# Detectores de Defeitos Autonômicos para Sistemas Distribuídos

**Alirio Santos de Sá, Raimundo José de Araújo Macêdo**

[1]Laboratório de Sistemas Distribuídos - LaSiD
Departamento de Ciência da Computação
Universidade Federal da Bahia, Salvador, Bahia, Brasil

{alirio,macedo}@ufba.br

***Abstract.*** *Existing failure detection approaches for distributed systems do not support the automatic configuration of failure detectors from Quality of Service (QoS) requirements. Such approaches only use QoS metrics to evaluate failure detection performance without considering self-configuration. However, when the behavior of the computing environment is unknown and changes over time, self-configuration of failure detector is a basic issue for building applications with response time and high availability requirements. In this paper we present the design and implementation of a novel autonomic failure detector based on feedback control theory, capable of self-configuration using previously defined QoS requirements.*

***Resumo.*** *As propostas tradicionais de detecção de defeitos em sistemas distribuídos não suportam a configuração automática de seus parâmetros a partir de métricas de qualidade de serviço (QoS). Tais abordagens consideram apenas o uso das métricas de QoS para a avaliação de desempenho sem se preocupar com a configuração automática do detector. Todavia, quando as características do ambiente são desconhecidas e podem mudar, a auto-configuração se torna uma habilidade fundamental para a construção de aplicações com requisitos de tempo de resposta e disponibilidade. Este artigo apresenta o projeto de um detector de defeitos baseado em teoria de controle, com capacidade de auto-configuração a partir de requisitos QoS.*

## 1. Introdução

Infra-estruturas para serviços de tecnologia da informação dos dias atuais (*data centers*, *grids*, etc.) utilizam diferentes componentes de hardware e software com intuito de prover serviços altamente disponíveis, seguros e escaláveis para aplicações com necessidades variadas em termos de qualidade de serviço (QoS). Para serem bem sucedidas em seus propósitos, essas infra-estruturas precisam estar equipadas com mecanismos que garantam rápida reação e recuperação em contingências de falhas, maximizando a disponibilidade continuada dos serviços. Nesse contexto, detectores de defeitos têm papel fundamental uma vez que são utilizados para ativar processos de recuperação. Conseqüentemente, tais mecanismos vêm sendo objeto de intensa pesquisa nas últimas décadas [Chandra and Toueg 1996, Chen et al. 2002, Bertier et al. 2003, Macêdo and Lima 2004, Nunes and Jansch-Pôrto 2004, Falai and Bondavalli 2005, Xiong et al. 2006, Satzger et al. 2008, de Sá and Macêdo 2009].

Em sistemas distribuídos sobre redes de computadores, detectores de defeitos são implementados através da troca de mensagens entre processos monitores e monitorados. Para garantir uma rápida recuperação em contingências de falhas, o período entre mensagens de monitoramento tem que ser o mais curto possível, o que, paradoxalmente, pode acarretar um maior consumo de recursos e conseqüente comprometimento do tempo de resposta das aplicações em geral e, em particular, do próprio mecanismo de detecção e recuperação (tolerância a falhas). Complicações adicionais aparecem quando as características do ambiente computacional ou das aplicações podem mudar em tempo de execução (o que é comum em infra-estruturas modernas como servidores de *cloud computing*, por exemplo). Nesses cenários, ajustar o período de monitoramento de forma automática passa a ser um enorme desafio, ainda não enfrentado de forma satisfatória na literatura existente. A grande maioria dos trabalhos publicados têm como foco principal a detecção adaptativa, a qual se utiliza de mecanismos de predição para *timeouts* de detecção, sem, contudo, se preocupar com o ajuste dinâmico dos períodos de monitoramento [Chen et al. 2002, Bertier et al. 2003, Nunes and Jansch-Pôrto 2004, Falai and Bondavalli 2005]. Mais ainda, os poucos trabalham que abordam ajustes de período de monitoramento, o fazem sem levar em conta as métricas de QoS de detecção de defeitos amplamente divulgadas na literatura [Mills et al. 2004, Xiong et al. 2006, Satzger et al. 2008].

Uma das maiores dificuldades na realização desses detectores dotados da capacidade de auto-configuração é a modelagem do comportamento dinâmico do sistema distribuído, uma vez que, na maioria dos ambientes abertos, é muito difícil caracterizar tais ambientes através de distribuições de probabilidades específicas. Tal desafio é enfrentado no presente artigo ao propor, de forma inédita, uma abordagem autonômica de detecção de defeitos capaz de se auto-configurar, ajustando de forma dinâmica o período de monitoramento e *timeout* de detecção em função de métricas de qualidade de serviço definidas pelo usuário. Para realizar tal modelagem, nos valemos do arcabouço de teoria de controle realimentado (*feedback control*) comumente utilizado em sistemas industriais de automação [Ogata 1995].

A abordagem de detecção autonômica proposta foi completamente implementada e avaliada, através de simulação, em função de métricas de QoS como taxa de falsas suspeitas, tempo de detecção, disponibilidade, intervalo entre falsas suspeitas e duração da falsas suspeitas. Tais métricas nos permitiram avaliar quão rápidas e precisas são as detecções realizadas em diferentes cenários de carga nos ambientes computacionais. Mesmo sem haver trabalhos correlatos para uma comparação direta de desempenho, as avaliações consideraram a comparação de um detector adaptativo existente na literatura e sua versão autonômica encapsulada por nossa abordagem. O detector adaptativo comparado foi configurado (de forma estática) com períodos de monitoramento diferentes. As avaliações mostraram que o desempenho de nossa abordagem autonômica nunca é inferior ao do detector adaptativo para cada um dos períodos de monitoria com os quais o mesmo foi configurado.

O restante deste artigo está estruturado da seguinte forma. A seção 2 discute trabalhos relacionados e o contextualização teórica. A seção 3 apresenta o detector autonômico proposto. A seção 4 avalia o desempenho do detector e, por fim, a seção

5 apresenta considerações finais e propostas para trabalhos futuros.

## 2. Trabalhos Relacionados e Contexto Teórico

Mecanismos de tolerância a falhas para sistemas distribuídos garantem serviço disponível mesmo na presença de falhas[1] [Avizienis et al. 2004]. O projeto de mecanismos de tolerância a falhas considera hipóteses sobre o comportamento dos componentes do sistema no caso de falha. O modelo mais comumente utilizado é o modelo de falha por parada silenciosa (falhas por *crash*). Nesse modelo, o processo pára de funcionar, deixando de responder a qualquer tipo de requisição. Mesmo quando tal modelo não se verifica normalmente na prática, o mesmo pode ser concretizado através de estruturas de mascaramento e hierarquia de falhas [Cristian 1991]. Detectar falhas por parada é fundamental para o funcionamento de diversos protocolos e algoritmos básicos para a implementação de sistemas confiáveis. Por exemplo, em um esquema de replicação passiva, o defeito de uma réplica primária precisa ser prontamente detectada para que uma das réplicas secundárias assuma o papel da réplica faltosa com o mínimo de impacto para a aplicação distribuída [Avizienis et al. 2004].

