ALIRIO SANTOS DE SÁ

MECANISMOS AUTONÔMICOS DE TOLERÂNCIA A FALHAS PARA SISTEMAS DISTRIBUÍDOS

Tese apresentada ao Programa Multiinstitucional de Pós-Graduação em Ciência da Computação da Universidade Federal da Bahia, Universidade Estadual de Feira de Santana e Universidade Salvador, como requisito parcial para obtenção do grau de Doutor em Ciência da Computação.

Orientador: Dr. Raimundo José de Araújo Macêdo

Salvador
2011

Ficha catalográfica elaborada pela Biblioteca Reitor Macedo Costa,
Instituto de Matemática

Sá, Alirio Santos de
Mecanismos autonômicos de tolerância a falhas para sistemas distribuídos / Alirio Santos de Sá. – Salvador, 2011.

185p. : il.

Orientador: Prof. Dr. Raimundo José de Araújo Macêdo.
Tese (doutorado) – Universidade Federal da Bahia, Instituto de Matemática, 2011.

1. Tolerância a Falhas. 2. Sistemas Distribuídos.
I. Macedo, Raimundo José de Araújo. II. Universidade
Federal da Bahia. Instituto de Matemática. III Título.

CDD 20.ed. 003.83

TERMO DE APROVAÇÃO

ALIRIO SANTOS DE SÁ

MECANISMOS AUTONÔMICOS DE TOLERÂNCIA A FALHAS PARA SISTEMAS DISTRIBUÍDOS

Esta tese foi julgada adequada à obtenção do título de Doutor em Ciência da Computação e aprovada em sua forma final pelo Programa Multiinstitucional de Pós-Graduação em Ciência da Computação da UFBA-UEFS-UNIFACS.

Salvador, 04 de novembro de 2011

Professor e orientador Raimundo José de Araújo Macêdo, Ph.D.
Universidade Federal da Bahia

Professor Joni da Silva Fraga, Docteur
Universidade Federal de Santa Catarina

Professor Renato Fontoura de Gusmão Cerqueira, Doutor
Pontifícia Universidade Católica do Rio de Janeiro

Professor Joberto Sérgio Barbosa Martins, Docteur
Universidade Salvador

Professor Flávio Morais de Assis Silva, Dr.-Ing.
Universidade Federal da Bahia

Professor Alfredo Goldman vel Lejbman, Ph.D.
Universidade de São Paulo

RESUMO

As facilidades de processamento e comunicação oriundas das novas tecnologias têm promovido o surgimento de uma nova classe de ambientes distribuídos. Estes ambientes são caracterizados pela dinamicidade em suas composições, no provisionamento de seus recursos e nas características e requisitos de suas aplicações. Isto traz novos desafios à confiabilidade, a qual é um atributo essencial à grande maioria dos sistemas distribuídos modernos. Um destes desafios está na incapacidade dos mecanismos tradicionais de tolerância a falhas em atender aos requisitos de desempenho, ao mesmo tempo em que suportam a confiabilidade. Isto ocorre porque o projeto destes mecanismos requer um conhecimento prévio das características dos ambientes e de suas aplicações, para que possam oferecer configurações adequadas ao atendimento dos requisitos especificados – isto representa um problema, uma vez que, nos ambientes distribuídos modernos, estas informações mudam dinamicamente. Neste contexto, nem mesmo os mecanismos adaptativos convencionais de tolerância a falhas obtêm sucesso, pois realizam a sua configuração dinamicamente, mas confiam em comportamentos e requisitos definidos em tempo de projeto. Para enfrentar este desafio, esta Tese introduz os mecanismos autonômicos de tolerância a falhas, baseados em teoria de controle e capazes de se auto-configurar face às mudanças dinâmicas nas características do ambiente ou nos requisitos de suas aplicações. Com o intuito de demonstrar a viabilidade destes mecanismos, foram implementados e avaliados, como estudo de caso, detectores autonômicos de defeitos e protocolos autonômicos de comunicação em grupo, dois mecanismos básicos à construção de muitos sistemas distribuídos confiáveis. Estes mecanismos autonômicos de detecção e de comunicação em grupo são os primeiros da literatura a suportar a auto-configuração em tempo de execução, baseada em requisitos de qualidade de serviço definidos pelos usuários. Tais mecanismos foram avaliados, usando simulações, em condições de carga variadas e falhas. Mesmo sem trabalhos relacionados, para uma comparação direta de desempenho, os mecanismos autonômicos propostos foram comparados com mecanismos tradicionais de tolerância a falhas existentes na literatura. Estes mecanismos tradicionais usaram diferentes configurações definidas por parâmetros manualmente fixados. Os experimentos realizados demonstram que os mecanismos autonômicos propostos possuem, na maioria dos casos, desempenho superior que às diferentes configurações dos mecanismos tradicionais considerados, principalmente quando variações nas características da carga, mudanças nos requisitos ou reconfigurações dinâmicas no ambiente são consideradas.

Palavras-chave: Tolerância a falhas, sistemas distribuídos, computação autonômica, computação baseada em controle realimentado.

ABSTRACT

The processing and communication capabilities of new computing technologies have given rise to a new class of distributed environments. These environments are dynamic because of the characteristics of their composition, resource provisioning and also because application requirements can change over time. Thus, they pose new challenges to dependability which is required by such modern distributed systems. One of these challenges is the inefficiency of traditional fault-tolerance mechanisms to address application performance requirements while at the same time addressing dependability issues in such a highly dynamic environments. As the design of these mechanisms requires a priori information about the characteristics of the environments and related applications, a problem is posed once such information is unknown and changes over time. In such contexts, even traditional adaptive fault-tolerant mechanisms cannot succeed because their adaptation approaches rely on requirements and environment behaviors defined during the design. To address such a challenge, we introduce autonomic fault-tolerant mechanisms, based on feedback control theories, which are capable of self-configuring according to changes in the computing environment and in user-defined requirements. In order to demonstrate the feasibility of these mechanisms, we implemented and evaluated, as a proof of concept, autonomic failure detectors and autonomic group communication protocols – two basic mechanisms for dependable distributed systems. The autonomic mechanisms proposed are the first mechanisms capable of self-configuring at runtime based on user-defined quality of service requirements. Our mechanism was evaluated using simulations considering varying workloads and component faults. Even without related works for a direct performance comparison, we compared our approach with traditional fault-tolerant mechanisms which use different configurations defined from manually fixed parameters. The results show that in the most cases our autonomic approach performed better than the manually configured fault-tolerant mechanisms, mainly when varying workloads, changes in the requirements and dynamic reconfigurations in the environment are considered.

Keywords: Fault-tolerance, distributed systems, autonomic computing, feedback control computing.

SUMÁRIO

Apêndices

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

Capítulo 1

Este capítulo aborda a motivação principal para esta Tese, bem como sua justificativa, seus objetivos e suas contribuições.

INTRODUÇÃO

1.1 CONTEXTO GERAL

A sociedade moderna, em suas diversas atividades (e.g. negócios, entretenimento, pesquisas etc.), é dependente do funcionamento de inúmeras aplicações *online*, as quais são concebidas considerando serviços e ambientes computacionais distribuídos. Por conta desta dependência, assegurar a confiabilidade e a disponibilidade dessas aplicações é um aspecto básico a ser perseguido no projeto e na gestão destes serviços e ambientes computacionais.

Para garantir o funcionamento correto e contínuo, mecanismos de tolerância a falhas são fundamentais. Estes mecanismos são responsáveis por dotar os ambientes e serviços distribuídos da redundância necessária para que os mesmos possam *mascarar* ou *compensar*Cristian (1991) a ocorrência de falhas. Entretanto, para cumprirem seus objetivos, tais mecanismos demandam custos computacionais adicionais, em termos de processamento e comunicação (Jalote, 1994). Assim, para não comprometerem o funcionamento normal dos sistemas, os mecanismos de tolerância a falhas devem ser adequadamente configurados, de modo a conciliar estes custos adicionais com a expectativa de carga das aplicações, com os requisitos de desempenho especificados e com o grau desejado de confiabilidade (Dai; Wang, 1992; Cristian, 1991; Veríssimo; Rodrigues, 2000).

Além disso, atividades de gerenciamento devem ser realizadas para assegurar a eficiência e eficácia destes mecanismos (Hegering; Abeck; Neumair, 1998). Estas atividades consistem no monitoramento contínuo dos sistemas e dos mecanismos de tolerância a falhas e, também, na realização de ajustes ou re-configuração de parâmetros, quando as condições do ambiente ou os requisitos definidos pelas aplicações divergem daqueles considerados durante o projeto.

Neste contexto, um dos grandes desafios para a implementação e gestão dos mecanismos de tolerância a falhas é suportar as nuances dos ambientes distribuídos abertos (e.g. a Internet), os quais são constituídos por um grande número de dispositivos heterogêneos, conectados a partir

de canais de comunicação com diferentes capacidades e sujeitos a condições de carga variadas. A conseqüência disto são ambientes caracterizados por incertezas em relação aos tempos de processamento e transmissão das mensagens.

Para lidar com essas incertezas, foram realizados diversos esforços de pesquisa relacionados, dentre outras coisas, à proposição de modelos de sincronia e mecanismos de tolerância a falhas adequados. As pesquisas relacionados aos modelos de sincronia se preocupavam com a prova de correção (i.e. *safety* e *liveness*) dos algoritmos e protocolos face às incertezas temporais dos ambientes considerados – ver, por exemplo, Dwork, Lynch e Stockmeyer (1988), Lamport e Lynch (1989), Chandra e Toueg (1996), Cristian e Fetzer (1999), Veríssimo, Casimiro e Fetzer (2000), Veríssimo e Casimiro (2002) etc. Em paralelo, muitas pesquisas relacionadas aos mecanismos de tolerância a falhas focavam na implementação de facilidades de adaptação – ver, por exemplo, Chockler, Huleihel e Dolev (1998), Macêdo (2000), Mills et al. (2004), Sivasubramanian et al. (2005), Sá e Macêdo (2006), Karmakar e Gupta (2007), Macêdo (2008a) etc.

Para oferecer um desempenho adequado, estas facilidades de adaptação permitem aos mecanismos de tolerância a falhas ajustarem dinamicamente alguns de seus parâmetros de configuração – usando requisitos definidos durante o projeto e alguma suposição sobre as características do ambiente (como distribuições de probabilidades, por exemplo) – ver, por exemplo, Chen, Toueg e Aguilera (2002), Bertier, Marin e Sens (2002), Falai e Bondavalli (2005), Rütti, Wojciechowski e Schiper (2006), Satzger et al. (2007) etc.

1.2 MOTIVAÇÃO E JUSTIFICATIVA

Na medida em que as novas facilidades de processamento e comunicação propiciam o surgimento de ambientes distribuídos com características dinâmicas, novos desafios se apresentam ao projeto e gerenciamento dos mecanismos de tolerância a falhas.

Esta dinamicidade é originada de aplicações com características e requisitos dinâmicos, além de plataformas que permitem não apenas a alocação, liberação e migração dinâmica de recursos virtualizados, mas também a definição de composições dinâmicas, em que os componentes são definidos em tempo de execução (a partir da entrada e saída espontânea de dispositivos e processos, por exemplo). Casos típicos desses ambientes distribuídos são encontrados, por exemplo, em plataformas de P2P (Peer-to-Peer) e de grades computacionais e em ambientes de computação em nuvens (Milojicic et al., 2002; Baker; Buyya; Laforenza, 2002; Rimal; Choi; Lumb, 2009).

Por um lado, esta dinamicidade é um benefício, pois permite que os recursos sejam ajustados, em tempo de execução, às necessidades das aplicações. Além disto, facilita a implementação de estratégias de reconfiguração dinâmica, necessárias em caso de falhas, por exemplo.

Por outro lado, novas componentes de incerteza são somadas às características dos ambientes distribuídos tradicionais, pois os recursos computacionais disponíveis e os requisitos e

características das aplicações variam ao longo do tempo. Este aspecto dificulta o projeto e a gestão da maioria dos mecanismos de tolerância a falhas, os quais usam estimativas *off-line* a respeito dos recursos e dos requisitos e características das aplicações para definirem adequadamente seus parâmetros operacionais.

Além disso, os mecanismos adaptativos de tolerância a falhas existentes na literatura não estão preparados para lidar com esta dinamicidade. Isto porque não consideram, na adaptação de seus parâmetros, a dinâmica do ambiente e dos requisitos das aplicações. Mais ainda, muitas das suposições usadas por estes mecanismos adaptativos, sobre o comportamento do ambiente, não são apropriadas. Em muitos ambientes, por exemplo, tais comportamentos são difíceis de serem caracterizados a partir de distribuições de probabilidades específicas.

1.3 OBJETIVOS

Esta Tese tem como desafio a implementação de estratégias que permitam que os mecanismos tradicionais de tolerância a falhas possam suportar a confiabilidade, adaptando dinamicamente seus parâmetros de configuração a partir das mudanças observadas nas características do ambientes distribuídos e nos requisitos de qualidade de serviço das aplicações – um aspecto negleciado pelas abordagens existentes na literatura, mas que é importante para assegurar o desempenho dos mecanismos de tolerância, quando ambientes distribuídos modernos são considerados.

1.3.1 Objetivos específicos

A partir do desafio proposto, consideram-se os seguintes objetivos específicos:

i) *Auto-gestão* – atender aos desafios impostos pelos ambientes distribuídos modernos, a partir da proposição de mecanismos de tolerância a falhas com habilidade de auto-gerenciamento;

ii) *Suporte aos requisitos dinâmicos de qualidade de serviço* – apresentar e avaliar estratégias para implementação desses mecanismos, considerando o atendimento de requisitos dinamicamente definidos pelas aplicações ou usuários;

iii) *Suporte ao legado de soluções existentes* – permitir que os mecanismos propostos sejam implementados aproveitando as facilidades dos mecanismos de tolerância a falhas existentes, sem comprometer a correção (i.e. *safety* e *liveness*) dos mesmos.

iv) *Reuso* – conceber estratégias com baixo acoplamento em relação ao ambiente de execução, podendo atuar em diferentes modelos de sistemas distribuídos.

v) *Modelagem do comportamento dinâmico* – explorar técnicas existentes na literatura que permitam a análise e síntese de modelos para lidar com o comportamento dinâmico do ambiente distribuído, sem comprometer o reuso da solução.

Em um contexto mais específico, os objetivos deste trabalho se enquadram nas linhas de pesquisa desenvolvidas no *Laboratório de Sistemas Distribuídos* (LaSiD[1], Departamento de Ciência da Computação, Universidade Federal da Bahia).

Na última década, os grupos de pesquisa do LaSiD têm trabalhado no desenvolvimento de modelos computacionais, algoritmos e protocolos, arquiteturas e ferramentas de software e mecanismos básicos direcionados ao atendimento dos desafios dos ambientes distribuídos com características híbridas e dinâmicas.

Assim, os mecanismos auto-gerenciáveis de tolerância a falhas, propostos e introduzidos nesta Tese, visam a complementação dos esforços do LaSiD na busca por soluções para o enfrentamento dos desafios relacionadas aos ambientes distribuídos modernos.

1.4 CONTRIBUIÇÕES

A principal contribuição desta Tese é a proposição, implementação e avaliação de mecanismos auto-gerenciáveis (ou autonômicos) de tolerância a falhas, baseados em teoria de controle, os quais são capazes de configurar dinamicamente seus parâmetros operacionais (e.g. freqüência de monitoramento de falhas, *timeouts* de detecção, freqüências de checkpoints etc.), observando as mudanças nas características dinâmicas do ambiente distribuído e considerando requisitos de qualidade de serviço dinamicamente definidos pelos usuários ou aplicações (como, por exemplo, sobrecarga no uso dos recursos, tempo de resposta, confiabilidade etc.). Assim, esta Tese:

a) Introduz os mecanismos autonômicos de tolerância a falhas, determinando uma metodologia para a implementação do laço de gestão.

b) Demonstra como os mecanismos tradicionais de tolerância a falhas podem ser especializados para apresentarem a habilidade de auto-gestão.

c) Demonstra como a teoria de controle de sistemas dinâmicos pode ser usada na construção dos mecanismos autonômicos de tolerância a falhas.

d) Propõe e implementa, como estudo de caso, a facilidade de gestão autonômica em dois mecanismos básicos para construção de sistemas distribuídos confiáveis: detectores de defeitos e protocolos confiáveis de comunicação em grupo.

e) Avalia, usando simulações, o desempenho dos mecanismos autonômicos de tolerância a falhas propostos, considerando condições de carga que variam com o tempo e requisitos de mudanças nos requisitos de qualidade de serviço.

f) Mostra como o compromisso entre custo computacional e desempenho dos mecanismos de tolerância a falhas pode ser atendido.

