente 47 falsas suspeitas

cometidas. Por outro lado, ao analisarmos o T_D com $Jac_{\phi=4}$ podemos notar que o valor aumenta consideravelmente se comparado ao tempo médio obtido por $tuning_\phi$, ou seja, o valor é reduzido em 41,1% do tempo para a detecção de uma falha.

Note que a ideia implementada pela estratégia $tuning_\phi$ concentra-se em evitar que o peso para a constante ϕ precise ser definido previamente. Sendo assim, vale ressaltar que a definição da análise de tendência utilizada para as previsões computadas, não levou em consideração aspectos mais detalhados, tais como a análise de autocorrelação nos resíduos, métodos para detecção da autocorrelação, sazonalidades, entre outros. Porém, com base nos experimentos realizados nesta seção, se pode constatar a validação da ideia proposta, onde claramente são apresentas diversas melhorias com a utilização da estratégia $tuning_\phi$. Em especial, se pode concluir que a estratégia $tuning_\phi$ apresentou um desempenho consistente, reduzindo o número de falsas suspeitas sem aumentar o T_D e com bom desempenho no tempo médio para correção de falsas suspeitas.

5. Conclusão

Neste trabalho foi proposta $tuning_\phi$, uma estratégia que calcula o valor do *timeout* para ser utilizado para o monitoramento dos estados dos processos em sistemas distribuídos. Em especial, $tuning_\phi$ ajusta o valor da constante usada no algoritmo de Jacobson responsável por multiplicar o valor obtido através das variações de atraso de comunicação. O ajuste é baseado em uma análise de tendência que retrata o comportamento de uma série temporal. Com a estratégia proposta, os cálculos previstos para o *timeout* seguem mais próximos do comportamento do ambiente de execução. Além disso, a própria estratégia é responsável por indicar um peso para a constante, sem a necessidade de uma indicação prévia.

Para demonstrar a eficiência da estratégia $tuning_\phi$, seu desempenho foi avaliado utilizando um serviço para detecção de falhas comumente empregado em sistemas distribuídos. Os experimentos realizados foram conduzidos utilizando um *trace* que descreve os tempos de comunicação entre dois processos na Internet. Considerando os resultados experimentais, $tuning_\phi$ apresentou um excelente desempenho reduzindo de forma expressiva o número de falsas suspeitas, não chegando a atingir 1% do número de falsas suspeitas cometidas pela estratégia original, isto é, quando a constante ϕ foi utilizada com valores fixos. A estratégia $tuning_\phi$ apresentou ainda uma redução no tempo de detecção de falhas mantendo um bom desempenho no tempo médio para correção de falsas suspeitas. Por fim, outra vantagem que vale ser ressaltada é que $tuning_\phi$ isenta a responsabilidade da indicação de um valor prévio e fixo para a constante ϕ. Trabalhos futuros incluem a realização de novas análises para a definição do cálculo que realiza a previsão do comportamento da rede, tais como considerar uma análise de autocorrelação nos resíduos, métodos para detecção da autocorrelação, sazonalidades, entre outros.

Referências

Bertier, M., Marin, O., and Sens, P. (2003). Performance Analysis of a Hierarchical Failure Detector. In *International Conference on Dependable Systems and Networks (DSN)*.

Braden, R. (acessado em 14 dez. 2017). Requirements for internet hosts - communication layers. https://tools.ietf.org/html/rfc1122.

Chandra, T. D. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of ACM*, 43(2).

Chen, W., Toueg, S., and Aguilera, M. K. (2000). On the Quality of Service of Failure Detectors. In *International Conference on Dependable Systems and Networks (DSN)*.

de Sá, A. S. and de Araújo Macêdo, R. J. (2010). Qos self-configuring failure detectors for distributed systems. In *International Conference on Distributed Applications and Interoperable Systems (DAIS)*. Springer-Verlag.

Dixit, M. and Casimiro, A. (2010). Adaptare-FD: A Dependability-Oriented Adaptive Failure Detector. *Symposium on Reliable Distributed Systems*.

Falai, L. and Bondavalli, A. Experimental evaluation of the qos of failure detectors on wide area network. In *2005 International Conference on Dependable Systems and Networks (DSN)*, pages 624–633.

Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of ACM*, 32(2).

Hayashibara, N., Défago, X., Yared, R., and Katayama, T. (2004). The Φ accrual failure detector. In *23rd International Symposium on Reliable Distributed Systems (SRDS)*.

Jacobson, V. (1988). Congestion Avoidance and Control. In *Symposium Proceedings on Communications Architectures and Protocols*.

Kebarighotbi, A. and Cassandras, C. (2011). Timeout control in distributed systems using perturbation analysis. In *50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*.

Kesselman, A. and Mansour, Y. (2005). Optimizing tcp retransmission timeout. In *Proceedings of The 4th International Conference on Networking (ICN)*.

Moraes, D. M. and Duarte Jr., E. P. (2011). A failure detection service for internet-based multi-as distributed systems. In *17th IEEE International Conference on Parallel and Distributed Systems ICPADS*.

Nunes, R. C. and Jansch-Porto, I. (2004). Qos of timeout-based self-tuned failure detectors: the effects of the communication delay predictor and the safety margin. In *International Conference on Dependable Systems and Networks (DSN)*, pages 753–761.

Shumway, R. H. and Stoffer, D. S. (2006). *Time Series Analysis and Its Applications With R Examples*. Springer.

Tomsic, A., Sens, P., Garcia, J., Arantes, L., and Sopena, J. (2015). 2w-fd: A failure detector algorithm with qos. In *2015 IEEE International Parallel and Distributed Processing Symposium*, pages 885–893.

Turchetti, R. C., Jr., E. P. D., Arantes, L., and Sens, P. (2016). A QoS-configurable failure detection service for internet applications. *J. Internet Services and Applications*, 7(1):9:1–9:14.

Xiong, N., Vasilakos, A., Wu, J., Yang, Y., Rindos, A., Zhou, Y., Song, W.-Z., and Pan, Y. (2012). A self-tuning failure detection scheme for cloud computing service. In *IEEE 26th International Parallel Distributed Processing Symposium (IPDPS)*.