Em geral, a detecção de falhas por parada considera que um processo (dito monitorado), envia periodicamente, de forma espontânea ou sob demanda, mensagens que indicam seu estado para um processo remoto (dito monitor). O componente monitor determina um intervalo de tempo limite (*timeout*) para a chegada de tais mensagens. Se a mensagem não chega dentro do *timeout*, o componente monitor aponta a falha do componente monitorado. Esse modelo de detecção é dependente das restrições temporais consideradas sobre tempo de transmissão e processamento das mensagens de monitoramento. Nos sistemas distribuídos assíncronos, geralmente assumidos nos ambientes de interesse deste trabalho, os limites temporais para o processamento e transmissão das mensagens inexistem, o que torna certos problemas de tolerância a falhas impossíveis de serem resolvidos de forma determinística [Fischer et al. 1985]. Para responder a essa impossibilidade, em [Chandra and Toueg 1996] foi introduzido o conceito de detectores de defeitos não confiáveis voltados para sistemas assíncronos. Esses detectores são ditos não confiáveis, pois podem apontar como falhos processos corretos ou deixar de apontar a falha daqueles que efetivamente falharam. O trabalho de Chandra e Toueg (1996) demonstrou como detectores não confiáveis podem, encapsulando certo grau de sincronia, resolver problemas fundamentais em sistemas distribuídos assíncronos (e.g. consenso e difusão atômica). Apesar de este trabalho ser um resultado importante para a resolução de problemas fundamentais em sistemas distribuídos, a ausência de restrições temporais do modelo de computação assíncrona coloca ainda forte desafios práticos para implementação dos detectores de defeitos. Uma das dificuldades está em determinar valores adequados para o *timeout*. Um *timeout* excessivamente grande torna a detecção lenta e pode comprometer o tempo de resposta do sistema durante a falha de componentes. Por outro lado, um *timeout* muito pequeno pode degradar a confiabilidade do detector e, da mesma forma, prejudicar o desempenho do sistema uma vez que muitos algoritmos e protocolos, que confiam nas informações do detector, podem ser levados a realizar processamento e troca de mensagens adicionais por conta de uma falsa suspeita de falha. Assim, diversos trabalhos têm

[1] Adota-se falha, erro e defeito como tradução para *fault*, *error* e *failure*, respectivamente.

estudado o uso de preditores de atraso na implementação de detectores de defeitos ([Bertier et al. 2003], [Macêdo and Lima 2004], [Nunes and Jansch-Pôrto 2004], [Falai and Bondavalli 2005] etc.). O papel desses preditores é tentar sugerir, em tempo de execução, valores adequados para o *timeout* de modo a tornar a detecção mais rápida com o mínimo possível de impacto para a confiabilidade do detector. No entanto, tais trabalhos não consideram o ajuste dinâmico do período de monitoramento, outro fator importante para o desempenho do detector.

Apesar do avanço propiciado pelos detectores de Chandra e Toueg, a especificação de tais detectores referem-se a propriedades difíceis de serem avaliadas na prática (por exemplo, em algum momento a falha de um componente será detectada). Nesse sentido, em [Chen et al. 2002] foram definidas métricas de QoS que têm sido usadas para avaliar a velocidade e a confiabilidade de diversas implementações de detectores de defeitos. Com isso, o trabalho do projetista é definir um período de monitoramento e usar um preditor de atraso de mensagens que consiga entregar um serviço de detecção com um nível de QoS adequado aos requisitos das aplicações. Quando o comportamento do ambiente não muda, ou quando se pode aproximar tal comportamento a partir de uma distribuição de probabilidade, é possível, em tempo de projeto (*off-line*), realizar adequadamente a configuração do detector de modo que o mesmo apresente um nível de QoS satisfatório. Por exemplo, em [Chen et al. 2002] é apresentado um procedimento para realizar a configuração *off-line* do detector de defeitos. Em [Mills et al. 2004] e [Xiong et al. 2006] realiza-se a configuração do detector em tempo de execução. Entretanto, esses trabalhos assumem que o comportamento do ambiente não muda e não demonstram como a configuração do detector pode ser realizada dinamicamente a partir de métricas de QoS que definam a rapidez e precisão do detector, como tempo de detecção, duração da falsa suspeita e intervalo entre falsas suspeitas.

Até onde sabemos, este é o primeiro esforço de pesquisa a propor um detector de defeitos capaz de se auto-configurar em tempo de execução, ajustando, de forma combinada, o período de monitoramento e *timeout* de modo a se ajustar a QoS de detecção especificada. A idéia de um detector autonômico utilizando engenharia de controle foi proposta por nós em [de Sá and Macêdo 2009]. Contudo, o trabalho aqui apresentado trata de mecanismos completamente distintos, e muito mais eficazes, dos que os apresentados nesse texto anterior. Nós também propusemos uma outra abordagem complementamente diferente para modelagem do comportamento dinâmico do ambiente computacional, mas com desempenho similar ao detector apresentado neste artigo, a qual pode ser encontrada em [de Sá and Macêdo 2010].

## 3. A Proposta de Detecção Autonômica

### 3.1. Modelo do Sistema e Definições

Considera-se um sistema distribuído constituído por um conjunto finito de $n$ processos $\Pi = \{p_1, p_2, ..., p_n\}$, interconectados através de canais de comunicação não confiáveis. Nestes canais, mensagens podem ser perdidas, atrasadas ou corrompidas. Se uma mensagem $m$, enviada por um processo $p_i$ chega ao processo $p_j$ corrompida, a mesma é descartada. Processos do sistema podem falhar por *crash*. Um serviço de detecção distribuída de defeitos é implementado através do estilo de monitoramento *Pull*[Felber 1998]. Este estilo de monitoramento, considera que a cada $\delta$ unidades de

tempo um processo monitor ($p_i$) envia uma mensagem, dita *"are you alive?"* (*aya*), para um processo monitorado ($p_j$). Ao receber uma mensagem *aya*, $p_j$ deve responder com uma mensagem, dita *"I am alive!"* ou *heartbeat* (*hb*), para $p_i$. As mensagens *aya* e *hb* trocadas entre dois processos $p_i$ e $p_j$ são seqüencialmente assinaladas a cada intervalo de monitoramento. Sendo assim, $aya_k$ e $hb_k$ denotam, respectivamente, a k-ésima mensagem *"are you alive?"* enviada e a correspondente k-ésima mensagem *heartbeat* recebida. $s_k$ e $r_k$ denotam, respectivamente, os instantes de envio e recebimento das mensagens *aya* e *hb* por um processo monitor, de acordo com seu relógio local. Não são assumidos relógios sincronizados e as taxas de desvio dos relógios em relação ao tempo real são valores muitíssimo menores que os tempos de transmissão das mensagens, portanto, são fatores desprezados nos cálculos dos tempos. A cada mensagem $aya_k$ enviada, $p_i$ determina um intervalo de tempo ($rto_k$) para a chegada do *heartbeat* ($hb_k$) oriundo de $p_j$. Se o *heartbeat* não chega dentro do intervalo de tempo esperado, $p_i$ coloca $p_j$ em suas lista de suspeitos. Caso $p_i$ receba um *heartbeat* com *timestamp* superior ao *timestamp* do último *heartbeat* recebido, $p_i$ retira $p_j$ de sua lista de suspeitos.