[1] ver http://www.lasid.ufba.br/

g) Propõe métricas operacionais, que podem ser usadas para avaliar, em tempo de execução, o desempenho dos mecanismos de tolerância a falhas, sem a necessidade do uso de distribuições de probabilidades específicas, por exemplo.

Nas subseções a seguir é apresentada uma visão geral da abordagem autonômica proposta, considerando: aspectos conceituais e de implementação; uma visão geral dos mecanismos autonômicos implementados como estudo de caso; e os principais artigos, no contexto desta Tese, publicados em workshops, conferências e simpósios da área.

1.4.1 Visão conceitual da contribuição

A idéia central da proposta desta Tese é levar o problema da configuração dos parâmetros dos mecanismos de tolerância a falhas para um nível de abstração mais próximo das aplicações – seguindo a abordagem da computação autonômica (Horn, 2001).

Isto implica no estabelecimento de um novo modelo de alocação de responsabilidades. Neste modelo, as aplicações ou os usuários devem definir, em tempo de execução, as suas demandas ou restrições em termos de requisitos de qualidade de serviço (e.g. tempo de resposta, confiabilidade, percentual de uso de recursos etc.). Os mecanismos autonômicos de tolerância a falhas, por sua vez, devem observar as variações dinâmicas nas características do ambiente e ajustar seus parâmetros operacionais de modo a atender os requisitos dinamicamente definidos.

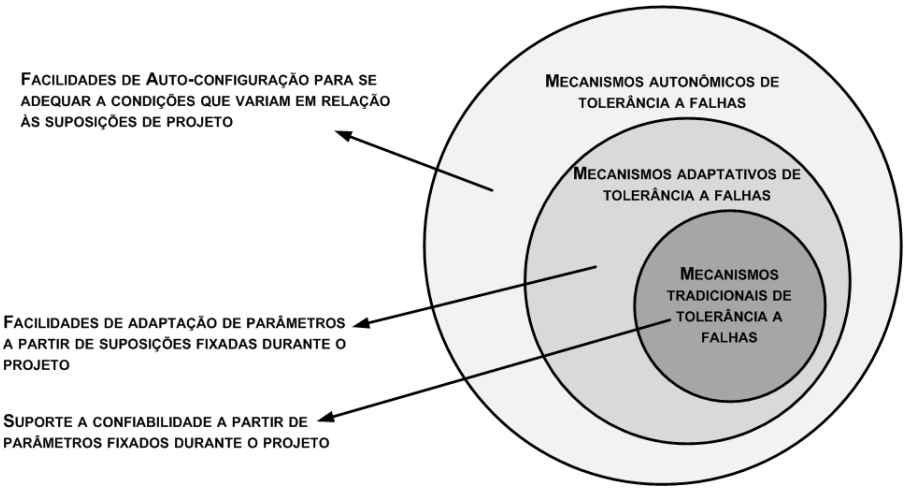


Figura 1.1. Visão conceitual, desta Tese, dos mecanismos autonômicos de tolerância a falhas

Conceitualmente, os mecanismos autonômicos propostos podem ser vistos como uma estratégia que provê o suporte à confiabilidade, encapsulando os mecanismos tradicionais ou adaptativos de tolerância a falhas (ver Figura 1.1), isto é (Macêdo, 2008b):

(a) mecanismos tradicionais de tolerância a falhas se preocupam com o suporte a confiabilidade, garantindo as propriedades de correção (*safety* e *liveness*) de seu serviço, a partir de parâmetros de configuração fixados durante o projeto.

(b) mecanismos adaptativos de tolerância a falhas encapsulam (ou especializam) as facilidades dos mecanismos tradicionais para permitir a adaptação da configuração, considerando para tanto, suposições (ou hipóteses) fixadas durante o projeto – e.g. distribuições de probabilidade, arranjo dos componentes, disponibilidade dos recursos etc.

(c) mecanismos autonômicos de tolerância a falhas encapsulam (ou especializam) as facilidades providas pelos mecanismos adaptativos, de modo a permitir a auto-configuração quando as características do ambiente ou requisitos das aplicações variam em relação às suposições previamente definidas durante o projeto.

A implementação dos mecanismos autonômicos de tolerância a falhas não é uma tarefa simples, pois requer, por exemplo: modelos adequados para a representação do comportamento dinâmico do ambiente distribuído; a conciliação dinâmica de relações de compromissos associados aos requisitos de qualidade de serviços definidos etc. Neste sentido, um dos objetivos desta Tese é apresentar estratégias que permitam lidar com tais questões de implementação.

1.4.2 Visão autonômica da gestão de tolerância a falhas

A concepção da habilidade de auto-gerenciamento (i.e. auto-configuração) requer que os mecanismos de tolerância a falhas implementem um laço de gerenciamento autonômico (Horn, 2001), envolvendo monitoramento, planejamento e intervenções – ver Figura 1.2.

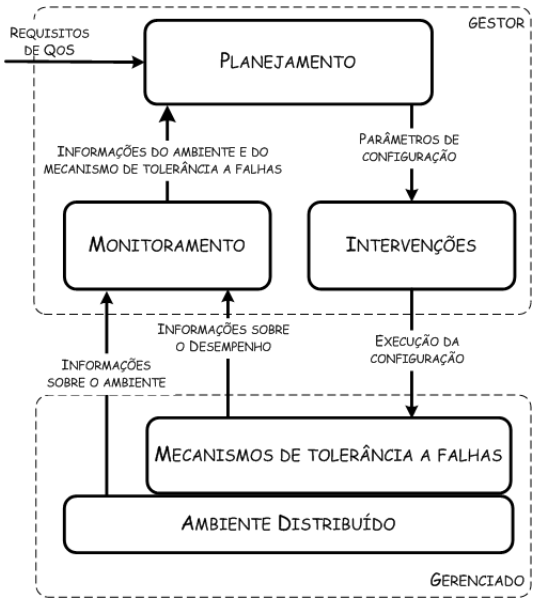


Figura 1.2. Laço de gestão implementado pelos mecanismos de tolerância a falhas propostos

O monitoramento envolve a coleta de informações associadas às características dinâmicas do ambiente e à qualidade de serviço entregue pelos mecanismos de tolerância a falhas. O planejamento envolve a definição de parâmetros operacionais adequados para levar o desempenho do mecanismo de tolerância a falhas a atender os requisitos de qualidade de serviço definidos,

considerando o estado atual do ambiente distribuído. As intervenções envolvem a implantação, nos mecanismos de tolerância a falhas, dos parâmetros operacionais definidos durante o planejamento.

Este laço de gestão se assemelha aos laços de controle implementados, no campo da engenharia, na automação e no controle de sistemas industriais (Ogata, 2009).

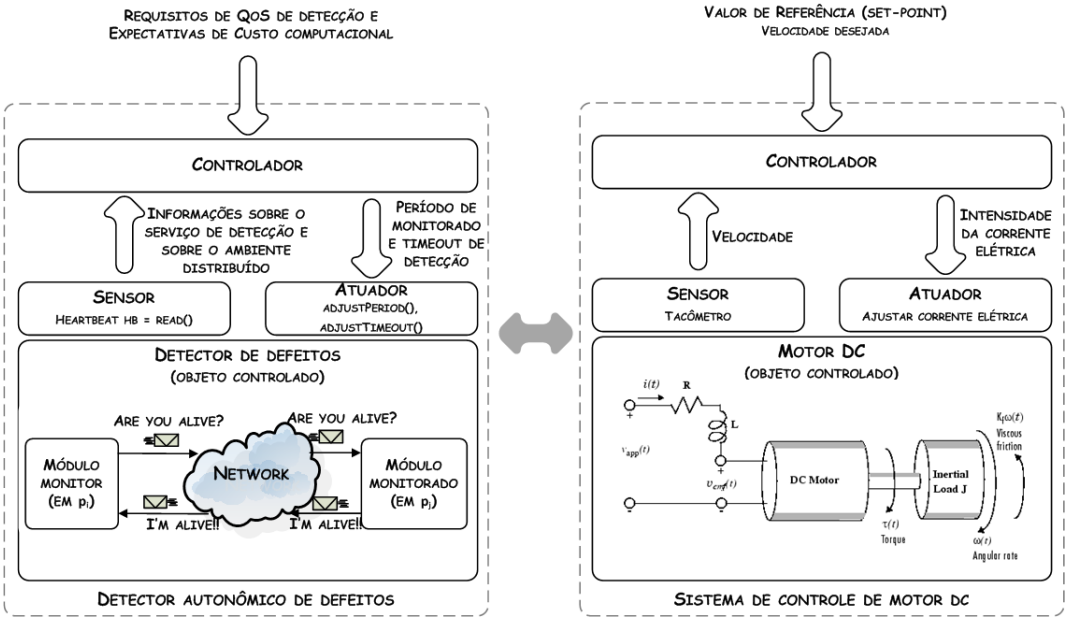


Figura 1.3. Comparativo entre detector autonômico e sistema de controle de motor

Como exemplo, pode-se fazer o paralelo entre um mecanismo autonômico de detecção de defeitos e um sistema de controle de motor DC (corrente contínua), ver Figura 1.3.

Nesse exemplo, um tacômetro (sensor) monitora a velocidade do motor (objeto controlado). Em seguida, um controlador verifica se a velocidade observada está de acordo com um valor de referência (*set-point*). Existindo desvios entre as velocidades, determina um novo valor de corrente elétrica (configuração) para o motor DC, de modo que o mesmo apresente a velocidade desejada. Então, um atuador ajusta a corrente elétrica do motor, usando o valor informado pelo controlador. No caso do detector, métodos de leitura (sensor) coletam informações sobre comportamento do ambiente e o desempenho do serviço de detecção (objeto controlado), usando mensagens de monitoramento, por exemplo. Em seguida, o controlador verifica se o desempenho observado está de acordo com os requisitos de qualidade de serviço de detecção e as expectativas de custo definidas pelo usuário (i.e. *set-points*). Existindo divergências, o controlador determina os novos parâmetros do detector (e.g. período de monitoramento de falhas e *timeout* de detecção), de modo que o detector apresente o desempenho desejado em termos de qualidade de serviço de detecção e custo computacional. Em seguida, o controlador ativa métodos de escrita (atuador) necessários para que os novos parâmetros de configuração sejam implantados.

Assim, na perspectiva do projeto, o laço de gestão dos mecanismos autonômicos de tole-

rância a falhas é projetado seguindo a mesma abordagem usada nos sistemas de controle, isto é: embutindo, nos mecanismos de tolerância a falhas existentes, controladores (ou gestores autonômicos) responsáveis pelo planejamento das configurações, além de métodos de sensoriamento e atuação que realizam a coleta das informações e a implantação dos novos parâmetros de configuração, respectivamente.

Por conta das técnicas de análise e síntese disponibilizadas pela teoria de controle de sistemas dinâmicos (Ogata, 1995; Hellerstein et al., 2004), a implementação dos mecanismos de tolerância a falhas propostos utilizam tal ferramental matemático para a modelagem de suas funções autonômicas.

1.4.3 Mecanismos autonômicos implementados

Como estudo de caso, a viabilidade da proposta foi avaliada a partir da implementação dos seguintes mecanismos: detectores autonômicos de detecção de defeitos e protocolos autonômicos de comunicação em grupo. Detectores de defeitos são elementos básicos na implementação da confiabilidade em sistemas distribuídos, seja para permitir a ativação de procedimentos de recuperação, seja para possibilitar o acionamento de mecanismos de reconfiguração na ocorrência de falhas. Protocolos de comunicação em grupo são fundamentais para implementação de facilidades de coordenação distribuída e sincronização de estados em esquemas de serviços replicados – um esquema comum em tolerância a falhas.

Estes mecanismos foram implementados considerando hipóteses de falhas por crash (parada) de componentes, uma vez que esta é a hipótese mais considerada na prática. No caso de outras hipóteses de falhas mais severas (e.g. falhas bizantinas), os modelos de falhas por crash podem ser considerados a partir de técnicas de mascaramento e hierarquia de falhas (Cristian, 1991) – o que garante a abrangência dos mecanismos autonômicos de tolerância a falhas propostos.

A detecção autonômica de defeitos proposta é uma inovação, pois é a primeira proposta a demonstrar como os parâmetros do detector (i.e. período de monitoramento e *timeout* de detecção) podem ser configurados em tempo de execução, considerando métricas de qualidade de serviço e mudanças nas características do ambiente. Além disto, na implementação deste mecanismo também foram introduzidos os conceitos de intervalo de incerteza e atraso de interação, como métricas operacionais para caracterizar a influência do comportamento do ambiente distribuído na interação entre processos do sistema. A métrica de disponibilidade de detecção também foi introduzida como uma métrica operacional para avaliação da confiabilidade do detector. Mais ainda, foi demonstrado como métricas tradicionais de qualidade de serviço usadas na detecção de defeitos (e.g. tempo de detecção, duração de falsas suspeitas e intervalos entre falsas suspeitas) podem ser estimadas em tempo de execução sem a necessidade do uso de distribuições de probabilidades específicas.

O protocolo autonômico de comunicação em grupo proposto também é uma inovação, pois

é o primeiro a demonstrar como a relação de compromisso entre custo (i.e. sobrecarga de mensagens) e latência de entrega (i.e. tempo de resposta) pode ser atendida, considerando requisitos definidos pelo usuário (e.g. percentual de consumo de recursos).

1.4.4 Publicações

Algumas das principais contribuições desta Tese, e outros resultados intermediários ou correlatos obtidos, foram publicados em workshops, simpósios e conferências da área. Estas publicações são listadas a seguir:

i) Macêdo, R. J. A.; Freitas, A. E. S.; Sá, A. S. de. A self-manageable group communication protocol for partially synchronous distributed systems. In: *Proceeding of Fifth Latin-Amarican Symposium on Dependable Computing*. São José dos Campos, SP, Brazil: IEEE Computer Society Press, 2011. (LADC'2011), p. 146–155. ISBN 978-0-7695-4320-8.

ii) Sá, A. S. de; Macêdo, R. J. A. QoS self-configuring failure detectors for distributed systems. In: *Proceedings of the 10th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS' 2010)*. Berlin, Heidelberg: Springer-Verlag, 2010. (Lecture Notes in Computer Science (LNCS), v. 6115), p. 126–140. Disponível em: <http://dx.doi.org/10.1007/978-3-642-13645-0>. Acesso em: October, 4, 2011.

iii) Sá, A. S. de; Macêdo, R. J. A. Detectores de defeitos autonômicos para sistemas distribuídos. In: *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Gramado, Brasil: SBRC / SBC, 2010. (SBRC'2010), p. 785–798. ISSN 2177-4978.

iv) Foletto, T.; Moraes, V.; Moreno, U.; Sá, A. S. de; Macêdo, R. J. A. Avaliação sobre a inclusão de estimadores baseados em filtragem de kalman em sistemas de controle via rede. In: *XVIII Congresso Brasileiro de Automática*. Bonito, MS, Brasil: SBA, 2010. (CBA'2010), p. 1–7. ISBN 978-8-5615-0702-2.

v) Sá, A. S. de; Macêdo, R. J. A. Uma proposta de detector autonômico de defeitos usando engenharia de controle. In: *Anáis X do Workshop de Testes e Tolerância a Falhas*. João Pessoa, PB, Brazil: SBC, 2009. (WTF'2009).

vi) Sá, A. S. de; Macêdo, R. J. A.; Moreno, U.; Santos, T. Um procedimento para avaliação de redes ethernet comutada baseada em uma métrica de qualidade de controle. In: *XVII Congresso Brasileiro de Automática*. Juiz de Fora, MG, Brasil: SBA, 2008. (CBA'2008), p. 1–6.

vii) Sá, A. S. de; Macêdo, R. J. A. Um framework para prototipagem e simulação de detectores de defeitos para sistemas de tempo real distribuídos de controle e supervisão. In: *VIII Workshop de Testes e Tolerância a Falhas*. Belém, PA, Brasil: SBRC/SBC, 2007. (WTF'2007), p. 43–56.

1.5 ESTRUTURA DA TESE

O restante desta Tese está organizada em duas partes. A primeira parte trata da revisão da literatura, abordando questões relacionadas aos sistemas distribuídos (Capítulo 2) e aos conceitos de teoria de controle (Capítulo 3). A segunda parte discute as contribuições desta Tese, apresentando os detectores autonômicos de defeitos (Capítulo 4), os protocolos autonômicos de comunicação de comunicação em grupo (Capítulo 5) e as considerações finais e perspectivas de trabalhos futuros (Capítulo 6).