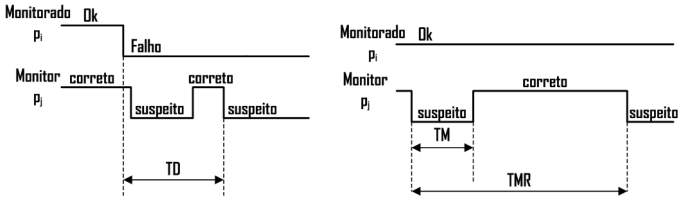


**Figure 1. Métricas de QoS:(a) $TD$; (b) $TM$ e $TMR$**

Para a configuração e avaliação do desempenho do serviço de detecção, considera-se as métricas de QoS propostas por [Chen et al. 2002]. Dentre as quais, destacam-se: *Tempo de detecção* (*Detection Time*); *Intervalo entre Falsas Suspeitas* (*Mistake Recurrence Time*); e *Duração da Falsa Suspeita* (*Mistake Duration*). O *Tempo de Detecção* ($TD$) é uma métrica relacionada ao tempo de resposta do detector e diz respeito ao intervalo de tempo entre o instante em que o processo monitorado falhou e o instante no qual o processo monitor passou a suspeitar definitivamente de tal falha [ver figura 1(a)]. O *Intervalo Entre Falsas Suspeitas* ($TMR$) é definido pelo intervalo de tempo observado entre erros consecutivos do detector [ver figura 1(b)]. Essa métrica é uma medida para a confiabilidade do detector e, em termos das métricas tradicionalmente usadas como medidas de confiabilidade [Avizienis et al. 2004], pode ser associada ao $MTBF$ (*Mean Time Between Failures*). A *Duração da Falsa Suspeita* ($TM$) diz respeito ao tempo necessário para que o detector corrija uma falsa suspeita de falha [ver figura 1(b)]. Esta métrica pode ser relacionada ao $MTTR$ (*Mean Time To Repair*)[2]. A partir de TM e TMR é possível estimar a disponibilidade ($AV$) do serviço de detecção por: $AV = (TMR - TM)/TMR$. As demandas das aplicações em termos de QoS de detecção são especificadas através de $TD^U$, $TM^U$ e $TMR^L$, os quais representam, respectivamente, o tempo máximo de detecção, a duração máxima de uma falsa sus-

[2]Da mesma forma que as métricas $TM$ e $TMR$, conforme originalmente proposto por [Chen et al. 2002], as associações ($TM$, $MTTR$) e ($TMR$, $MTBF$) são observadas (e calculadas) considerando um par de processos (monitor e monitorado).

peita e o intervalo mínimo entre falsas suspeitas. A notação $TD^U$, $TM^U$ e $TMR^L$ é utilizada em [Chen et al. 2002], onde $U$ e $L$ representam, respectivamente, *upper* e *lower bounds*. Neste artigo seguiremos este padrão de notação.

### 3.2. Processo de Configuração Automática

O processo de configuração automática considera a implementação de um gestor autonômico (ou controlador), dito $\mathcal{GA}$, o qual observa o comportamento dos *heartbeats* recebidos pelo módulo básico do serviço de detecção de defeitos (planta ou elemento gerenciado[3]) e, então, define, a partir dos requisitos de QoS ($TD^U$, $TM^U$ e $TMR^L$) e consumo de recursos ($RC^U$), o período de monitoramento ($\delta$) e o *timeout* ($rto$), a serem usados por um processo $p_i$ no monitoramento de um processo $p_j$ (ver figura 2). Para tanto, $\mathcal{GA}$ executa três atividades básicas: (i) Sensoriamento do Serviço de Detecção e do Ambiente Computacional; (ii) Regulação do *Timeout*; e (iii) Regulação do Período de Monitoramento.

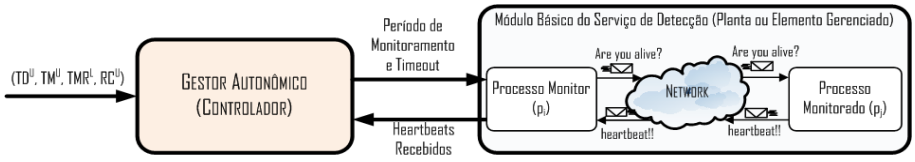


**Figure 2. Detector Autonômico**

#### 3.2.1. Sensoriamento

A atividade de sensoriamento consiste em estimar os atrasos do ambiente computacional e o desempenho do detector em termos de QoS de detecção e consumo de recursos. Nesse sentido as estratégias utilizadas são descritas a seguir.

**Sensoriamento dos atrasos do ambiente computacional.** A cada *heartbeat* recebido, é possível mensurar o atraso de ida-e-volta ($rtt$) de uma mensagem de monitoramento por $rtt_k = r_k - s_k$, em que $s_k$ e $r_k$ são, respectivamente, os instantes de envio de um *"are you alive"* e recebimento de um *heartbeat*. Perdas de mensagens dificultam a interação entre processos do sistema e impactam diretamente na percepção de um processo monitor $p_i$, a respeito do estado de um processo monitorado $p_j$. Todavia, mensurar o atraso de ida-e-volta ($rtt$) a partir da diferença entre $r_k$ e $s_k$ não permite que o gestor autonômico perceba o impacto provocado por perdas de mensagens na interação entre um par de processos. Isto porque o recebimento de um *heartbeat* por $p_i$, em $r_k$, indica apenas que $p_j$ esteve funcionando corretamente até o instante $r_k - rtt_k/2$ (aproximadamente). Sendo assim, mesmo quando $p_i$ recebe uma mensagem de *heartbeat*, o mesmo não tem qualquer garantia de que $p_j$ ainda esteja funcionando e quanto maior o intervalo de tempo decorrido desde o recebimento do último *heartbeat* maior é a incerteza de $p_i$ a respeito do estado de $p_j$.