Assim, o Capítulo 2 discute, principalmente, os modelos de sincronia, aspectos de confiabilidade, mecanismos de tolerância a falhas e gerenciamento de sistemas distribuídos (e gestão autonômica). O Capítulo 3 foca nas representações de sistemas discretos, projeto de sistemas de controle discretos e abordagens de controle adaptativo.

O Capítulo 4 contextualiza a proposta de detecção autonômica de defeitos com relação ao estado da arte e discute os aspectos de implementação, os experimentos realizados e os resultados obtidos. O Capítulo 5 contextualiza o protocolo autonômico de comunicação em grupo com relação ao estado da arte e discute os aspectos de implementação, os experimentos realizados e os resultados obtidos. Por fim, o Capítulo 6 faz um resumo geral das questões apresentadas na Tese, resume os benefícios e desafios da implementação dos mecanismos autonômicos de tolerância a falhas e, então, discute os trabalhos em andamento e as perspectivas de trabalhos futuros a partir da abordagem autonômica proposta.

PARTE I

FUNDAMENTAÇÃO TEÓRICA

Capítulo 2

Este Capítulo apresenta conceitos básicos relacionados aos sistemas distribuídos, discutindo, principalmente, modelos de sincronia, aspectos de confiabilidade, mecanismos de tolerância a falhas e gerenciamento de sistemas distribuídos.

FUNDAMENTOS DE SISTEMAS DISTRIBUÍDOS

2.1 INTODUÇÃO

Um sistema distribuído é caracterizado por um conjunto de unidades de processamento interconectadas através de canais de comunicação. Sobre as unidades de processamento, executam um conjunto de processos que se comunicam através de troca de mensagens e colaboram para atingir um objetivo comum (Coulouris; Dollimore; Kindberg, 2005). A Figura 2.1 apresenta um exemplo de sistema distribuído, em que aplicações executando em dispositivos com diferentes características em termos de processamento (e.g. servidores, computadores pessoais, smartphones etc.) interagem, usando troca de mensagens, através da Internet.

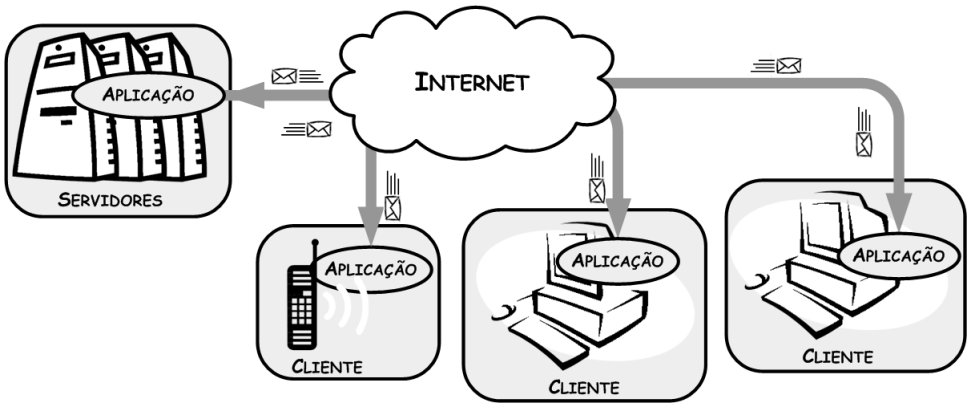


Figura 2.1. Exemplo de sistema distribuído

O que diferencia os sistemas distribuídos de outros sistemas com características similares[1] é o fato de cada unidade de processamento possuir um relógio local e independente dos demais, o qual é utilizado para registrar o momento de ocorrência dos eventos locais – isto é, estes sistemas

[1]Como em alguns casos de sistemas multiprocessados, por exemplo.

são fortemente caracterizados pela ausência de um relógio único e global para determinar a cadência das ações ou marcar os instantes de ocorrência de eventos (Veríssimo; Rodrigues, 2000).

A ausência de relógio global e de memória compartilhada torna os problemas em sistemas distribuídos mais difíceis de serem solucionados – principalmente quando é necessária a execução de ações coordenadas ou a obtenção de estado global em ambientes sem garantias temporais em termos de processamento e comunicação. Todavia, essa independência dos componentes na computação distribuída traz benefícios, principalmente quando aspectos como disponibilidade, confiabilidade e extensibilidade precisam ser considerados.

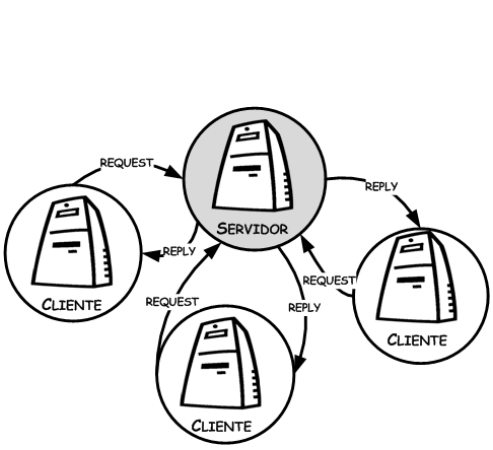
A concepção de sistemas distribuídos deve considerar diferentes perspectivas, as quais, quando combinadas, buscam atender aos requisitos demandados pelos usuários e suas aplicações. Fundamentalmente, essas perspectivas estão apoiadas em aspectos básicos que envolvem, entre outras coisas: os modelos de computabilidade; os modelos arquiteturais; e os mecanismos de suporte básico.

Os modelos de computabilidade permitem a separação entre problemas solúveis e insolúveis em ambientes distribuídos. Para tanto, estes modelos são construídos considerando semânticas relacionadas ao tempo, às falhas de componentes e aos mecanismos e infra-estruturas subjacentes de suporte ao sistema (Veríssimo; Rodrigues, 2000). As semânticas relacionadas ao tempo dizem respeito às especificações temporais dos componentes do sistema e de suas interações, ditando, por exemplo, o nível de sincronia das ações relacionadas ao processamento e à transmissão das mensagens. As semânticas de falhas determinam os comportamentos dos componentes ou dos protocolos em caso de falhas, além de especificar, por exemplo, a freqüência e os prováveis tipos de falhas considerados. As semânticas relacionadas às infra-estruturas subjacentes dizem respeito às facilidades disponibilizadas pelas abstrações suportadas pelos ambientes de execução subjacentes, determinando, por exemplo, aspectos ligados à previsibilidade, à consistência e à ordenação das operações (ou primitivas) disponíveis em sistemas operacionais, em *middlewares* ou em outros elementos subjacentes que suportem o desenvolvimento das aplicações.

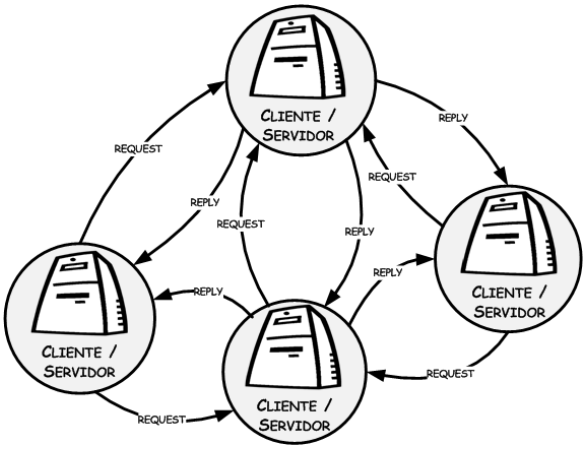
Os modelos arquiteturais são representados pelos estilos arquiteturas e pelas arquiteturas de sistema (Tanenbaum; Steen, 2007). Os estilos arquiteturais definem as formas de organização lógica e responsabilidades dos componentes de software (Garlan; Shaw, 1993) – como, por exemplo, arquiteturas em camadas, baseadas em eventos, centradas em dados ou ainda baseadas em objetos. No contexto dos sistemas distribuídos, a escolha do estilo arquitetural é realizada de modo a permitir que a aplicação distribuída exponha mais facilmente os requisitos não funcionais desejados – por exemplo, o estilo arquitetural baseado em eventos beneficia o baixo acoplamento e a interação anônima do tipo muitos-para-muitos. A arquitetura de sistema representa a instanciação do estilo arquitetural em uma distribuição física dos componentes no sistema (Tanenbaum; Steen, 2007). De um modo geral, as arquiteturas de sistemas distribuídos são classificadas em centralizadas (e.g. cliente/servidor), descentralizadas (e.g. P2P[2]) ou híbridas

[2]Ver Milojicic et al. (2002)

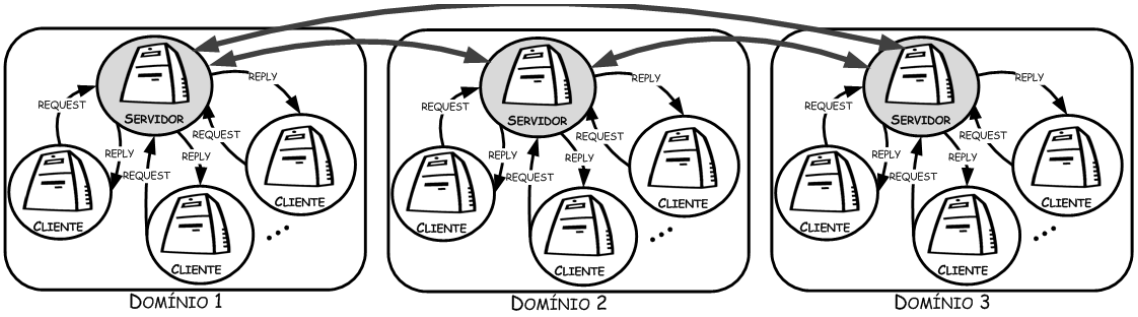
(e.g. federações de Data Centers[3]). A Figura 2.2 apresenta um exemplo de cada uma dessas arquiteturas de sistema. A escolha da arquitetura de sistema tem impacto direto na qualidade do serviço oferecida pelo sistema distribuído – ditando, por exemplo, questões relacionadas ao tempo de resposta, a confiabilidade e a segurança.



(a) Serviço centralizado



(b) Serviço distribuído



(c) Serviço distribuído entre domínios e centralizado intra-domínio

Figura 2.2. Exemplos de arquitetura de sistemas distribuídos

Os mecanismos de suporte básico pautam a implementação de facilidades ou protocolos que suportam as características básicas demandas aos sistemas distribuídos, e.g. confiabilidade, segurança, transparência, gerenciamento, adaptação etc. Um exemplo são os mecanismos de tolerância a falhas (e.g. detectores de defeitos, mecanismos de sincronização de estado etc.), os quais permitem que a infra-estrutura de execução atendam aos requisitos de confiabilidade demandados pelas aplicações. Outro exemplo são as facilidades de interação disponibilizadas por mecanismos de comunicação, como aqueles que implementam os protocolos de comunicação em grupo (Chockler; Keidar; Vitenberg, 2001; Défago; Schiper; Urbán, 2004). Os mecanismos de suporte compõem os *middlewares* para aplicações distribuídas, os quais podem suportar uma ou mais garantias de qualidade de serviço – consideradas nas semânticas especificadas nos modelos de computabilidade (e.g. restrições temporais, semânticas de falha, ordenação e atomicidade das operações etc.). Estes *middlewares* integram as infra-estruturas básicas de execução oferecidas às aplicações distribuídas, ver Figura 2.3.

[3] Ver Rochwerger et al. (2009)

Figura 2.3. Organização de sistema distribuído

O entendimento dos conceitos apresentados são relevantes para as discussões a serem realizadas nos próximos capítulos. Dentro do escopo deste trabalho, a gestão da confiabilidade é suportada por mecanismos básicos com habilidade de gerenciamento autonômico. Estes mecanismos são concebidos observando os modelos de computabilidade, os quais apóiam a definição das hipóteses apresentadas pelo modelo do sistema distribuído considerado – focando, particularmente, nos aspectos pertinentes às semânticas de tempo e à possibilidade de resolução dos problemas mediante às condições de falhas. A implementação da facilidade de gestão autonômica é definida a partir de um estilo arquitetural próprio, que habilita os mecanismos a exporem as propriedades inerentes à tal facilidade de gestão. Assim, as próximas seções deste capítulo se atêm a uma discussão mais detalhada destes conceitos em sistemas distribuídos, concentrando a argumentação em, respectivamente: (a) modelos de sincronia (i.e. modelos de computabilidade focados em semânticas de tempo); (b) aspectos de confiabilidade; (c) mecanismos básicos de tolerância a falhas; e (d) gestão autonômica.

2.2 MODELOS DE SINCRONIA

De acordo com a sincronia, modelos de sistemas distribuídos podem ser classificados entre dois extremos: modelos síncronos ou modelos assíncronos (Lamport; Lynch, 1989). Em um modelo de sistema completamente assíncrono os limites temporais são desconhecidos ou inexistentes. Assim, entre outras coisas, não se pode fazer considerações sobre os tempos de processamento e de viagens das mensagens. Em um sistema síncrono, por outro lado, os limites temporais são conhecidos a priori.

Uma vez que os modelos assíncronos são implementados livres de restrições temporais, existe um maior potencial de reutilização e adaptação da solução em ambientes com um alto grau de incerteza. Entretanto, se falhas de processos ou dos meios de comunicação forem considerados, a solução de certos problemas básicos em um modelo puramente assíncrono se torna uma tarefa difícil e sem solução determinística (Fischer; Lynch; Paterson, 1985).

Os modelos de sistemas síncronos são bastante convenientes para sistemas em que as restrições temporais são imperativas para o correto funcionamento do sistema. Entretanto, as de-

finições dos limites temporais podem ser bastante difíceis de serem obtidas na prática. Por exemplo, o tempo máximo de execução de um algoritmo depende da implementação da plataforma de hardware, do desempenho do processo de geração do código objeto do programa, das políticas e facilidades de comunicação usadas, dos defeitos em componentes, da carga computacional imposta pelas aplicações, do grau de dinamicidade e de heterogeneidade do ambiente etc. Além disso, falhas na especificação de limites temporais podem levar os sistemas a apresentarem defeitos durante a execução.

Modelos síncronos e assíncronos ocupam dois extremos em termos de sincronia e, de certo modo, a escolha entre um destes dois modelos pode não ser adequada para a concepção de muitos sistemas distribuídos típicos. Sistemas reais tipicamente possuem alguma hipótese temporal e, mesmo quando as restrições temporais são fortes, algumas componentes de tempo são incertas. Esse fato tem motivado o desenvolvimento de modelos que ocupem uma posição intermediária entre os modelos síncronos e assíncronos (Dwork; Lynch; Stockmeyer, 1988). A exemplo disso, estão os modelos assíncronos com tempo de Cristian e Fetzer (1999) e os assíncronos com detectores de defeitos de Chandra e Toueg (1996). Modelos como estes são ditos parcialmente síncronos, haja vista que possuem alguma informação de tempo, mas essa informação pode não ser exata. Os modelos parcialmente síncronos possuem particular importância, uma vez que consideram o nível de sincronia necessário para que certos problemas fundamentais em sistemas distribuídos, impossíveis de serem solucionados em sistemas puramente assíncronos, venham a ter solução – como, por exemplo, a solução para o problema do consenso distribuído apresentada em Dwork, Lynch e Stockmeyer (1988).

Os modelos parcialmente síncronos tradicionalmente são compostos por configurações homogêneas, enquanto que muitos ambientes distribuídos modernos são formados por composições heterogêneas, em que diferentes condições de sincronia se apresentam em diferentes componentes do sistema.

Nesse contexto, Veríssimo e Casimiro (2002) propõem o *TCB* (*Timely Computing Base*), o qual apresenta modelos de computabilidade e de arquitetura híbridos em termos de sincronia. No *TCB*, as componentes síncronas e assíncronas coexistem de modo a prover uma solução unificada para o problema da conciliação de incertezas temporais na especificação de sistemas de tempo real (i.e. sistemas síncronos). Um contexto de aplicação do modelo *TCB* pode ser encontrado em ambientes industriais de controle e manufatura, nos quais dispositivos de um sistema de controle sobre rede[4], com restrições temporais fortes, precisam coexistir com sistemas supervisórios com restrições temporais mais fracas. Nesses cenários as ações síncronas dos dispositivos de controle devem considerar a interferência de cargas de trabalho assíncronas dos sistemas supervisórios.