Em um dado instante $t$, a duração da incerteza ($uti$) de $p_i$ a respeito do estado de $p_j$ pode ser mensurada por $uti(t) = t - (r_u - rtt_u/2)$. Em que $r_u$ e $rtt_u$ representam, respectivamente, o instante de chegada de último *heartbeat* ($hb_u$) e o

[3]Os termos planta e controlador, usados na área de engenharia de controle (ver [Ogata 1995]), se referem, respectivamente, ao sistema a ser controlado (e.g. um motor, uma caldeira etc.) e ao dispositivo que contém o algoritmo a ser utilizado no controle da planta. Estes conceitos são similares aos adotados na área de sistemas autonômicos[Huebscher and McCann 2008], podendo-se associar gestor autonômico e elemento gerenciado a controlador e planta, respectivamente.

atraso de ida-e-volta observado entre o envio de $aya_u$ e recebimento de $hb_u$. Se $p_i$ calcula a duração da incerteza a cada instante $s_k$, então: $uti_k = s_k - (r_u - rtt_u/2)$. Mais ainda, o atraso na interação ($d$) entre os processos $p_i$ e $p_j$ pode ser obtido por: $d_k = |uti_k - \delta_{k-1}|$. Se $p_j$ não falha e mensagens de monitoramento não são perdidas, então $d_k = rtt_k/2$, senão $d$ crescerá proporcionalmente a $\delta$ vezes o número de heartbeats não recebidos por $p_i$.

A magnitude do atraso de interação ($d$) pode ser usada como um indicativo para a utilização dos recursos do ambiente computacional. Sendo assim, é necessário que o gestor autonômico ($\mathcal{GA}$) não apenas descubra o valor mínimo para $d$, dito $d^L$, mas também observe a sua máxima variação, dita $j$. Os valores de $d^L$ e $j$ podem ser estimados assumindo o menor valor de $d$ e sua maior variação durante a execução do detector, respectivamente. Entretanto, se as características do ambiente computacional mudam, é possível que os valores de $d^L$ e $j$ observados por $\mathcal{GA}$ também mudem. Com isso, considera-se um fator de esquecimento $f$ na estimativa de tais valores. Deste modo, $\mathcal{GA}$ observa $d^L$ e $j$ usando o seguinte algoritmo:

- defina $d_0^L = rtt_0/2$ e $j_0 = 0$;
- se $d_{k-1}^L > d_k$ então $d_k^L = d_k$, senão faça $d_k^L = f * d_{k-1}^L + (1 - f) * d_k$;
- em seguida, calcule $var_k = |d_k - d_k^L|$; e se $j_{k-1} < var_k$, então $j_k = var_k$, senão faça $j_k = f * j_{k-1} + (1 - f) * var_k$;

Definir $f$ arbitrariamente pode levar $\mathcal{GA}$ a realizar ajustes inadequados em $\delta$, por conta da velocidade na variação de $d^L$ e $j$. Sendo assim, $f$ foi definido a partir do tempo máximo de detecção $TD^U$ e do atraso de interação mínimo observado $d^L$ por: $f = max[0, (TD^U - d^L)/TD^U]$. Se $TD^U \approx d^L$, então $f$ tende a 0 (esquecimento máximo), assim $\mathcal{GA}$ não memoriza os valores de $d^L$ e de $j$ e os mesmos variarão a cada observação. No outro extremo, se $TD^U >> d^L$, então $f$ tende a 1 (esquecimento nulo), significando que a atualização dos valores de $d^L$ e $j$ não é relevante e os mesmos serão atualizados de forma mais conservadora[4].

Variações espúrias do atraso podem levar a ajustes desnecessários no período de monitoramento[5]. Deste modo, após encontrar $d^L$ e $j$, $\mathcal{GA}$ realiza uma filtragem em $d$ de modo a eliminar tais variações. Assim, considere $d^F$ a versão filtrada de $d$, com $d_0^F = rtt_0$, então calcule: $d_k^F = f * d_{k-1}^F + (1 - f) * d_k$. A versão filtrada do atraso considerando $j$ é dada por: $d_k^J = d_k^F + j_k$.

**Sensoriamento da QoS de Detecção.** O sensoriamento da QoS de detecção consiste em estimar $TD$, $TM$ e $TMR$. A estimativa do tempo de detecção (TD) pode ser realizada considerando duas hipóteses distintas: (i) hipótese otimista – o processo monitorado ($p_j$) falha imediatamente antes de enviar um *heartbeat* $hb_k$ (i.e. $p_i$ suspeita de $p_j$ em $s_k + rto_k$); (ii) hipótese pessimista – $p_j$ falha imediatamente depois de enviar um *heartbeat* $hb_k$ ($p_i$ suspeita $p_j$ em $s_{k+1} + rto_{k+1}$). O gestor autonômico ($\mathcal{GA}$), em $p_i$, estima $TD$ assumindo a hipótese (ii). Assim, $\mathcal{GA}$ pressupõe uma falha de $p_j$ em $tcrash_k = r_{k-1} - rtt_{k-1}/2$ e $p_i$ suspeitará de $p_j$ em $tsuspect_k =$

[4]Se $TD^U \approx d^L$, então $\mathcal{GA}$ não ajustará $\delta$, por outro lado, se $TD^U >> d^L$, então $\mathcal{GA}$ ajustará $\delta$ de modo a atender a relação de compromisso entre consumo de recursos e tempo de detecção.

[5]O atraso fim-a-fim depende não apenas do $\delta$, mas também do uso dos recursos de processamento e comunicação pelas demais aplicações, o qual pode variar de forma brusca e aleatória. Daí a necessidade do filtro para evitar ajustes desnecessários causados por distúrbios momentâneos.

$s_k + rto_k$. Deste modo, pode-se estimar: $TD_k = tsuspect_k - tcrash_k$. Além disso, a cada instante de envio $s$, $\mathcal{GA}$ calcula: $TM_k = \sum_{i=1}^{k} sd_i/nf_k$ e $TMR_k = s_k/nf_k$. Em que $nf_k$ e $sd_k$ são, respectivamente, o número de falsas suspeitas cometidas e a duração da falsa suspeita em um dado instante $k$, com: a) $sd_k = 0$ e $nf_k = nf_{k-1}$, quando a informação de $p_i$ a respeito do estado de $p_j$ não muda ou muda de correto para suspeito; e b) $sd_k = r_k - (s_u + rto_u)$ e $nf_k = nf_{k-1} + 1$, quando a informação de $p_i$ a respeito de $p_j$ muda de suspeito para correto. A variável $s_u$ representa o instante de envio do último $aya_u$, para o qual um respectivo $hb_u$ não foi recebido dentro do timeout de detecção $rto_u$.