A proposta do TCB é generalizada por Veríssimo (2006), o qual propõe o modelo de *Wormholes*. Tal modelo pressupõe a existência de um modelo assíncrono equipado com componentes com propriedades especiais, dito *wormholes*, que garantem certas propriedades como

[4]NCS, *Networked control systems* – ver Yang (2006).

sincronia ou segurança, as quais permitem que problemas fundamentais de sistemas distribuídos assíncronos possam ser solucionados. Um contexto típico de aplicação do modelo de *wormholes* é aquele em que ambientes distribuídos baseados em grades computacionais (*Grid computing*), formadas por aglomerados de servidores (*clusters*) em diferentes *Data Centers*, interagem através da Internet – a exemplo da arquitetura de pesquisa na *Web* disponibilizada por empresas como o *Google*, ver Barroso, Dean e Hölzle (2003). Um problema existente no modelo *Wormholes* é o fato de o mesmo considerar composições estáticas em relação às mudanças da qualidade de serviço das componentes de sincronia, o que torna esse modelo limitado, quando se precisa considerar infra-estruturas modernas como aquelas que se apresentam em ambientes de computação em nuvens (*cloud computing*) ou P2P – para os quais, além das composições heterogêneas, aspectos dinâmicos em termos das infra-estruturas de execução subjacentes precisam ser considerados.

Nesse sentido, para lidar com os aspectos dinâmicos e híbridos dos sistemas distribuídos modernos, atendendo às demandas dos novos ambientes com qualidades de serviço variadas, modelos híbridos e dinâmicos (e mecanismos básicos relacionados) foram introduzidos em Gorender e Macêdo (2002), Gorender, Mâcedo e Raynal (2005), Macêdo (2007), Gorender, Macêdo e Raynal (2007), Macêdo e Gorender (2009) e Gorender e Macêdo (2010).

Em Gorender e Macêdo (2002), foi apresentado um algoritmo de consenso que requer uma *spanning tree* síncrona no sistema distribuído, onde processos são síncronos e canais de comunicação podem ser síncronos ou assíncronos. Nos trabalhos de Gorender, Mâcedo e Raynal (2005) e Gorender, Macêdo e Raynal (2007), o requisito de *spanning tree* síncrona foi removido e foram apresentadas soluções para o consenso uniforme em ambientes dinâmicos. Em Macêdo (2007), o modelo foi generalizado para que processos e canais de comunicação pudessem variar entre síncrono e assíncrono e foi apresentado (em Macêdo (2007) e Macêdo e Freitas (2009)) um algoritmo de comunicação em grupo capaz de ligar com ambientes híbridos e dinâmicos.

Em Macêdo e Gorender (2008) e Macêdo e Gorender (2009) foi introduzido o modelo híbrido e dinâmico *partitioned synchronous* que requer menos garantias temporais do que o modelo síncrono e através do qual foi provado ser possível a implementação de detectores perfeitos (mecanismo fundamental para a solução de consenso distribuído). Vale salientar que a implementação de detectores perfeitos no modelo *partitioned synchronous* não requer a existência de um *wormhole* síncrono (Veríssimo; Casimiro, 2002) ou *spanning tree* síncrona (Gorender; Macêdo, 2002), onde seria possível implementar ações síncronas globais em todos os processos, como sincronização interna de relógios. No sistema *partitioned synchronous* proposto, componentes do ambiente distribuído necessitam ser síncronos, mas os mesmos não precisam estar conectados entre si via canais síncronos, o que torna impossível a execução de ações síncronas distribuídas em todos os processos do sistema. E mesmo que parte dos processos não esteja em qualquer das componentes síncronas, pode-se ainda assim tirar proveito das partições síncronas existentes para melhorar a robustez das aplicações de tolerância a falhas.

Finalmente, em Macêdo e Gorender (2008) e Gorender e Macêdo (2011) explora-se o mo-

delo *partitioned synchronous* para propor uma solução robusta para o consenso distribuído. Em particular, a robustez de tal solução se aplica mesmo se a maioria dos componentes não estiver em partições síncronas, o que representa uma vantagem quando comparado com soluções para modelos convencionais. O estudo de soluções de tolerância a falhas para o modelo *partitioned synchronous* tem interesse prático uma vez que muitas configurações reais incluem componentes síncronas, como, por exemplo, processos em clusters em nuvens locais que se comunicam com processos clientes ou clusters em nuvens remotas através de redes de longa distância.

2.3 ASPECTOS DE CONFIABILIDADE EM SISTEMAS DISTRIBUÍDOS

A confiabilidade é uma propriedade relacionada à capacidade de um sistema em continuamente prover um serviço correto, i.e. de acordo com sua especificação (Avizienis et al., 2004). Entretanto, diversos são os elementos de incerteza, muitos dos quais imprevisíveis, que podem levar o sistema a falhar. Nesse sentido, um sistema é dito confiável se o mesmo entrega um serviço correto com alta probabilidade (Cristian, 1991).

Falhas, *Erros* e *Defeitos*[5] são elementos nocivos à confiabilidade de um sistema e devem ser considerados na implementação de sistemas confiáveis. Para caracterizar a ocorrência de um destes elementos nocivos, necessita-se de uma especificação bem definida dos requisitos do sistema – a especificação do sistema é a base para validação dos requisitos e das hipóteses de falhas.

Conceitualmente, um defeito denota um desvio entre o serviço que é entregue e o que foi especificado (Avizienis et al., 2004). Tal desvio é originado de um estado errôneo (i.e. erro) ao qual é levado o sistema na presença de uma falha. Um erro pode permanecer latente até que algum evento no sistema promova a sua ativação. A falha é um evento indesejado que pode inserir um erro latente no sistema. Dessa forma, pode-se afirmar que a falha é a causa indireta e primária do defeito – i.e. o evento de falha causa o estado de erro que, quando ativado, leva a um serviço defeituoso. Observe que, nesse sentido, *Falha*, *Erro* e *Defeito* são abreviações para denotar, respectivamente, *Evento de falha*, *Estado de erro* e *Serviço defeituoso*[6].

Os conceitos de Defeito e Falha são relativizados pela organização hierárquica dos sistemas. De modo geral, um sistema é composto por outros sistemas (ditos subsistemas ou componentes). Da mesma forma, estes subsistemas são estruturados a partir de outros componentes e assim por diante, até se alcançarem os subsistemas mais elementares (ou atômicos), para os quais esta subdivisão não é evidente ou não tem um sentido próprio – ver Figura 2.4.

Neste contexto, o desvio no serviço provido por um sistema (i.e. defeito) é uma conseqüência da falha de um de seus componentes. Este componente falho deixou de entregar, para o sistema usuário, o serviço da forma na qual foi especificado, assim, pode-se dizer que, no nível

[5] Traduz-se, do inglês, *fault*, *error* e *failure* como falha, erro e defeito, respectivamente – usando a mesma terminologia de Avizienis et al. (2004).

[6] A repetição dos conceitos de falha, erro e defeito no texto é intencional, para deixar clara a abordagem dada a estes conceitos neste trabalho – a qual segue a conceituação usada em Avizienis et al. (2004).

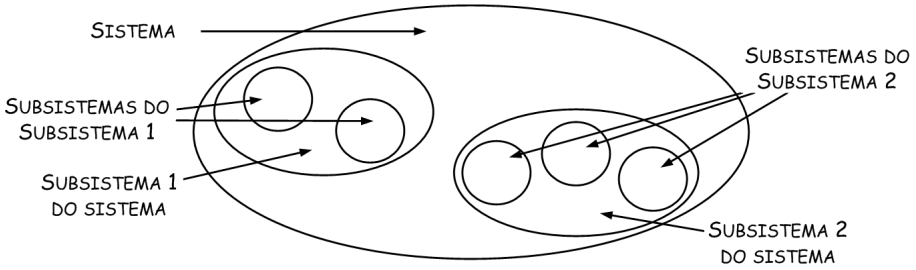


Figura 2.4. Hierarquia de sistema (Jalote, 1994)

desse subsistema, o mesmo está defeituoso. Da mesma forma, o defeito nesse subsistema foi provocado pela falha de um de seus componentes em um nível mais interno, e assim por diante – configurando, assim, uma cadeia de falhas, erros e defeitos.

A Figura 2.5, adaptada de Deswarte, Kanoun e Laprie (1998), apresenta um exemplo desse encadeamento entre falhas, erros e defeitos.

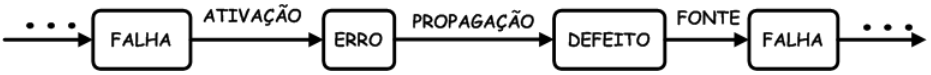


Figura 2.5. Cadeia de falha, erro e defeito

Na presença de falhas, erros e defeitos, a confiabilidade é obtida a partir de uma combinação de estratégias, apoiadas em diferentes técnicas e mecanismos que são implementados considerando fases que buscam desde a identificação do componente defeituoso até o tratamento da causa primária de tais defeitos. O uso das técnicas e de seus mecanismos associados define diferentes níveis de confiabilidade com diferentes custos, demandando, deste modo, um projeto que adequadamente concilie tais custos e níveis de confiabilidade aos requisitos demandados pelas aplicações e os tipos de falhas que precisam ser levados em consideração. Assim, o restante desta seção discute cada um desses aspectos, abordando-os com uma maior ou menor profundidade, de acordo com o grau de relevância dos mesmos para o entendimento da proposta apresentada neste trabalho.

2.3.1 Classificação das falhas

A tipificação das falhas é o elemento primário no projeto de um sistema confiável. As falhas podem ser tipificadas de diferentes formas, ver Avizienis et al. (2004) e Veríssimo e Rodrigues (2000). De um modo geral, as pesquisas na área em sistemas distribuídos têm focada a atenção em uma tipificação de falha a partir da maneira como as mesmas afetam o comportamento dos componentes do sistema. Nesse sentido, as falhas podem ser classificadas em:

- *Falha por parada* (*Crash fault*) – Ocorre quando, uma vez manifestada a falha, o componente pára de funcionar, deixando de responder a qualquer requisição de serviço. Esse é

o tipo de falha mais comum nos modelos de sistemas relacionados ao projeto de sistemas distribuídos.

- *Falha por omissão* (*Omission fault*) – Ocorre quando, dada uma entrada, o componente omite (ou deixa de produzir) o resultado de saída. Observe que a falha por parada é um tipo de falha por omissão, na qual o componente omite permanentemente os resultados de saída.

- *Falha temporal* (*Timing fault*) – Ocorre quando o componente responde fora do prazo especificado. Observe que a falha por omissão é um tipo de falha temporal, na qual o valor omitido nunca será entregue dentro do prazo.

- *Falha de valor* (*Value fault*) – Ocorre quando o componente produz uma saída incorreta para um dado valor de entrada.

- *Falha maliciosa* ou *bizantina* (*Byzantine fault*) – Ocorre quando um componente pode manifestar qualquer tipo de comportamento na presença de falha, podendo parar de funcionar, omitir resultados, produzir resultados fora do prazo ou ainda produzir valores incorretos.

A Figura 2.6 apresenta o relacionamento entre estes diferentes tipos de falhas.

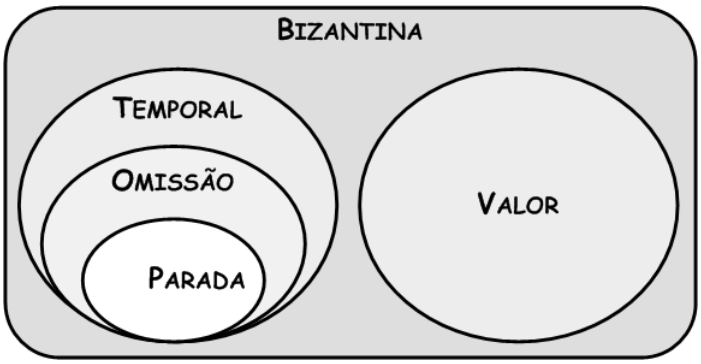


Figura 2.6. Tipificação das falhas de componente

2.3.2 Níveis de confiabilidade

De um modo geral, os sistemas podem apresentar diferentes níveis de confiabilidade. Estes níveis de confiabilidade estão relacionados à capacidade do sistema em atender a duas propriedades básicas: *liveness* e *safety*. Segundo Lamport e Lynch (1989), informalmente, a propriedade *safety* assegura que algo de ruim nunca acontece, enquanto que a propriedade *liveness* garante que alguma coisa boa em algum momento irá acontecer. De um modo mais formal, pode-se dizer que a propriedade *safety* está relacionada à correção das ações realizadas pelo sistema. A propriedade *liveness*, por sua vez, está relacionada à capacidade do sistema em se manter operacional (i.e. *live*).

A garantia das propriedades *safety* e *liveness* confere ao sistema quatro diferentes níveis de confiabilidade (Gärtner, 1999), ver Figura 2.7.

Figura 2.7. Nível de confiabilidade

Se o sistema não é capaz de garantir nem *safety* nem *liveness*, então o mesmo é dito não confiável. Se o sistema garante *safety*, mas não garante *liveness*, o mesmo é dito *fail-safe* – i.e. o mesmo não é capaz de continuar operacional, mas o mesmo realiza uma parada segura (ou desligamento seguro). Se o sistema garante *liveness*, mas não garante *safety*, o mesmo continuará operacional, podendo realizar algumas ações incorretas – neste caso a ocorrência da falha pode ser percebida pelo sistema usuário, mas o sistema continua a operar mesmo que de forma degradada. Observe que, neste último caso, é possível que ações automáticas de recuperação sejam realizadas de modo a permitir que o sistema volte a produzir resultados corretos. Por fim, se o sistema garante *safety* e *liveness*, o mesmo é capaz de mascarar a ocorrência da falha, não permitindo que a mesma seja percebida pelo sistema usuário. Um aspecto importante é que, quanto maior o nível de confiabilidade, maior é o custo de sua implementação. Deste modo, um sistema capaz de mascarar a ocorrência de falhas possui um custo de implementação maior que um sistema que garante apenas funcionamento degradado, e assim por diante.

2.3.3 Métricas de confiabilidade

Qualificar a confiabilidade em termos de propriedades *safety* e *liveness* não é suficiente para que se possa realizar uma determinação prática tanto da expectativa do usuário quanto do desempenho efetivamente entregue pelo sistema em termos de tal atributo (ou requisito de qualidade de serviço).

Na prática, duas implementações distintas de um mesmo serviço podem ambas realizar o mascaramento da falha, atendendo as propriedades de *safety* e de *liveness*, mas, ainda assim, podem prover diferentes desempenhos em termos de confiabilidade – uma das implementações pode, por exemplo, realizar recuperações de componentes defeituosos mais rapidamente ou apresentar menores taxas de falhas de componentes.

Além disso, tanto na perspectiva do projeto quanto na perspectiva do desempenho efetivo do sistema durante a execução, é necessário que esses diferentes níveis de qualidade de serviço em termos de confiabilidade sejam devidamente especificados ou obtidos, de modo a garantir uma projeto adequado ou permitir um gerenciamento mais eficaz do desempenho do sistema. Assim, para possibilitar uma avaliação quantitativa da confiabilidade e permitir que os sistemas sejam corretamente qualificados e diferenciados entre si, são usadas as seguintes métricas básicas (Avizienis et al., 2004):

- *Tempo médio de operação* ($MTTF$, *Mean time to failure*) – representa uma expectativa para o intervalo de tempo médio em que o sistema se mantém operacional até a ocorrência do defeito. De um modo geral, é obtido dividindo-se o somatório dos intervalos de tempo no qual o sistema esteve operacional pelo número de defeitos ocorridos.

- *Tempo médio do reparo* ($MTTR$, *Mean time to repair*) – representa uma expectativa para o intervalo de tempo médio para o reparo do sistema. De um modo geral, é obtido através da relação entre o total de tempo gasto com reparos e o número de reparos realizados durante um dado período de tempo.

- *Tempo médio entre defeitos* ($MTBF$, *Mean time between failures*) – representa uma expectativa para o intervalo de tempo entre defeitos do sistema. De um modo geral, é obtido dividindo-se o tempo de vida do sistema pelo número de defeitos ocorridos.

A Figura 2.8 apresenta um representação dessas métricas durante a vida útil do sistema – a partir da mesma é possível observar que: $MTBF = MTTF + MTTR$.

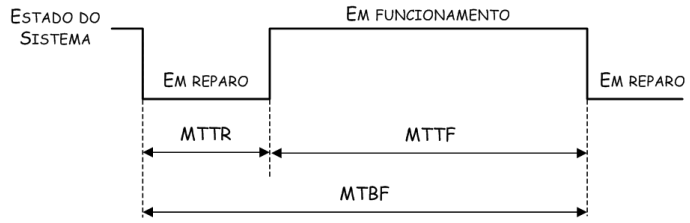


Figura 2.8. Métricas de confiabilidade

Usando as métricas básicas é possível definir a disponibilidade do sistema (AV). Tal métrica representa uma medida da capacidade do sistema em estar apto para uso (i.e. disponível) quando requisitado – podendo ser obtida a partir da relação entre tempo médio $MTTF$ e $MTBF$, isto é: $AV = \frac{MTTF}{MTBF} = \frac{MTTF}{MTTF+MTTR}$

2.3.4 Estratégias para a obtenção da confiabilidade

Prevenção e Tolerância a Falhas são as estratégias básicas para tornar os sistema confiáveis, permitindo que se possa atribuir *Confiança ao funcionamento*[7] dos mesmos (Avizienis et

[7]Tradução do inglês para *Dependability*

al., 2004). A *Prevenção de Falhas*[8] baseia-se na utilização de técnicas e componentes robustos durante a construção do sistema, para evitar possíveis falhas durante o funcionamento. A *Tolerância a Falhas*[9], por outro lado, utiliza a redundância para garantir a continuidade do funcionamento mesmo na presença de falhas.