### 3.2.2. Regulação do *Timeout*

Para a detecção de defeitos, é interessante que o ajuste de *timeout* use preditores de atrasos que: (a) possuam convergência rápida, de modo assimilar mais rapidamente as variações na magnitude dos atrasos; (b) sejam precisos, de modo a não prejudicar o tempo de detecção; (c) não realizem subestimativas do atraso, de modo a permitir que o detector evite falsas suspeitas. Sendo assim, o gestor autonômico determina um valor para o *timeout* usando uma das estratégias disponíveis na literatura, de modo a atender aos requisitos (a) e (b). Em seguida, estima, com base na confiabilidade do detector, uma margem de segurança ($\alpha$) de modo a minimizar a quantidade de falsas suspeitas cometidas, com um menor prejuízo para o tempo de detecção e atendendo ao requisito (c). Assim, sendo $AV^L$ e $AV_k$, respectivamente, a disponibilidade mínima requerida e a disponibilidade observada para o detector no instante $k$, então defina $\alpha_0 = 0$ e $rto_0 = TD^U/2$ e a cada instante $s_k$:

- $AV^L = (TMR^L - TM^U)/TMR^L$ e $AV_k = (TMR_k - TM_k)/TMR_k$;
- obtenha $error_k = AV^L - AV_k$ e, então, calcule $\alpha_k = \alpha_{k-1} + \delta_{k-1} * error_k$;
- verifique se $\alpha_k < 0$ e, em caso afirmativo, faça $\alpha_k = 0$. Em seguida, utilize uma abordagem de ajuste de *timeout* existente na literatura para sugerir um *timeout* $rto_k^C$ e, então, obtenha o *timeout* de detecção $rto_k = rto_k^C + \alpha_k$;

O algoritmo acima implementa uma estratégia de controle integral, a qual segue o método de integração de Riemann (ver [Lima 2006]) através do somatório dos retângulos de área $\delta_{k-1} * error_k$. Qualquer estratégia de ajuste de *timeout* pode ser utilizada para sugerir $rto^C$.

### 3.2.3. Regulação do Período

O projeto da estratégia de regulação do período considera quatro passos: (i) caracterização do consumo de recursos do ambiente; (ii) definição do problema de controle; (iii) definição do modelo da planta; e (iv) projeto e sintonia do controlador. Estes passos culminam no algoritmo de regulação de período descrito no final desta seção.

**(i) Caracterização do Consumo de Recursos do Ambiente.** Para realizar uma estimativa do consumo de recursos (ou utilização) do ambiente computacional em um dado instante, $\mathcal{GA}$ observa o comportamento dos atrasos de ida-e-volta das mensagens de monitoramento. Uma mensagem de monitoramento possui tamanho fixo e precisa de $rtt^L = 2 * d^L$ unidades de tempo para ser processada pelo ambiente (i.e. envio de *aya* e recebimento de um *hb*). Se é observado um atraso de $rtt^J = 2 * d^J$ no processamento de uma mensagem de monitoramento, então, pelo menos existiam $M = rtt^J/rtt^L$ mensagens sendo processadas. Um processo $p_i$ gera apenas uma

mensagem de monitoramento a cada intervalo $\delta$, então pelos menos $Q = M - 1$ [ou $Q = (rtt^J - rtt^L)/rtt^L$] mensagens não foram gatilhadas por $p_i$ no período $\delta$. Em um intervalo $\delta$, o ambiente é capaz de processar ao menos $M^U = \delta/(rtt^L)$ mensagens. Sendo $RC$ uma medida para o consumo de recursos, o mesmo é definido por $RC = Q/M^U$, ou para uma freqüência de monitoramento $\lambda = 1/\delta$: $RC = (rtt^J - rtt^L) * \lambda$.

**(ii) Problema de Controle.** O problema de controle é regular $\lambda$ de modo a obter um menor $TD$, usando uma fração dos recursos disponíveis no ambiente a cada instante. Assim, seja $RC^U \in [0, 1)$ a quantidade máxima de recursos que pode ser consumida. Então, $RC_k$ é comparado a $RC^U$ e a diferença $error_k = RC^U - RC_k$ é calculada[6]. Quando $error_k > 0$, a utilização está abaixo do limite especificado, logo $\lambda$ é incrementado para permitir uma detecção mais rápida. Se $error_k < 0$, a utilização está acima do limite desejado, logo $\lambda$ é decrementado.

**(iii) Modelo Matemático da Planta.** Do ponto de vista de $\mathcal{GA}$, o módulo básico de detecção de defeitos representa a planta[3], ver figura 2. O modelo matemático da planta pode ser escrito em termos de equações diferenciais por $\partial RC/\partial\lambda = rtt^J - rtt^L$. Para o projeto do controlador, é interessante que tal modelo seja representado usando *funções de transferência*[7] descritas através da transformada $\mathcal{Z}$[Ogata 1995]. Para tanto, inicialmente, pressupõe-se que a diferença $rtt^J - rtt^L$ é constante, o que será contornado mais adiante, uma vez que os parâmetros do controlador variarão em função de tal diferença. Sendo assim, a variação de $RC$ em função da variação de $\lambda$ é uma constante. Deste modo, a função de transferência da planta pode ser descrita como uma função *step*[Ogata 1995] com amplitude igual a $rtt^J - rtt^L$. Uma vez que o efeito da escolha de $\lambda$ no instante $k - 1$ só será percebida em $RC$ no instante $k$, a função de transferência pode ser descrita por: $P(z) = (rtt^J - rtt^L)/(z - 1)$.

**(iv) Projeto e Sintonia do Controlador.** O gestor autonômico utiliza um controlador $P$[Hellerstein et al. 2004], com função de transferência $C(z)$, no qual a ação de controle (variação de $\lambda$) é proporcional ao desvio entre o valor desejado ($RC^U$) e o valor obtido ($RC$) para o consumo de recursos, ou seja $\partial\lambda = Kp * (RC^U - RC_k)$, ou $C(z) = Kp$. A sintonia do controlador diz respeito a encontrar o valor de $Kp$ de modo a atender aos seguintes requisitos[Hellerstein et al. 2004]: estabilidade; precisão; tempo de convergência (*settling time*); e máximo sobre-sinal (*overshoot*)[8]. Uma vez que, o comportamento do ambiente muda, os requisitos de desempenho do controlador não podem ser garantidos. Além disso, o projeto do controlador considera a escolha de intervalos fixos de tempo, dito período de amostragem ($\tau$), nos quais o controlador observa o valor do consumo de recursos e então atua sobre a freqüência de monitoramento de falhas. Tais períodos de amostragem também não podem ser garantidos uma vez que a observação de $RC$ em um dado instante depende do tempo de ida-e-volta de uma mensagem de monitoramento. Mais ainda,

[6]Uma referência mais intuitiva para o regulador de período seria usar $TD^U$. Entretanto, como $TD^U$ refere-se ao pior desempenho esperado, utiliza-se o mesmo como limitador para $\delta$ e passa-se a controlar o consumo de recursos para administrar o conflito entre $TD$ e o consumo de recursos.

[7]i.e. um modelo matemático para a relação entre entrada e saída de um sistema[Ogata 1995].

[8]Um sistema é dito estável se para uma entrada limitada produz uma saída limitada e é dito preciso se consegue minimizar o erro entre a resposta desejada e a obtida (Steady-state error). O tempo de convergência ($K_s$) representa o tempo que o sistema leva para atingir o estado desejado. O máximo sobre-sinal ($M_p$) é a diferença máxima entre saída desejada e a obtida [Hellerstein et al. 2004].

a transformada $\mathcal{Z}$, somente é válida se $\tau$ é fixo, portanto, qualquer análise utilizando funções de transferência em $\mathcal{Z}$ também não serão válidas. Sendo assim, pressupõe-se uma configuração inicial do controlador para atender a cada um dos requisitos de desempenho do controle e em seguinda utiliza-se uma lei de adaptação para o ganho $Kp$ de modo a atender ao menos o requisito de estabilidade. Com isso, para a configuração inicial do controlador, observa-se a função de transferência em malha fechada definida por[Ogata 1995]: $F_r(z) = C(z) * P(z)/[1 + C(z) * P(z)]$. A estabilidade é garantida se $1 + C(z)P(z) = 0$, isto é: $z - 1 + Kp * (rtt^J - rtt^L) = 0$. Usando a técnica de localização dos pólos em malha fechada[Hellerstein et al. 2004] e definindo um máximo sobre-sinal $M_p = 1 - RC^U$, é possível calcular: $Kp_k = 1$, se $rtt_k^J = rtt^L$, ou $K_p = RC^U/(rtt_k^J - rtt_k^L)$, se $rtt_k^J > rtt^L$.

**Algoritmo de Regulação de Período.** Finalmente, o algoritmo de regulação do período é descrito a seguir. A cada instante $s_k$:

- calcule $Kp_k$, obtenha $error_k = RC^U - RC_k$ e calcule $\lambda_k = \lambda_{k-1} + Kp * error_k$;
- se $\lambda_k < \lambda^L$, então $\lambda_k = \lambda^L$, ou se $\lambda_k > \lambda^U$, então $\lambda_k = \lambda^U$; faça $\delta_k = 1/\lambda_k$;

Em que $\lambda^L$ e $\lambda^U$ são as freqüências de monitoramento mínima e máxima, as quais são calculadas por: $\lambda^L = 1/max[d^L, (TD^U - d^L)]$ e $\lambda^U = 1/d^L$.

## 4. Avaliação de Desempenho

**Descrição do Ambiente de Simulação.** Os experimentos foram realizados em um ambiente simulado com o auxílio do pacote *Matlab/Simulink/TrueTime 1.5*[Henriksson and Cervin 2003]. Tal ambiente possui três computadores, ditos $c_1$, $c_2$ e $c_3$, conectados através de uma rede *Switched Ethernet* com taxa nominal de transferência de $10Mbps$, *buffer* de $1MB$ e em caso de *buffer overflow* mensagens são descatadas. O processo em $c_1$ monitora defeitos do processo em $c_2$. Em $c_3$, um processo é utilizado para gerar rajadas aleatórias de trafego na rede. Para tanto, tal processo é ativado periodica e continuamente. A cada ativação, tal processo escolhe um valor aleatório $v \in [0, 1]$. O valor $v$ é comparado a um fator de consumo de banda $BW \in [0, 1]$ e se $v < BW$, então $c_3$ envia uma mensagem para $c_1$. Dentro de uma janela de observação de $1s$, o total de mensagens geradas por $c_3$ representa uma ocupação de banda de $\approx BW * 100\%$. A cada janela de tempo de $1s$, $BW$ é incrementado em $0,1$ e retorna a zero quando $BW > 0,9$. O algoritmo executado em $c_3$ permite que os detectores sejam avaliados sob diferentes condições de carga. Por fim, todas as mensagens possuem tamanho fixo de 1518 bits e as simulações consideram um tempo simulado de $40s$, o que representa $\approx 10^5$ mensagens de monitoramento.

**Métricas de Desempenho.** O desempenho dos detectores foram verificados considerando as métricas $TD$, $TM$, $TMR$ e $AV$. Além disso, considera-se a taxa de falsas suspeitas ($SR$) cometidas, a qual é obtida dividindo o número de falsas suspeitas ($nf$) pelo número total de predições realizadas pelo detector.

**Configuração dos Detectores.** O desempenho do detector autonômico ($\mathcal{AFD}$) proposto é verificado comparando-o com o detector adaptativo ($\mathcal{AD}$) de [Jacobson 1988]. Nas comparações, consideram-se duas configurações para $\mathcal{AD}$, as quais usam período de monitoramento fixo e equivalentes a $1ms$ e $5ms$, respectivamente. $\mathcal{AFD}$ foi configurado da seguinte forma: $RC^U = 0,5$; e $TD^U = 50ms$, $TM^U = 1ms$ e $TMR^L = 10000ms$ (ver seção 3.2). Além disso, $\mathcal{AFD}$ encapsula o algoritmo de [Jacobson 1988] na regulação do *timeout*(ver seção 3.2.2).

**Resultados Obtidos.** As figuras de 3 a 7 apresentam o desempenho dos detectores de defeitos em termos das métricas de desempenho consideradas. Em termos do tempo de detecção, figura 3, o detector adaptativo com $\delta = 1ms$ ($\mathcal{AD}\{1ms\}$) apresenta baixos tempos de detecção para baixas condições de carga, mas eleva drasticamente $TD$ quando considerado cenários com cargas maiores que $\approx 70\%$. Observe que $\mathcal{AD}\{1ms\}$ consome $\approx 30,36\%$ da banda, significando que, nos intervalos $[7s, 9s]$, $[17s, 19s]$, $[27s, 29s]$ e $[37s, 39s]$, a utilização é maior que 100% (o que justifica os picos nos valores de $TD$ observados em tais intervalos para tal detector). O detector adaptativo com $\delta = 5ms$ ($\mathcal{AD}\{5ms\}$), por sua vez, possui tempos de detecção muito próximo ao valor do seu período de monitoramento. O detector autonômico ($\mathcal{AFD}$) possui tempo de detecção que variam entre $1,5ms$ e $7,0ms$ dependendo da carga, apresentando em média $TD < 4,5ms$. Em termos da duração das falsas

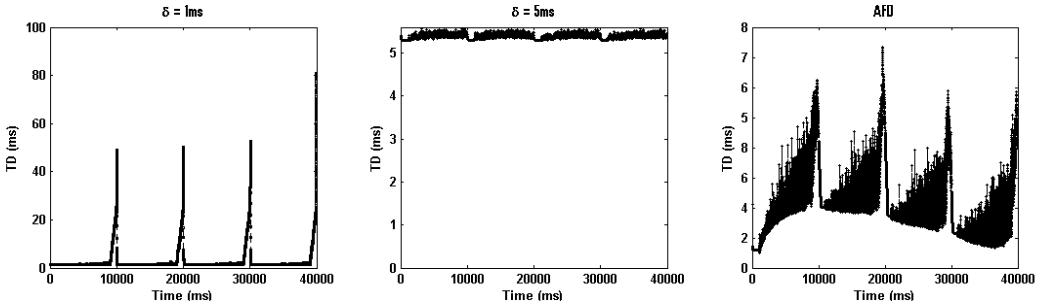


**Figure 3. Desempenho em Termos do Tempo de Detecção**

suspeitas, figura 4, com exceção $\mathcal{AD}\{1ms\}$, o qual apresentou $TM$ na ordem de $10^{-1}ms$, todos os demais apresentaram $TM$ na ordem de $10^{-2}ms$. Nesta métrica, $\mathcal{AFD}$ obteve o melhor desempenho. Em termos da taxa de falsas suspeitas, figura