Por melhores que sejam as técnicas e componentes utilizados durante a construção do sistema é impossível prevenir todas as falhas: eventos físicos externos podem levar o sistema a falhar, componentes do sistema podem se desgastar com o tempo ou, ainda, alguns sistemas podem ser extremamente complexos e de composição extremamente heterogênea, tornando a prevenção de falhas uma atividade extremamente complicada. Desta forma, dotar o sistema da habilidade de tolerar falhas é extremamente necessário.

Prevenção e Tolerância a Falhas são estratégias essenciais e que devem ser aplicadas em conjunto para garantir confiança no funcionamento dos sistemas. Apesar de a Prevenção de Falhas ser uma abordagem de extrema importância para a obtenção da confiabilidade, a mesma está fora do escopo deste trabalho.

2.3.5 Fases em tolerância a falhas

Tolerar falhas é uma característica fundamental para sistemas críticos – isto é, sistemas nos quais a ocorrência de falhas pode levar a situações em que a conseqüência da falha implica em custos superiores aos benefícios esperados durante a operação normal do sistema, por exemplo: grandes perdas financeiras, desastres ambientais, perda de vidas humanas etc.

Nesse contexto, a implementação da habilidade de tolerar falhas se baseia na utilização de técnicas e mecanismos subjacentes (ditos mecanismos de tolerância a falhas), os quais não estão relacionados aos benefícios do sistema durante a operação normal (i.e. livre de falhas), mas que têm importância para o contingenciamento do sistema no caso de falhas em algum de seus componentes.

A concepção de sistemas confiáveis, considerando o uso dessas técnicas e desses mecanismos de tolerância a falhas, exige, dos projetistas, um trabalho árduo de engenharia durante o projeto. Isto porque, durante o projeto, além das especificações funcionais e temporais a serem atendidas, a concepção dos mecanismos de tolerância a falhas exige que os projetistas considerem diferentes fases, as quais estão relacionadas às ações que devem ser tomadas por um ou mais mecanismos para contornar os defeitos de componentes e garantir a continuidade do serviço. Tais fases podem ser divididas em (Jalote, 1994): detecção do erro (*error detection*) no estado de algum dos componentes; confinamento dos riscos (*damage confinement*) de propagação do erro; processamento do erro (*error processing*) para restaurar o estado correto; e tratamento da falha (*fault treatment*) para evitar a reincidência de erros. A seguir, cada uma destas fases são discutidas com maiores detalhes.

[8]Tradução do inglês para *Fault-Avoidance*
[9]Tradução do inglês para *Fault-Tolerance*

2.3.5.1 Detecção de erros

Uma vez que a manifestação da falha pode levar o sistema a um estado de erro (Avizienis et al., 2004) e a ativação do mesmo é a causa do aparecimento do defeito no sistema, deve-se detectar o estado de erro para que medidas possam ser tomadas de modo a evitar que o defeito se manifeste no serviço. Dado que a ocorrência da falha é percebida a partir do estado de erro, a detecção de erro implica na detecção da ocorrência da falha (i.e. detecção de falha). Mais ainda, a falha no sistema é a manisfestação do defeito em algum de seus componentes, assim a detecção de falha no sistema pode também ser chamada de detecção de defeitos em componentes (i.e. detecção de defeitos).

Segundo Jalote (1994), os principais testes aplicados no processo de detecção de erro podem ser:

- *Testes por replicação* (*Replication checks*). Esta categoria de teste faz uso de replicação de componentes do sistema para checar se o valor produzido pelo componente replicado é correto. Em geral, nas implementações deste tipo de teste os resultados entre os diferentes componentes replicados são comparados a fim de se detectar erros no resultado gerado. Um comparador (ou votador) recolhe os diversos resultados e utiliza uma função para selecionar o valor correto. A quantidade de componentes replicados a serem utilizados nestes testes dependerá da estrutura do sistema e do tipo de defeito que se quer detectar.

- *Testes temporais* (*Temporal checks*). Os testes temporais são utilizados para verificar se os componentes do sistema estão cumprindo os prazos (deadlines) para realização de uma determinada tarefa. Normalmente, esta categoria de teste é realizada através de temporizadores que atuam de acordo com a especificação de tempo, verificando se o componente possui um comportamento temporal consistente.

- *Testes estruturais* (*Structural checks*). Este tipo de testes é utilizado quando se deseja fazer a verificação da validade da estrutura utilizada para armazenar um dado. De modo geral, esta categoria de teste faz uso de informações redundantes que pode indicar erros na estrutura do dado. Um exemplo de teste estrutural é aquele que é feito na validação de código móvel antes que o mesmo seja posto em execução – e.g. antes da execução de *applets Java*, por exemplo, existe um processo de validação em que primeiramente se usa um conjunto de restrições estáticas embutidas no código para observar se o *byte-code* (i.e. código objeto) é composto por uma seqüência de instruções válidas e, em seguida, um conjunto de assertivas são usadas para verificar se os tipos de dados manipulados são válidos (i.e. compatíveis com as regras do Java), ver Yellin (1995) e Leroy (2003).

- *Testes de código* (*Coding checks*). Tipo de teste estrutural muito utilizado em hardware para chegar se o conjunto de bits de um código é realmente válido. Este teste é caracterizado pela inserção de bits extras com informação sobre a estrutura do código a ser checado. Caso o código seja corrompido, não haverá correspondência entre a estrutura

do código e a informação do conjunto de bits de checagem adicionais e, desta forma, o erro poderá ser detectado.

- *Testes de coerência* (*Reasonableness checks*). Nesta categoria de teste, é verificado se as informações sobre o estado do sistema estão coerentes com as faixas de consistência estabelecidas pela especificação. Um exemplo de teste de coerência é a verificação se um valor está dentro de uma faixa especificada.

- *Auto-teste* (*Diagnostics checks*). Neste tipo de teste, o módulo detector de erro faz parte do próprio sistema. Através desse módulo, o sistema realiza uma série de testes para verificar se algum de seus componentes está defeituoso. Em geral, o auto-teste utiliza valores fixos de entrada e seus respectivos valores de saída armazenados em uma memória do sistema. Os valores de entrada são submetidos aos seus respectivos componentes e a saída gerada é comparada com a saída armazenada no detector do sistema. Caso qualquer um dos componentes não produza valores iguais aos esperados para a saída, o sistema detecta que o componente em questão está defeituoso.

Observe que a detecção de *crashs* é uma das abordagens mais utilizadas em sistemas distribuídos. Esse método de detecção se baseia no uso de testes temporais para verificar o funcionamento do componente. Muitos algoritmos e protocolos para sistemas distribuídos confiáveis se baseiam no uso desse método de detecção para cumprirem corretamente seus objetivos. Todavia, uma questão importante abordada na literatura é como detectar falhas por *crash* em ambientes em que os limites temporais para processamento ou transmissão das mensagens são incertos. Essa questão motivou o desenvolvimento de inúmeras estratégias de detecção de falhas por *crash*[10]. Por conta disto e do estudo de caso apresentado no Capítulo 4, os aspectos da detecção de falhas por *crash* em sistemas distribuídos são discutidos com maiores detalhes na Seção 2.4.1 do Capítulo 2.4.

2.3.5.2 Confinamento do risco

A detecção de um erro não é instantânea, os testes dos componentes do sistema podem consumir certa quantidade de tempo e ainda o processo de detecção pode não ser ativado de forma contínua, mas em ciclos periódicos – por estas razões, a detecção da falha não é imediata. Neste intervalo, é possível que o defeito se propague do componente defeituoso para os demais componentes com os quais este interage. Portanto, determinar a extensão de atuação do erro e realizar o confinamento das possíveis regiões de risco é importante.

As suposições quanto aos limites de disseminação do erro podem ser feitas de forma dinâmica, utilizando informações sobre o fluxo de comunicação entre os componentes, ou de forma estática, inserindo estaticamente barreiras (*firewalls*) que evitem a propagação do erro antes do

[10]Ver exemplos em Doudou, Garbinato e Guerraoui (2002), Raynal (2005) e Pasin, Fontaine e Bouchenak (2008).

próximo ciclo de detecção (Jalote, 1994). Em relação ao confinamento do risco, Alima et al. (2002) ressaltam quatro técnicas básicas:

- *Separação física* (*Physical separation*) – Nessa estratégia a mesma versão de um componente (ou processo) é executada em diferentes localizações físicas. Assim, quando uma das versões é afetada por uma falha física, uma outra versão, livre de erros, passa a responder as requisições de serviço. Observe que esta técnica possui pelo menos duas limitações: (i) se as diferentes versões precisarem realizar sincronização de estado, é possível que a versão defeituosa contamine as demais; (ii) até que o estado de erro seja detectado, esta técnica não garante que os componentes consumidores do serviço provido pelo componente defeituoso fiquem livres do risco de contaminação.

- *Redundância N-modular* (*N-modular redundancy*) – Esta estratégia é a mesma usada no *teste de replicação*, isto é, um mesmo valor de entrada é submetido a *N* réplicas de um mesmo componente e o resultado gerado pelos mesmos é então comparado. Diferentes níveis de confinamento podem ser realizados usando esta técnica: (i) com dois componentes, é possível detectar (sem identificar) o componente defeituoso e simular uma falha por omissão, evitando que a informação se propague (Cristian, 1991); (ii) com três ou mais réplicas, é possível detectar (e possivelmente identificar) os componentes defeituosos, isolar o mesmo e propagar apenas o valor correto.

- *Proteção* (*Protection*) – Nesta técnica, os componentes têm acesso apenas àqueles recursos necessários durante sua operação normal, o que evita que um componente defeituoso acesse recursos não permitidos. Um exemplo clássico de confinamento de risco é encontrado em esquemas de proteção implementados nos sistemas multiprogramados – nestes esquemas uma combinação de facilidades de hardware e de sistema operacional são usadas como barreiras para evitar, por exemplo, que processos de um usuário acessem regiões de memória dos outros processos, reduzindo, dessa forma, o risco de um processo falho interferir nos demais (Lampson, 1974; Tanenbaum, 2007). A técnica de confinamento por proteção pode ser divida em duas categorias: (i) lista de acesso, em que para cada componente compartilhável existe uma lista que determina quais componentes usuários possuem autorização para uso – um componente guardião verifica se os componente compartilháveis são acessados apenas por aqueles componentes que possuem autorização para tal; e (ii) competências (*capabilities*), em que cada componente tem um *ticket* (i.e. competência) que lhe concede acesso a um componente compartilhável – nesse tipo de proteção, é possível que um componente usuário passe o seu ticket para um componente provedor de serviço, o que torna a estratégia mais flexível, mas também a torna mais frágil.

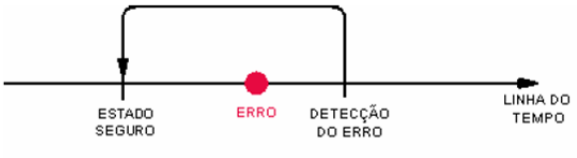
- *Ações atômicas* (*Atomic actions*) – está técnica consiste em estruturar as atividades internas de um sistema em ações atômicas, de modo que, se um erro acontece durante a execução de uma ação atômica, então o estados dos componentes que participaram de

tal ação é recuperado – evitando, assim, que o erro se propague para os fluxos de ações livres de erro. Ações atômicas é uma estratégia comum em banco de dados distribuídos (Silberschatz; Korth; Sudarshan, 2006, Cap. 22).

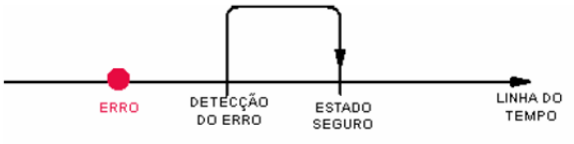
2.3.5.3 Processamento do erro

Confinado o erro, é necessário realizar um processamento que possa recuperar o sistema do estado de erro (*error recovery*) ou compensar o erro através do uso de redundância (*error compensation* ou *error masking*).

A recuperação do erro consiste em substituir o estado errôneo por um estado livre de erros, podendo ser feita de duas formas básicas (Campbell; Randell, 1986):



(a) Recuperação por retrocesso



(b) Recuperação por avanço

Figura 2.9. Processamento do erro usando recuperação

- *Recuperação por retrocesso* (*backward error recovery*) – consiste em trazer o sistema para um estado consistente antes da ocorrência do erro, ver Figura 2.9(a). Para tanto, se baseia, de modo geral, na criação periódica de pontos de recuperação (i.e. *checkpoints*), consistindo no armazenamento estável do último estado correto, para o qual o sistema será reiniciado em caso de falha (Wilfredo, 2000).

- *Recuperação por avanço* (*forward error recovery*) – consiste em, na presença do erro, encontrar um estado consistente que nunca tenha ocorrido antes (ou que não tenha ocorrido até a presença do erro), ver Figura 2.9(b). Para que esta técnica seja bem sucedida, é necessário que a extensão do dano provocado pela falha seja cuidadosamente estimada. Um exemplo clássico de implementação dessa técnica é encontrada no uso do mecanismo de manipulação de exceções, ver Randell e Xu (1995).

O processamento do erro por recuperação demanda fortemente uma abordagem de detecção de erros, implicando que o tempo de resposta de mecanismo é dependente não apenas do tempo de recuperação, mas também do tempo de detecção. Esse aspecto pode ser um complicador em

sistemas com restrições temporais mais fortes (e.g. sistemas de tempo real) – principalmente quando a recuperação por retrocesso é considerada, ver Kopetz (1997, Cap. 1).

A técnica de *compensação do erro*, por outro lado, é feita através do uso de redundância de forma a permitir a entrega do serviço mesmo na presença do estado errôneo, permitindo que o erro seja mascarado – i.e. não seja percebido pelo usuário do serviço.

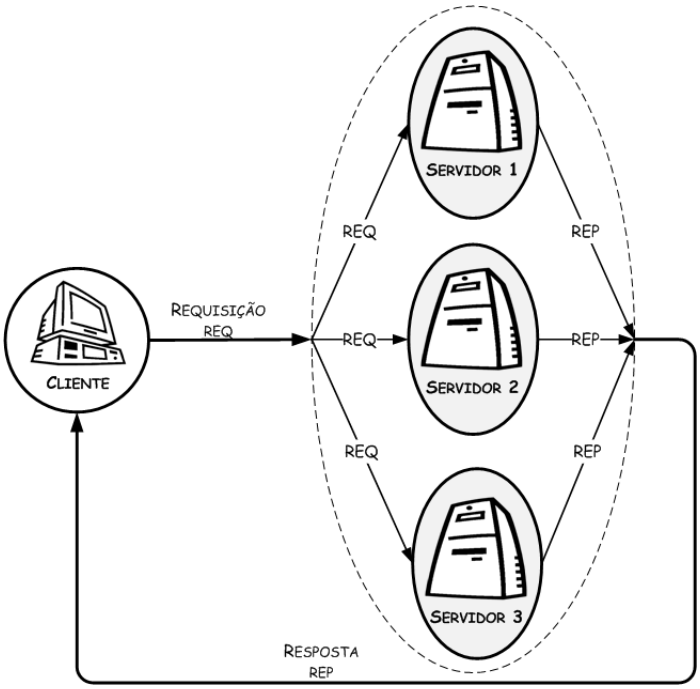


Figura 2.10. Processamento do erro por compensação

A Figura 2.10 mostra um exemplo da técnica de compensação. Nesse exemplo um cliente envia requisições ao conjunto de servidores, os quais as executam na mesma ordem e produzem resultados de volta para o cliente. Neste caso, a falha de algum dos servidores pode ser compensada pelo processamento redundante executado pelos demais. Uma discussão mais detalhada de exemplos desse tipo de estratégia pode ser encontrada em Cristian (1991).

2.3.5.4 Tratamento da falha

Uma vez processado o erro, é preciso que o tratamento da falha causadora do erro seja executado. Este tratamento implica na identificação do componente defeituoso e na reconfiguração do sistema de modo a prevenir que tal componente venha a ocasionar o defeito de forma recorrente (Jalote, 1994).

A localização da falha pode ser realizada usando três estratégias básicas (Alima et al., 2002): diagnóstico, replicação ou votação. A localização por diagnóstico implica no uso da técnica de detecção baseada em auto-teste e é usada principalmente na identificação em falhas em componentes de hardware. A principal dificuldade do uso dessa estratégia em componente de software é o tempo consumido para realização das checagens – por conta disso, normalmente a checagem em software é feita *off-line*. A localização por replicação usa a detecção por replicação,

de modo que o componente defeituoso é aquele cuja saída diverge da maioria. Na localização por votação, por sua vez, um componente é dito defeituoso quando a maioria dos componentes declaram o mesmo como defeituoso. Observe que a localização por replicação e por votação são bastante similares. Contudo, na localização por replicação, o componente defeituoso é localizado a partir da saída produzida, enquanto que na localização por votação o mesmo é explicitamente declarado como defeituoso pelos demais componentes.

A reconfiguração do sistema pode ser feita de forma manual (i.e. por intervenção humana) ou de forma dinâmica (i.e. em resposta a um comando do administrador do sistema ou a partir de mecanismos inseridos no sistema). A reconfiguração dinâmica a partir de mecanismos de reconfiguração é também denominada de reconfiguração espontânea (Alima et al., 2002).

2.3.6 Técnicas de Tolerância a falhas

O uso de redundância é o elemento chave para a implementação de tolerância a falhas. A redundância é a parte desnecessária para o regime normal de funcionamento, mas que é requerida para garantir o funcionamento do sistema na presença de falhas (Jalote, 1994).

De forma geral, a redundância pode ser classificada em (Veríssimo; Rodrigues, 2000): *Redundância espacial*; *Redundância temporal*; e *Redundância de valor*.

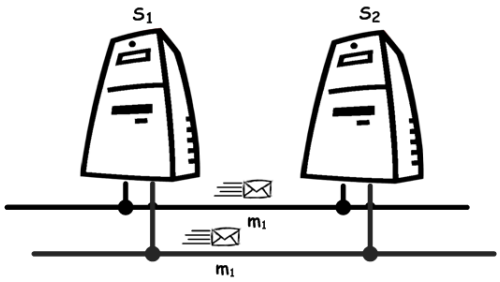


Figura 2.11. Exemplo de redundância espacial

Na *Redundância espacial*, componentes extras são adicionados de modo que possam realizar um mesmo processamento em paralelo – capturando as mesmas entradas, executando as mesmas ações e gerando os mesmos resultados. Esse tipo de redundância garante que, quando um componente falha, o resultado do processamento realizado estará disponível em qualquer um dos demais componentes replicados. A Figura 2.11 apresenta um exemplo de um esquema de redundância espacial, em que os canais de comunicação são duplicados – observe na Figura que S_1 envia simultaneamente, em ambos os canais, a mesma mensagem para S_2.

Na *Redundância temporal*, o sistema é projetado de modo que exista uma folga de tempo para que um mesmo processamento possa ser realizado diversas vezes da mesma forma ou de formas diferentes. A retransmissão de mensagens em uma rede é um exemplo de redundância temporal para tolerar falhas transientes na rede, ver Figura 2.12.

Na *Redundância de valor*, informações extras são adicionadas ao valor de um dado, de modo que procedimentos de detecção ou correção de erros possam dar garantias à integridade

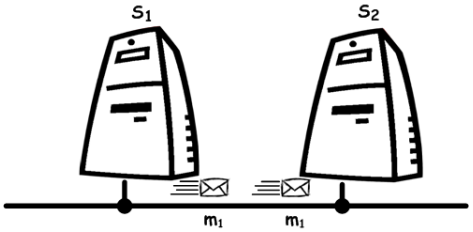


Figura 2.12. Exemplo de redundância temporal

do dado. A técnica de paridade combinada, usada em redes de transmissão de dados, é um exemplo da redundância de valor, ver Figura 2.13 – nestes esquemas os bits dos dados são organizados em blocos e é gerado um bit de paridade por palavra (i.e. paridade vertical) e por bits alinhados (i.e. paridade horizontal), assim, combinando as paridades vertical e horizontal, é possível detectar e corrigir pelo menos um bit defeituoso, evitando, desse modo, a necessidade de retransmissões.

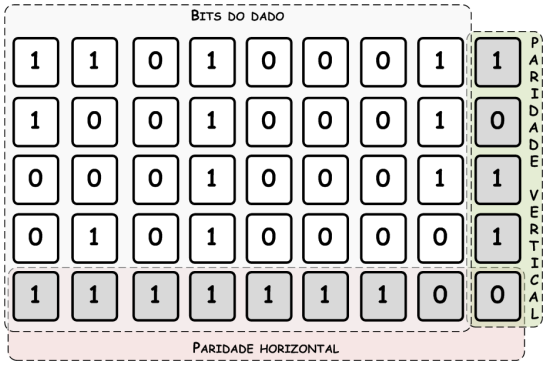


Figura 2.13. Exemplo de redundância de valor

2.4 MECANISMOS DE TOLERÂNCIA A FALHAS

A tolerância falhas em sistemas distribuídos é provida a partir das facilidades oferecidas por mecanismos de suporte básico, ditos mecanismos de tolerância a falhas. A implementação destes mecanismos implica em custos adicionais, uma vez que os mesmos necessitam de processamento e troca de mensagens adicionais para cumprirem corretamente seus objetivos. Assim, apesar de necessários em situações de falha, os mecanismos de tolerância a falhas podem comprometer o desempenho do sistema durante o funcionamento normal. Com isso, o projetista deve conciliar estes custos adicionais com a estimativa da carga das aplicações, com os requisitos de desempenho e com o grau de confiabilidade desejada para o sistema.

Na literatura, são diversos os mecanismos de tolerância a falhas que dão suporte à implementação de sistemas distribuídos confiáveis, por exemplo: detectores de defeitos; protocolos de comunicação em grupo; mecanismos de sincronização de estados entre servidores replicados; protocolos para replicação bizantina, entre outros. Estes mecanismos variam de acordo com a severidade da falha a ser mascarada e com o modelo de sistema distribuído que os mes-

mos se propõem a suportar. Dentre estes mecanismos, detectores de defeitos e os protocolos de comunicação em grupo são componentes básicos para a implementação de muitas das facilidades necessárias aos demais mecanismos.

Tome, como exemplo, um esquema no qual um serviço, apoiado na técnica de replicação, é concebido usando um servidor primário e servidores secundários, ver Figura 2.14. Neste exemplo, detectores de defeitos são usados pelos servidores secundários para monitorar falhas do servidor primário e possibilitar a reconfiguração do sistema – notificando, por exemplo, os mecanismos responsáveis por promover um servidor secundário a servidor primário. Além disso, os protocolos de comunicação em grupo são fundamentais na difusão (ou sincronização) continuada dos estados entre os servidores, permitindo que um servidor secundário esteja apto a assumir o papel do primário na ocorrência de falhas.

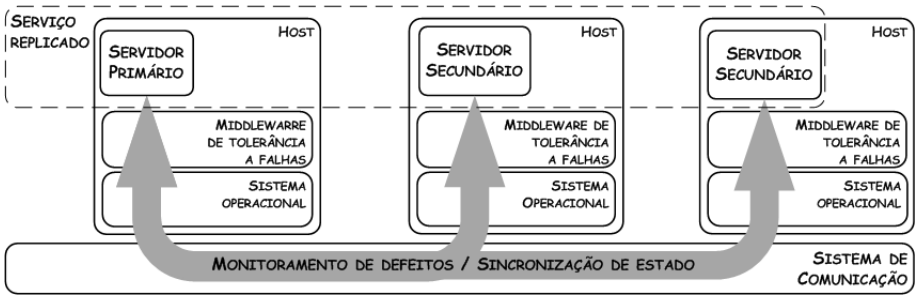


Figura 2.14. Exemplo de aplicação distribuída com servidores replicados

Os detectores de defeitos são usados, no Capítulo 4, para a demonstração da proposta de detecção autonômica de defeitos. Assim, o restante desta seção discute alguns conceitos básicos associados à implementação dos detectores de defeitos para sistemas distribuídos, focando, para tanto, em: estilos de monitoramento; classes de detectores de defeitos; qualidade de serviço em detecção de defeitos; e detectores adaptativos.

Protocolos de comunicação em grupo são usados no Capítulo 5 para a demonstração da proposta autonômica de comunicação em grupo. No entanto, uma revisão destes protocolos não é tratada neste Capítulo, uma vez que uma discussão bastante extensa e concisa sobre os principais conceitos destes protocolos podem ser encontrada em Chockler, Keidar e Vitenberg (2001) e Défago, Schiper e Urbán (2004).

2.4.1 Detecção de defeitos para sistemas distribuídos

No contexto dos sistemas distribuídos, o detector de defeitos é um serviço com módulos distribuídos entre os processos que fazem parte do sistema (Chandra; Toueg, 1996). Esses módulos se comunicam com o intuito de trocar informações a respeito dos estados dos processos que estão sendo monitorados. A depender do estilo de monitoramento usado, essa troca de mensagens pode implicar em maiores ou menores custos de computacionais (Felber, 1998).

A detecção de defeitos por *crash* é uma estratégia de detecção de defeitos temporal (ver Seção 2.3.5.1), em que os módulos monitores do detector de defeitos usam temporizadores

para determinar prazos (i.e. *timeouts*) para chegada das mensagens de monitoramento oriundas dos processos monitorados – o não atendimento destes prazos sinaliza possíveis falhas dos processos monitorados.

Em sistemas distribuídos síncronos, uma vez que os limites temporais para o processamento e transmissão das mensagens são conhecidos, os *timeouts* podem ser estabelecidos com precisão – assim, quando um processo monitorado não atende ao prazo especificado, o mesmo é tido como falho. Em sistemas assíncronos, por outro lado, os limites temporais para o processamento e transmissão das mensagens são desconhecidos e podem variar de forma arbitrária. Por conta disso, a definição de *timeouts* é um problema e não se pode determinar com certeza se um processo monitorado efetivamente falhou ou se a mensagem de monitoramento está atrasado (Fischer; Lynch; Paterson, 1985). Por conta disso, quando uma mensagem não chega no prazo, o processo monitorado é apenas suspeito de ter falhado (Chandra; Toueg, 1996). Neste sentido, para reduzir o número de falsas suspeitas de falhas, *timeouts* mais longos podem ser usados, o que implica em longas latências de detecção dos defeitos e pode comprometer o desempenho das aplicações, ou dos demais mecanismos de tolerância a falhas, que dependem do serviço do detector para cumprirem seus objetivos com segurança.

Na tentativa de evitar não apenas o uso de *timeouts* muito longos (o que é prejudicial para a velocidade das detecções), mas também reduzir o número de falsas suspeitas de falhas (melhorando a confiabilidade do detector), estratégias de adaptação de *timeout* vêm sendo usadas[11]. Estas estratégias utilizam estimadores de atrasos, os quais se baseiam em alguma informação a respeito dos atrasos de comunicação fim-a-fim (ou das variação desses atrasos) para realizar suas estimativas.

Detectores de defeitos em sistemas assíncronos são não confiáveis, pois podem apontar a falha de um processo correto, ou deixar de apontar a falha de um processo efetivamente defeituoso (Chandra; Toueg, 1996). Esses detectores podem ser classificados de acordo com a sua capacidade de detectar defeitos (i.e. correção) e de evitar falsas suspeitas (i.e. precisão). Esta classificação é importante para determinar o potencial dos detectores de defeitos na resolução de problemas fundamentais de tolerância a falhas em sistemas distribuídos (Raynal, 2005) – sendo usada para auxiliar na composição de modelos de sistemas de distribuídos e ajudar na prova das propriedades de *safety* e *liveness* dos algoritmos de tolerância a falhas.

Contudo, essa classificação, baseada na correção e precisão, não é suficiente para permitir que os aspectos de desempenho (e.g. velocidade e confiabilidade) dos detectores de defeitos sejam determinados – o que é um problema para a implementação prática de sistemas distribuídos confiáveis. Com isso, métricas de qualidade de serviço são usadas no projeto e avaliação dos detectores de defeitos, permitindo que os mesmos possam entregar um serviço adequado aos requisitos das aplicações distribuídas (Chen; Toueg; Aguilera, 2002).

O entendimento dos conceitos apresentados acima são de particular interesse para algumas

[11] ver, por exemplo, Macêdo (2000), Chen, Toueg e Aguilera (2002), Bertier, Marin e Sens (2002), Nunes e Jansch-Pôrto (2004) etc.

das discussões apresentadas nos capítulos 4 e 5 desta Tese – por conta disso, esses conceitos são discutidos com maiores detalhes a seguir.

2.4.1.1 Estilos de monitoramento de defeitos

Em um ambiente distribuído, a implementação de detectores de defeitos considera dois estilos básicos para o monitoramento remoto do estado dos processos do sistema: *Pull* e *Push* (Felber, 1998).

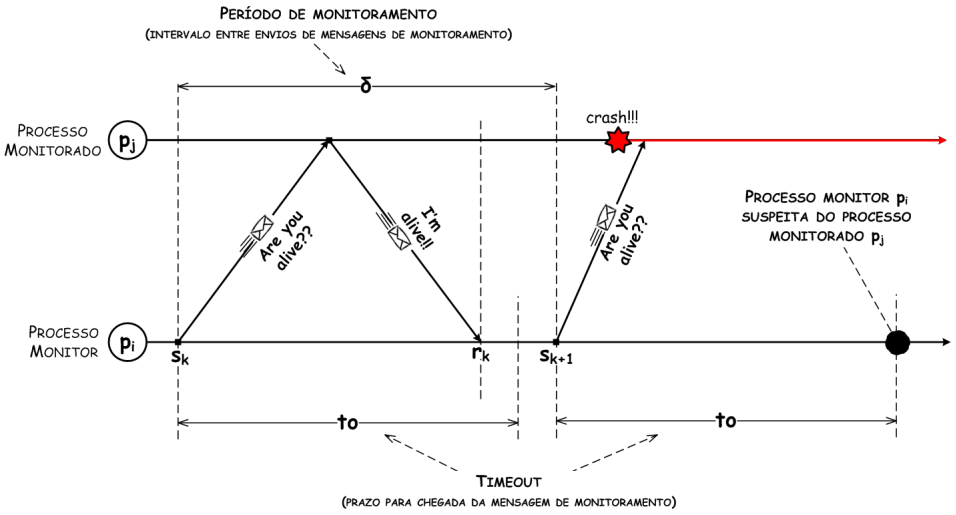


Figura 2.15. Estilo de monitoramento Pull

No estilo de monitoramento *Pull*, Figura 2.15, o processo monitor (p_i) do detector de defeitos envia periodicamente uma mensagem de monitoramento do tipo *"Are you Alive?"*, dita *aya*. Uma vez recebida a mensagem de *"Are you Alive?"*, o processo monitorado (p_j) deve responder com seu atual estado usando uma mensagem de monitoramento do tipo *"I am alive!"* ou *heartbeat*, dita *hb*. A cada mensagem de monitoramento enviada, o processo monitor deve estimar o intervalo de tempo necessário (*timeout*) para a chegada da mensagem de resposta oriunda do componente monitorado. Caso a mensagem não chegue dentro do intervalo esperado, o processo monitor passa a suspeitar da falha do processo monitorado. Neste estilo de monitoramento, o processo monitor controla o ritmo do monitoramento e calcula os timeouts baseados nos atrasos de ida-e-volta (*round-trip-time*). Sendo assim, uma abordagem de detecção, construída a partir de tal estilo de monitoramento, consegue não apenas ter uma estimativa mais eficiente dos atrasos de comunicação, mas também pode controlar de forma mais eficiente o ritmo (i.e. período) de monitoramento.