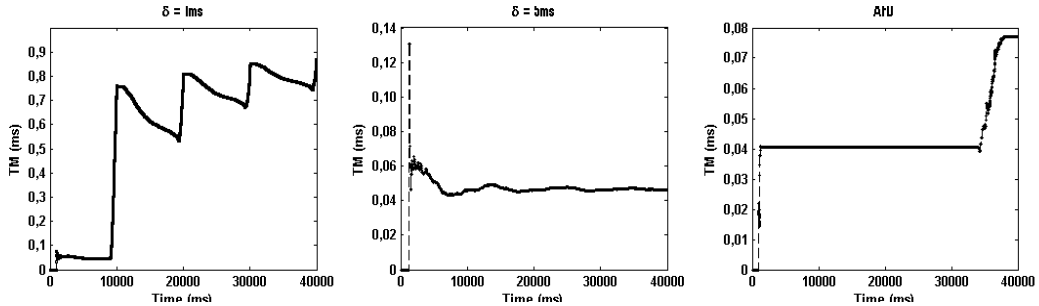


**Figure 4. Desempenho em Termos da Duração da Falsa Suspeita**

5, $\mathcal{AFD}$ apresenta as menores taxas de falsas suspeitas. As diferentes configurações de $\mathcal{AD}$ apresentam, na maioria dos casos, taxas de falsas suspeitas superiores $0,08$, o que significa um desempenho médio 20 vezes pior que aquele obtido por $\mathcal{AFD}$. Em termos do intervalo entre falsas suspeitas, figura 6, os valores de $TMR$ estabilizam em $6,85ms$ e $55,32ms$ para $\mathcal{AD}\{1ms\}$ e $\mathcal{AD}\{5ms\}$, respectivamente. Para $\mathcal{AFD}$, por outro lado, $TMR$ cresce linearmente. Sendo que, após uma rajada de falsas suspeitas o mesmo estabiliza em $142,8ms$. As rampas apresentadas no gráfico do detector autonômico indicam que o mesmo leva mais tempo para cometer falsas suspeitas. Por fim, a figura 7 apresenta o comportamento dos detectores em termos de $AV$. A disponibilidade de $\mathcal{AD}\{1ms\}$ decai com o tempo, estabilizando

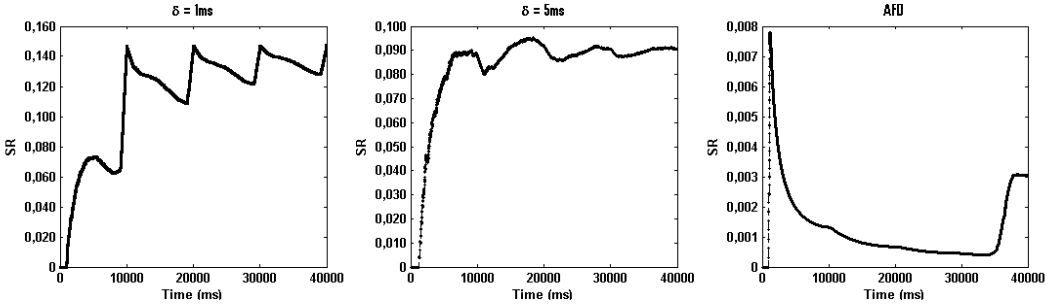


**Figure 5. Desempenho em Termos da Taxa de Falsas Suspeitas**

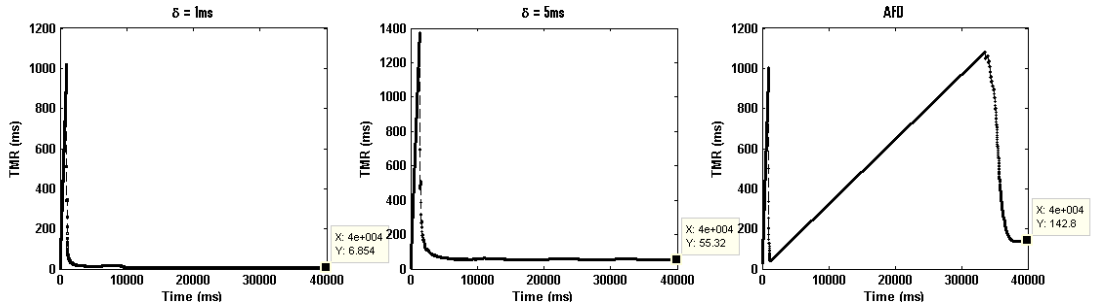


**Figure 6. Desempenho em Termos do Intervalo entre Falsas Suspeitas**

em algo em torno de $0,86$. Para $\mathcal{AD}\{5ms\}$, a disponibilidade finaliza o experimento oscilando em algo em torno de $0,9991$. $\mathcal{AFD}$ possui desempenho médio em termos desta métrica um pouco melhor, ficando em algo em torno de $0,9998$. Nas figuras

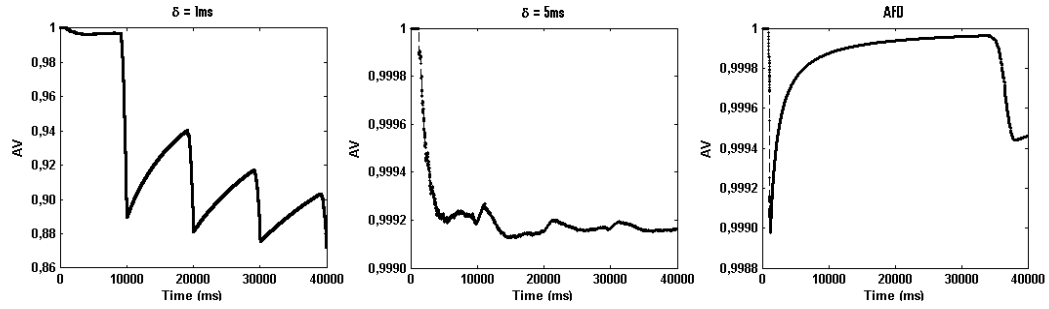


**Figure 7. Desempenho em Termos da Disponibilidade do Serviço de Detecção**

4, 5, 6 e 7, observa-se uma queda de QoS para $\mathcal{AFD}$, em $35s$. Isto porque, neste ponto, $\mathcal{AFD}$ está com $\delta$ mínimo e decrescendo (verifique a figura 3), chegando ao ponto de interferência máxima (menor $\delta$), até o ponto em que o período estabiliza e começa a aumentar como indicam os gráficos no fim da curva.

## 5. Considerações Finais

As propostas de detecção de defeitos em sistemas distribuídos existentes não suportam a configuração automática do detector a partir de métricas de QoS. Entretanto, quando as características do ambiente computacional são desconhecidas e podem mudar, a auto-configuração se torna uma habilidade fundamental para atender à relação de compromisso entre os requisitos de tempo de resposta e disponibilidade. Implementar a habilidade de auto-configuração requer a modelagem do comportamento dinâmico do sistema distribuído, o que se apresenta como um desafio, uma

vez que, na maioria dos ambientes abertos, é muito difícil caracterizar tal comportamento através de distribuições de probabilidades específicas. Neste sentido, este artigo apresentou, de forma inédita, o projeto e implementação de um detector baseado em teoria de controle e com habilidade de configurar, de forma dinâmica, o período de monitoramento e o *timeout* de detecção em função de restrições de QoS definidas pelo usuário. O uso da teoria de controle, neste contexto, permitiu suplantar as imprecisões decorrentes da modelagem do comportamento dinâmico.

As avaliações realizadas, através de simulação, verificaram o desempenho do detector em função da taxa de falsas suspeitas, tempo de detecção, disponibilidade, intervalo entre falsas suspeitas e duração das falsas suspeitas. Tais métricas nos permitiram avaliar quão rápido e preciso são as detecções realizadas em diferentes cenários de carga nos ambientes computacionais. Mesmo sem haver trabalhos correlatos para uma comparação direta de desempenho, as avaliações consideraram comparações com um detector adaptativo disponível na literatura e configurado (de forma estática) com períodos de monitoramento diferentes. Além da capacidade do ajuste automático do período de monitoramento para cumprir a QoS acordada, as avaliações mostraram que o desempenho de nossa abordagem autonômica nunca é inferior ao do detector avaliado. Os experimentos consideram um detector autonômico construído de modo a encapsular um detector adaptativo com o algoritmo de predição de [Jacobson 1988], entretanto, qualquer detector adaptativo pode ser encapsulado em nossa abordagem. Outros experimentos foram realizados considerando o detector adaptativo de [Bertier et al. 2003] e os resultados foram similares[9]. As avaliações foram realizadas para redes locais, típicas de aplicações como *cluster computing*. Para trabalhos futuros pretendemos estender nossa avaliação para cenários WAN e aplicar os mecanismos avaliados num gestor de aplicações autonômicas em desenvolvimento no LaSiD [Andrade and Macêdo 2009].

## Referências

Andrade, S. S. and Macêdo, R. J. A. (2009). A non-intrusive component-based approach for deploying unanticipated self-management behaviour. In *Proc. of IEEE ICSE 2009 Workshop Software Engineering for Adaptive and Self-Managing Systems*.

Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33.

Bertier, M., Marin, O., and Sens, P. (2003). Performance analysis of hierarchical failure detector. In *Proc. Of Int. Conf. On DSN*, pages 635–644, USA. IEEE.

Chandra, T. D. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal Of The ACM*, 43(2):225–267.

Chen, W., Toueg, S., and Aguilera, M. K. (2002). On the quality of service of failure detectores. *IEEE Trans. On Computer*, 51(2):561–580.

Cristian, F. (1991). Understanding fault-tolerant distributed systems. *Commun. ACM*, 34(2):56–78.

[9]Tais experimentos estão disponíveis em: http://www.lasid.ufba.br/publicacoes/papers.xml.

de Sá, A. and Macêdo, R. J. A. (2009). Uma proposta de detector de defeitos autonômico usando engenharia de controle. In *X Workshop de Testes e Tolerância a Falhas (WTF 2009)*.

de Sá, A. S. and Macêdo, R. J. A. (2010). Qos self-configuring failure detectors for distributed systems. In *The 10th IFIP international conference on Distributed Applications and Interoperable Systems (DAIS 2010)*, Amsterdan, Netherlands.

Falai, L. and Bondavalli, A. (2005). Experimental evaluation of the qos failure detectors on wide area network. In *Proc. of Int. Conf. on DSN*, pages 624–633. IEEE CS.

Felber, P. (1998). *The Corba Object Group Service : A Service Approach to Object Groups in CORBA*. PhD thesis, Département D'Informatique, École Polytechnique Fédérale De Lausanne.

Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382.

Hellerstein, J. L., Diao, Y., Parekh, S., and Tilbury, D. M. (2004). *Feedback Control of Computing Systems*. Wiley-Interscience, Canada.

Henriksson, D. and Cervin, A. (2003). Truetime 1.13 - reference manual. Tech. Report TFRT-7605-SE, Dep. Of Automatic Control, Lund Institute Of Technology.

Huebscher, M. C. and McCann, J. A. (2008). A survey of autonomic computing—degrees, models, and applications. *ACM Comput. Surv.*, 40(3):1–28.

Jacobson, V. (1988). Congestion avoidance and control. *ACM CC Review; Proc. Of The Sigcomm '88 Symp. In Stanford, Ca, August, 1988*, 18, 4:314–329.

Lima, E. L. (2006). *Análise Real: Funções de Uma Variavél*, volume 1. Instituto de Matemática Pura e Aplicada, Brasil, Rio de Janeiro.

Macêdo, R. J. A. and Lima, F. R. L. (2004). Improving the quality of service of failure detectors with snmp and artificial neural networks. In *Anais do 22o. Simpósio Brasileiro de Redes de Computadores, pp.583–586, ISBN: 85-88442-81-7*.

Mills, K., Rose, S., Quirolgico, S., Britton, M., and Tan, C. (2004). An autonomic failure-detection algorithm. In *WOSP '04: Proc. of the 4th int. workshop on Software and Performance*, pages 79–83, New York, USA. ACM.

Nunes, R. C. and Jansch-Pôrto, I. (2004). Qos of timeout-based self-tuned failure detectors: the effects of the communication delay predictor and the safety margin. In *International Conf. On DSN*, pages 753–761. IEEE Computer Society.

Ogata, K. (1995). *Discrete-Time Control Systems*. Prentice-Hall, Upper Saddle River, NJ 07458, USA, 2nd edition.

Satzger, B., Pietzowski, A., Trumler, W., and Ungerer, T. (2008). A lazy monitoring approach for heartbeat-style failure detectors. In *3rd Int. Conf. on Availability, Reliability and Security, 2008 (ARES2008)*, pages 404–409.

Xiong, N., Yang, Y., Chen, J., and He, Y. (2006). On the quality of service of failure detectors based on control theory. In *20th Int. Conf. on Advanced Information Networking and Applications*, volume 1.