No estilo de monitoramento *Push*, Figura 2.16, o processo monitorado (p_j) espontaneamente envia seu estado atual para o processo monitor (p_i). Baseado no intervalo entre chegada das mensagens, o processo monitor deve estimar o instante de chegada da próxima mensagem de monitoramento. Caso a mensagem não chegue dentro do intervalo esperado, o processo monitor suspeita da falha do processo monitorado. Neste estilo de monitoramento, o número de

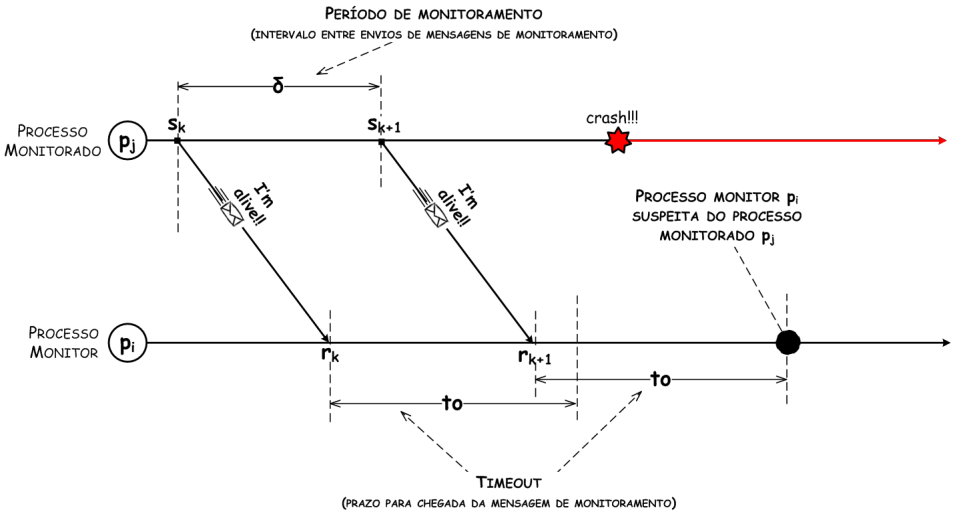


Figura 2.16. Estilo de monitoramento Push

mensagens de monitoramento trocadas entre os módulos monitor e monitorado é menor que no estilo *Pull*, o que implica em um menor consumo de recursos do canal de comunicação.

2.4.1.2 Classes de detectores de defeitos

A detecção de defeitos por *crash* é fundamentalmente apoiada pela realização de testes temporais (ver Seção 2.3.5.1). Enquanto o uso destes testes é razoável em sistemas de tempo real (i.e. síncronos), parece ser um ponto controverso em sistemas livres de tempo (i.e. *time-free* ou assíncronos). Entretanto, sistemas reais não são necessariamente livres de tempo, possuindo algum nível de sincronia (ver Seção 2.2). Por outro lado, embutir explicitamente essas hipóteses de sincronia nos algoritmos pode comprometer as propriedades de resiliência a falhas[12], de generalidade e de reuso – principais benefícios dos algoritmos que consideram modelos de sistemas assíncronos (Hermant; Lann, 2002; Lann; Schmid, 2003).

Neste contexto, Chandra e Toueg (1996) apresentam os detectores de defeitos não confiáveis, como uma estratégia para ocultar a sincronia necessária à resolução de problemas de tolerância a falhas em sistemas distribuídos assíncronos, sem a necessidade explícita de embutir as hipóteses de sincronia nos algoritmos e protocolos que fazem uso desses detectores.

O argumento de Chandra e Toueg (1996) se baseia no fato de que sistemas reais apresentam períodos de estabilidade (GST, *Global Stabilization Time*), nos quais os atrasos de processamento e troca de mensagens são desconhecidos, mas limitados – o que permite que um teste temporal possa em algum momento detectar, de forma definitiva, a falha por *crash*. Assim, é possível embutir essa hipótese de sincronia no detector de defeitos, sem a necessidade de a mesma ser considerada na implementação dos algoritmos – bastando, portanto, que os algoritmos acreditem que em algum momento o detector de defeitos apontará como defeituoso um

[12]Note que, se um algoritmo é confiável em sistemas livres de tempo, então o mesmo também é confiável em sistemas com alguma nível de sincronia – dado que a sincronia é ortogonal a confiabilidade de tal algoritmo.

processo que tenha falhado e deixará de apontar, como falho, um processo correto.

A qualidade de serviço dos detectores de Chandra e Toueg (1996) varia de acordo com as propriedades de correção e precisão, em que:

- *completeness* – determina a capacidade do detector em identificar os processos que tenham falhado. De acordo com essa propriedade, o detector pode apresentar os seguintes níveis de qualidade de serviço:
  - *strong completeness* – todo processo defeituoso é permanentemente suspeito de falha por todo processo correto.
  - *weak completeness* – todo processo defeituoso é permanentemente suspeito de falha por algum processo correto.
- *accuracy* – determina a capacidade do detector em evitar erros de detecção (i.e. falsas suspeitas de falhas). De acordo com essa propriedade, o detector pode apresentar os seguintes níveis de qualidade de serviço:
  - *strong accuracy* – nenhum processo correto é suspeito de falha.
  - *weak accuracy* – algum processo correto nunca é suspeito de falha.
  - *eventual strong accuracy* - existe um tempo depois do qual nenhum processo correto é suspeito de falha.
  - *eventual strong accuracy* – existe um tempo depois do qual algum processo correto deixa de ser suspeito de falha.

De acordo com essas propriedades, oito classes de detectores podem ser definidas, ver Tabela 2.1. Dentre estas classes, os detectores das classes $\mathcal{P}$ e $\Diamond\mathcal{W}$ são os que apresentam o maior e o menor nível de qualidade de serviço, respectivamente.

Tabela 2.1. Classes de detectores de defeitos

Correção	Precisão			
	Strong	Weak	Eventually Strong	Eventually Weak
Forte	$\mathcal{P}$	$\mathcal{S}$	$\Diamond\mathcal{P}$	$\Diamond\mathcal{S}$
Fraca	$\mathcal{D}$	$\mathcal{W}$	$\Diamond\mathcal{D}$	$\Diamond\mathcal{W}$

2.4.1.3 Qualidade de serviço de detecção de defeitos

A classificação proposta por Chandra e Toueg (1996) é baseada em comportamentos temporais futuros não quantificáveis, sendo, portanto, não apropriada para algumas aplicações que necessitem que a qualidade de serviço dos detectores de defeitos seja expressa de forma mais explícita – apresentando, quantitativamente, a velocidade e a precisão do serviço de detecção.

Nesse sentido, Chen, Toueg e Aguilera (2002) propõem métricas probabilísticas de qualidade de serviço para medir a capacidade do detector em prevenir falsas suspeitas (*precisão*) e a rapidez com a qual o mesmo detecta falhas. Estas métricas estão divididas em métricas primárias e secundárias, em que:

- **Métricas Primárias**, as quais não podem ser deduzidas de quaisquer outras métricas, mas a partir das quais se deduz as demais. Essas métricas são:

  - **Tempo de detecção** (T_D, *Detection Time*) – intervalo de tempo necessário para que um processo monitor (p_j) suspeite a falha de um processo monitorado (p_i). Esse intervalo é medido a partir do instante no qual o processo monitorado efetivamente falhou (ver Figura 2.17).

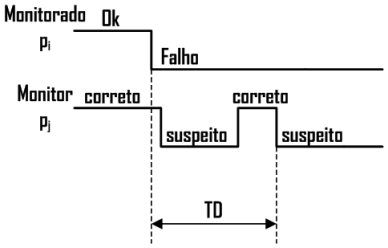


Figura 2.17. Tempo de detecção (TD)

  - **Intervalo entre falsas suspeitas** (T_{MR}, *Mistake Recurrence Time*) - intervalo de tempo entre duas falsas suspeitas de falhas consecutivas (ver Figura 2.18).
  - **Duração das falsas suspeitas** (T_M, *Mistake duration*) - intervalo de tempo que o detector leva para corrigir uma falsa suspeita de falha (ver Figura 2.18).

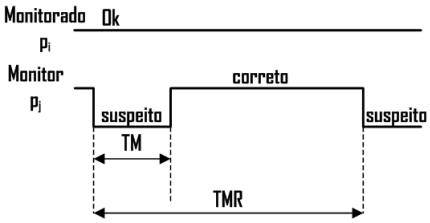


Figura 2.18. Duração da falsa suspeita (TM) e Intervalo entre falsas suspeitas (TMR)

- **Métricas Secundárias**, obtidas a partir das métricas primárias:

  - **Taxa de falsas suspeitas** (λ_M, *Average Mistake Rate*) – indica a taxa de falsas suspeitas de falhas cometidas pelo detector.
  - **Probabilidade de consulta correta** (P_A, *Query Accuracy Probability*) – indica a probabilidade do detector produzir a saída correta em um instante qualquer.

- **Período bom** (T_G, *Good Period Duration*) – indica o intervalo médio de tempo em que o detector de defeitos produz a saída correta.
- **Intervalos entre períodos bons** (T_{FG}, *Foward Good Period Duration*) – indica o intervalo médio de tempo entre dois períodos bons T_G.

Discutindo a relação entre as métricas. Sejam $Pr(A)$, $E(x)$, $E(xk)$, $V(x)$ a probabilidade de um evento A ocorrer, o valor esperado de x, o momento k e a variância de x, respectivamente, as relações entre as métricas podem ser mapeadas com base no seguinte teorema (Chen; Toueg; Aguilera, 2002):

Teorema 2.1 (Relação entre as métricas de *QoS*) *Para qualquer detector de defeitos ergótico*[13], *os seguintes resultados se mantém:*

i) $T_G = T_{MR} - T_M$.

ii) *Se* $0 < E(T_{MR}) < \infty$, *então* $\sigma_M = \frac{1}{E(T_{MR})}$ *e* $P_A = \frac{E(T_G)}{E(T_{MR})}$.

iii) *Se* $0 < E(T_{MR}) < \infty$ *e* $E(T_G) = 0$, *então* T_{FG} *é sempre* 0. *Se* $0 < E(T_{MR}) < \infty$ *e* $E(T_G) \neq 0$, *então:*

a. $\forall x \in [0,\infty)$, $Pr(T_{FG} \leq x)$, $\frac{1}{E(T_G)} \int_0^x Pr(T_G > y)dy$.

b. $E(T_G k) = \frac{E(T_G k+1)}{[(k+1)E(T_G)]}$, *em particular* $E(T_{FG}) = [\frac{1+V(T_G)}{E(T_G)^2}]E(T_G)/2$.

Com base no Teorema 2.1, algumas conclusões podem ser obtidas, entre as quais:

- Os bons períodos acontecem nos intervalos em que o detector não comete falsas suspeitas;
- A taxa de falsas suspeitas cometidas pelo detector de defeitos representa o número de falsas suspeitas por unidade de tempo.
- A probabilidade do detector produzir um valor correto é a relação entre o valor esperado para períodos bons e o valor esperado para intervalo entre falsas suspeitas.
- Se existe a possibilidade do detector não cometer erros, então o valor esperado para o próximo período bom será a relação entre o somatório das probabilidades de todos os possíveis intervalos de períodos bons e valor esperado para o período bom.

Sintonia de detectores de defeitos com QoS. Chen, Toueg e Aguilera (2002) propõem um processo de sintonia (*offline*) para determinar, através do comportamento probabilístico do canal

[13] Um detector é ergótico se, em rodadas livres de defeitos, tal detector lentamente *esquece* o histórico de suas saídas. Desta forma, seu comportamento depende apenas de suas saídas mais recentes.

de comunicação, o período de monitoramento (δ) do detector e a margem de segurança (α) – que compõe o *timeout* e é usada para evitar que atrasos extras levem o detector a cometer falsas suspeitas.

A sintonia do detector de defeitos está sujeita ao nível de qualidade de serviço desejado. Assim, deve-se definir uma tupla (T_D^U, T_{MR}^L, T_M^U), representando, respectivamente, o tempo máximo de detecção, o intervalo mínimo entre falsas suspeitas recorrentes e o tempo máximo para duração das falsa suspeitas [14]. Esta tupla especifica a *QoS* desejada para o detector e as seguintes relações devem ser atendidas:

$$T_D < T_D^U \tag{2.1}$$

$$E(T_{MR}) \geq T_{MR}^L \tag{2.2}$$

$$E(T_M) \leq T_M^U \tag{2.3}$$

O processo de sintonia é um problema de pesquisa operacional, no qual se deve encontrar o máximo δ sujeito às restrições 2.1, 2.2 e 2.3. Para tanto, de posse de δ e T_D^U, pode-se encontrar uma margem de segurança inicial α_0, através do procedimento de configuração a seguir.

(a) Calcular

$$q_0^{'} = (1 - p_L)Pr(D < T_D^U) \tag{2.4}$$

em que p_L e $q_0^{'}$ representam, respectivamente, a probabilidade de perda de mensagens e a probabilidade de uma mensagem de monitoramento m_k chegar dentro do intervalo $[k, k+1)$. Com isso, é possível calcular o período de monitoramento máximo, usando:

$$\delta_{max} = q_0^{'} T_M^U \tag{2.5}$$

Se $\delta_{max} = 0$, então a *QoS* desejada não pode ser encontrada, senão o passo **(b)** deve ser realizado.

(b) Encontrar o maior $\delta < \delta_{max}$ tal que $f(\delta) \geq T_{MR}^L$, em que:

$$f(\delta) = \delta \left\{ q_0^{'} \prod_{j=1}^{\left\lceil \frac{T_D^U}{\delta} - 1 \right\rceil} \left[p_L + (1 + p_L) Pr(D > T_D^U - j\delta) \right] \right\}^{-1} \tag{2.6}$$

[14] A notação T_D^U, T_M^U e T_{MR}^L é usada em Chen, Toueg e Aguilera (2002), em que U e L representam os limites superior (*upper*) e inferior (*lower*), respectivamente. No entanto, no Capítulo 4, desta Tese, usa-se este mesmo padrão de notação com algumas diferenças (TD, TM e TMR ao invés de T_D, T_M e T_{MR}, como originalmente proposto em Chen, Toueg e Aguilera (2002)).

(c) Por fim, encontra-se $\alpha_0 = T_D^U - \delta$

2.4.1.4 Detectores adaptativos de defeitos

Mediante a natureza arbitrária dos atrasos de processamento e de comunicação nos sistemas assíncronos, a definição do período de monitoramento e do *timeout* de detecção representa um desafio. O período de monitoramento impacta no custo computacional e na velocidade da detecção, enquanto que o *timeout* de detecção impacta na velocidade e na precisão do detector. Muitos trabalhos têm desconsiderado as questões relacionadas ao custo e focado no aspecto relacionado à provisão de uma detecção mais precisa e com um menor tempo de detecção. Para tanto, esses trabalhos têm considerado estratégias adaptativas para o ajuste de *timeout* de detecção que atendam a tais requisitos de velocidade e precisão.

Essa adaptação se apoia na mesma premissa de Chandra e Toueg (1996) de que sistemas reais (em modelos assíncronos) experimentam períodos de estabilidade (i.e. o GST). Mais ainda, a maioria das estratégias de adaptação, usadas na implementação dos detectores adaptativos de defeitos, consideram que, a partir de alguma informação a respeito do ambiente[15] ou das mensagens de monitoramento do serviço de detecção[16], é possível realizar estimativas a respeito dos atrasos e prover *timeouts* de detecção adequados – reduzindo o número de falsas suspeitas de falhas e aumentando a velocidade de detecção de defeitos. Evidentemente, essas hipóteses são mais restritivas que as observadas nos detectores de Chandra e Toueg (1996), entretanto, são hipóteses muitas vezes verificadas na prática – ver, por exemplo, Jain e Routhier (1986), Afanasyev et al. (2010) etc.

Em termos de implementação, a adaptabilidade do detector de defeitos consiste em estimar *timeouts*, considerando alguma estimador de atrasos em conjunto com uma margem de segurança (α) – definida estática ou dinamicamente e cujo papel é evitar que valores de atraso subestimados leve o detector a cometer falsas suspeitas.

Dentre as diversas abordagens de adaptação de *timeout* de detecção, discute-se, a seguir, as abordagens de Jacobson (1988) e de Bertier, Marin e Sens (2002), as quais são usadas, mais adiante, durante a avaliação de desempenho da proposta de detecção autonômica de defeitos apresentada no Capítulo 4.

A abordagem de adaptação de Jacobson (1988). No protocolo de comunicação TCP[17], a cada mensagem enviada, o transmissor estima um intervalo de tempo para a chegada de uma mensagem de confirmação de recebimento (*acknowledgment*) oriunda do receptor. Caso a confirmação não chegue no intervalo de tempo estimado, o transmissor retransmite da mensagem.

Jacobson (1988) sugere um estimador de atrasos para evitar que retransmissões desneces-

[15] ver, por exemplo, Macêdo e Lima (2004), Wiesmann, Urban e Defago (2006) etc.

[16] ver, por exemplo, Chen, Toueg e Aguilera (2002), Bertier, Marin e Sens (2002), Nunes e Jansch-Pôrto (2004), Falai e Bondavalli (2005) etc.

[17] *Transport Control Protocol*, ver (Tanenbaum, 2003)

sárias de mensagens colaborem para o congestionamento em redes TCP. Este estimador é apresentado no Algoritmo 2.1.

Algoritmo 2.1: *Estimador de Jacobson (1988)*

input *: the measured (or observed) value (i.e. obs)*
output*: the estimated value, considering a safety margin*
1 $error_k = obs_k - est_k$;
2 $est_{k+1} = est_k + c * error_k$;
3 $var_{k+1} = var_k + c * (|error_k| - var_k)$;
4 $output_{k+1} = a * est_{k+1} + b * var_{k+1}$;

Para realizar uma estimativa, o estimador de Jacobson (1988), primeiramente obtém o desvio (*error*) entre os últimos valores observado (obs_k) e estimado (est_k), ver Linha 1. Em seguida, usando um fator de confiança (c) no erro calculado, estima-se o próximo valor (est_{k+1}), ver Linha 2. Então, o algoritmo realiza uma estimativa das variações do valor observado (var_{k+1}), Linha 2. Por fim, o algortimo retorna a soma da nova estimativa (est_{k+1}) com sua variação (var_{k+1}), considerando as margens de segurança a e b, respectivamente (ver Linha 4).

O ajuste de *timeout* considerando o estimador de Jacobson (1988) é realizado de acordo com o Algoritmo 2.2 – em que primeiramente o atual atraso (*delay*) é medido e, então, repassado para o estimador de Jacobson (1988), ver Linhas 1 – 2.

Algoritmo 2.2: *Adaptação de timeout usando a abordagem de Jacobson (1988)*

1 ***measure*** $delay_k$;
2 $timeout_{k+1} =$ ***jacobson***$(delay_k)$;

No TCP, os parâmetros a, b e c do estimador são definidos como $\frac{1}{10}$, 1 e 2, respectivamente.

A abordagem de adaptação de Bertier, Marin e Sens (2002). Na estratégia de adaptação de *timeout* de Bertier, Marin e Sens (2002), ver Algoritmo 2.3, o atraso é estimado usando a média dos últimos atrasos observados[18], ver Linha 1. Então, uma margem de segurança adaptativa é calculada usando o algoritmo de Jacobson (1988), ver Linha 3. Note que, o cálculo desta margem de segurança é realizado em função dos desvios entre o atrasos estimados e observados (Linha 2) – esse desvio é usado como entrada para o estimador de Jacobson (1988). Além disso, uma vez que, o atraso estimado representa o atraso médio, o desvio entre os valores estimado e observado é uma estimativa para a variação dos atrasos. Por fim, o *timeout* é obtido a partir da soma do atraso estimado (*est*) com margem de segurança (α) calculada, ver Linha 4.

Na abordagem de Bertier, Marin e Sens (2002), o parâmetro *ws* é igual a 1000.

[18] Seguindo a estratégia proposta por Chen, Toueg e Aguilera (2002).

Algoritmo 2.3: *Adaptação de timeout usanda a abordagem de Bertier, Marin e Sens (2002)*

1 $est_{k+1} = \frac{1}{ws} * \sum_{i=k-ws}^{k} obs_k$;

2 $error_k = obs_k - est_k$;

3 $\alpha_{k+1} = jacobson(error_k)$;

4 $timeout_{k+1} = est_{k+1} + \alpha_{k+1}$;

2.5 GERENCIAMENTO DE SISTEMAS DISTRIBUÍDOS

Em sistemas computacionais, o gerenciamento é uma disciplina que se manifesta de forma recorrente (i.e. transversalmente) em diferentes aspectos dos sistemas e visa assegurar a operação eficiente e eficaz dos sistemas computacionais e de seus recursos, de acordo com os objetivos do negócio para o qual tais sistemas foram projetados. Nesse sentido, o gerenciamento é uma atividade importante para garantir que os requisitos funcionais e não funcionais serão mantidos durante a vida útil do sistema.

Sendo o gerenciamento uma disciplina transversal, a gestão de sistemas distribuídos deve ser observada em um contexto integrado, no qual o gerenciamento dos componentes e mecanismos no nível do sistema distribuído supre às demandas de níveis superiores (i.e. mais próximos no negócio) e determina requisitos que devem ser atendidos pelos níveis inferiores de gestão (i.e. mais próximos das infra-estruturas de execução subjacentes).

Entretanto, independente do nível no qual a disciplina de gerenciamento é realizada, a implementação da mesma deve considerar, não apenas um ciclo de atividades básicas, mas também funções de gerenciamento relacionadas aos requisitos básicos para a garantia da efetividade e da eficiência dos sistemas computacionais.

Mais recentemente, novas facilidades de gerenciamento, ditas autonômicas, vem sendo usadas para suportar a complexidade de gerenciamento dos ambientes de computação modernos com características e requisitos dinâmicos.

Todas essas questões são discutidas no restante desta seção. Contudo, é realizada uma discussão mais aprofundada das questões relacionadas à gestão autonômica – um dos conceitos base do trabalho desenvolvido nesta Tese.

2.5.1 Atividades básicas do gerenciamento

A gestão de sistemas computacionais envolve atividades de planejamento e intervenções para adaptar o sistema às possíveis mudanças nas condições que foram inicialmente consideradas durante o projeto, exemplo: aumento na demanda de serviço, mudanças nos requisitos de negócio, falhas de componentes, ocorrência de ações maliciosas ao funcionamento do serviço ou aos dados relacionados etc.

Para tanto, a implementação do gerenciamento deve considerar (Figura 2.19):

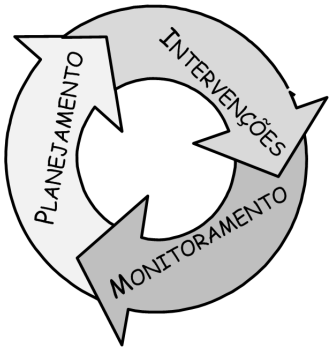


Figura 2.19. Laço de gerenciamento

i) *Monitoramento* continuado dos mecanismos de suporte e das infra-estruturas subjacentes, de modo a identificar as mudanças nas condições de projeto;

ii) *Planejamento* dos ajustes a serem realizados quando mudanças no ambiente computacional ou nos requisitos de negócios são observados;

iii) *Intervenções* nos componentes ou em seus parâmetros de configuração de modo adequar o sistema às mudanças ou aos novos riscos potenciais.

Em sistemas distribuídos, o gerenciamento é apoiado pelo uso de ferramentas e plataformas que automatizam algumas das atividades de gestão – principalmente para a atividade de monitoramento e intervenção (em alguns casos), ver Veríssimo e Rodrigues (2000, Cap. 21–25). Contudo, essas ferramentas e plataformas não consideram procedimentos automatizados e integrados que permitam adaptação dos mecanismos de suporte às aplicações distribuídas face à dinâmica dos requisitos e das políticas de negócio e às mudanças nos ambientes computacionais – o que se apresenta como um desafio para o gerenciamento de sistemas distribuídos modernos, ver Ganek (2007).

2.5.2 Disciplinas do gerenciamento

Para negócios que são apoiados em serviços suportados por sistemas computacionais em ambientes distribuídos, o gerenciamento deve considerar diferentes níveis em visão integrada, conforme pode ser visto na Figura 2.20 – adaptada de Hegering, Abeck e Neumair (1998).

A gestão no nível do negócio enfatiza o processo e as políticas do negócio, observando, por exemplo, o gerenciamento financeiro, de pessoal, tecnológico e de produção etc. – orientando, assim, a determinação das políticas da infra-estrutura de Tecnologia da Informação (TI) e ditando, por exemplo, processos de operação, serviços associados e dados que devem ser considerados na implementação e gerenciamento de tais infra-estruturas.

No nível dos clientes, esse gerenciamento se orienta pelas expectativas dos mesmos em termos de custo, segurança, flexibilidade, efetividade dos serviços fornecidos – determinando, assim, as diretrizes de qualidade do serviço que devem ser perseguidas pelo pessoal de TI no ge-

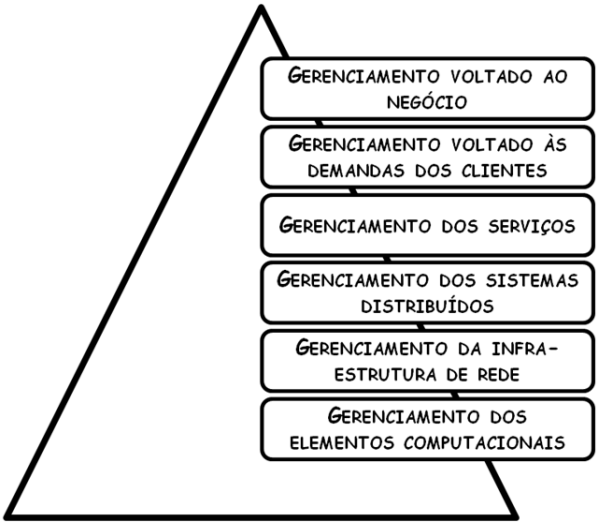


Figura 2.20. Pirâmide de gerenciamento

renciamento dos serviços computacionais oferecidos e que devem conciliar os diferentes aspectos operacionais relacionados ao desempenho, à escalabilidade, à confiabilidade, à segurança etc.

As metas de desempenho atribuídas aos serviços que devem ser entregues pelas aplicações, são mapeadas verticalmente, determinando, assim, novas metas ou requisitos de qualidade de serviço para os componentes e mecanismos de suporte a computação distribuída. No nível da computação distribuída, o gerenciamento observa como estes mecanismos e componentes asseguram o desempenho da colaboração e coordenação distribuída – assim, a gestão foca, por exemplo, na configuração da arquitetura e dos parâmetros operacionais dos componentes distribuídos de modo que aspectos de sincronia, custo computacional dos algoritmos, confiabilidade e consistência das operações sejam combinados de forma adequada para entregar a qualidade de serviço desejada.

Os aspectos demandados pela gestão no nível dos sistemas distribuídos, originam os aspectos do serviço a serem entregues pela infra-estrutura de comunicação. No nível da rede, o gerenciamento é pautado pela observância aos aspectos relacionados à comunicação e aos seus respectivos componentes relacionados. Nesse nível supervisiona-se, por exemplo, a qualidade dos enlaces de dados, o uso eficiente e justo dos canais de comunicação, a alocação de recursos nos sistemas de interconexão e nos sistemas fins, a segurança no encaminhamento das mensagens etc.

O gerenciamento dos elementos computacionais estão mais voltados à questão da administração no nível mais baixo, focando questões relacionadas à adequação das facilidades de processamento e armazenamento, à gestão dos processos e contas de usuários, supervisão de sistemas de arquivos e módulos de software – todos esses aspectos considerando o elemento computacional (i.e. a estação de trabalho, o servidor etc.).

Esse gerenciamento integrado, envolve diferentes especialidades e agrega um alto custo de

operacionalização. Desse modo, na grande maioria dos casos, o gerenciamento é implementado focando apenas um ou outro dos níveis de gerenciamento da pirâmide.

2.5.3 Funções do gerenciamento

A gestão dos sistemas computacionais deve considerar diferentes funções (ou aspectos)[19], descritas abaixo (Hegering; Abeck; Neumair, 1998; Veríssimo; Rodrigues, 2000):

- **Gestão da configuração** (*configuration management*) – é uma função de gestão preocupada com a análise e manutenção da configuração do sistema. Na perspectiva do sistema distribuído, a base da gestão da configuração é implementada a partir dos atributos de configuração disponíveis nos componentes, podendo ser realizada a partir de três classes de atributos: *estado*, o qual se refere ao estado global dos componentes (e.g. em funcionamento ou falho); *status*, se referindo aos detalhes das variáveis internais dos componentes (e.g. processando ou bloqueado); *relação*, se referindo ao relacionamento entre os componentes (e.g. um componente é uma réplica secundária de outra).

- **Gestão da confiabilidade** (*fault management*) – é uma função de gestão preocupada, não apenas, em lidar com procedimentos e mecanismos para a detecção, isolamento e eliminação do estado de erro do sistema, mas também, em permitir uma reação eficiente no caso da ocorrência de uma falha.

- **Gestão do desempenho** (*performance management*) – é uma função de gestão preocupada com o problema de manter o desempenho ou a qualidade de serviço dos sistemas dentro dos limites especificados. Atividades típicas da gestão de desempenho são: a medição dos parâmetros de QoS; o monitoramento para detecção de violações de QoS; verificação de sobrecargas e atrasos; realização de estatísticas e relatórios de produção; medição e planejamento de capacidade etc. É esta função de gestão que deve se preocupar com o mapeamento horizontal[20] e vertical[21] dos parâmetros de qualidade de serviço.

- **Gestão da segurança** (*security management*) – é uma função de gestão preocupada em determinar as políticas e medidas necessárias para proteger os recursos da instituição, incluindo informação, infra-estrutura de TI, serviços e dados de produção. Atividades típicas da gestão de segurança incluem: medidas para detecção e tratamento de intrusões; medidas para autenticação e proteção de usuários e serviços; planos de contingenciamento para o caso de violações das medidas de segurança etc.

[19] Muitas vezes referenciadas por funções FCAPS (i.e. *Fault*, *Configuration, Accounting, Performance and Security*).

[20] Quando um mesmo serviço é disponibilizado por diferentes provedores, esses provedores podem ter interpretações diferenciadas para um mesmo conjunto de métricas de qualidade de serviço. Assim, no mapeamento horizontal, as métricas de qualidade de serviço do sistema são traduzidas considerando os diferentes provedores.

[21] Quando é considerado o mapeamento das métricas entre os diferentes níveis do sistema. Por exemplo, o requisito de tempo de resposta do serviço distribuído pode derivar a latência desejada para os canais de comunicação.

- Gestão do uso de recursos (*accounting management*) – é uma função de gestão focada no relacionamento dos usuários com os sistemas, tendo assim como atividades: a coleta de dados de utilização; a construção de estatísticas de uso; a alocação de recursos e cotas de uso e de custeio etc.

A gestão de falhas e a gestão de desempenho trazem características comuns, uma vez que a ocorrência de falhas compromete o desempenho dos sistemas. Entretanto, a gestão de falhas está preocupada apenas em certificar a continuidade do funcionamento correto do sistema, enquanto que, a gestão de desempenho deve certificar que todos os componentes do sistema (incluindo os mecanismos de tolerância a falhas) operam bem, alcançando o melhor desempenho possível para uma determinada configuração.

Da mesma forma, a gestão de desempenho e a gestão de uso tem algumas similaridades. Entretanto, enquanto a gestão de desempenho está focada no uso em uma perspectiva geral de demanda de serviço, a gestão de uso de recursos, por sua vez, observa e trata as demandas e custeio de cada usuário em particular – lidando, por exemplo, com atividades de *provisionamento* de recursos por demanda de usuário. A gestão de uso de recursos é uma premissa para realização de atividades relacionadas à tarifação (i.e. *bilhetagem*), uma atividade comum em grandes provedores de TI.

2.5.4 Gestão autonômica como base para a implementação do auto-gerenciamento

O crescente avanço das tecnologias de comunicação e computação associado ao desenvolvimento de modernas ferramentas, técnicas e metodologias de construção de software, tem resultado em uma crescente explosão na implementação de aplicações e serviços de informação sobre rede, os quais cobrem diversos aspectos da vida cotidiana (Ganek, 2007). Esses serviços e aplicações vêm se tornando cada vez mais complexos, heterogêneos e dinâmicos (Parashar; Hariri, 2005). O resultado disso, é uma larga infra-estrutura de informação agregando, em geral, um grande número de unidades de processamento distribuídas associadas a um sistema de comunicação e armazenamento de dados, os quais são, por si só, complexos, heterogêneos e dinâmicos. Essa associação resulta em complexidades para o desenvolvimento, configuração e gerenciamento de aplicações, as quais se tornam difíceis de serem mantidas usando as estratégias tradicionais baseadas em requisitos, comportamentos, interações e composições estáticas (Müller et al., 2006). Esses fatores fazem com que as aplicações, os ambientes de programação e a infra-estrutura de informação se tornem ingerenciáveis e inseguras muito rapidamente (Parashar; Hariri, 2005).

Nesse sentido, tornar o sistema adaptável, confiável, seguro e principalmente auto-gerenciável – levando ao usuário do sistema um alto nível de abstração no seu uso e em seu funcionamento, é um desafio para os sistemas computacionais modernos e de larga escala (Parashar; Hariri, 2005). Esse fator tem motivado uma corrida na indústria de Tecnologia da Informação (TI) por mecanismos que permitam a construção de sistemas que possam atender a tais requisitos (Kephart,

2005). A exemplo disso, está a abordagem inovadora de Sistemas Autonômicos (*Autonomic Systems*) da IBM (Horn, 2001) seguida pelas Arquiteturas Adaptativas (*Adaptive Enterprise*) da HP (Mont; Bramhall; Pato, 2003) e pelos Sistemas Dinâmicos (*Dynamic Systems*) da Microsoft (Microsoft, 2005).

O enfoque da abordagem dos sistemas autonômicos é o projeto e a construção de sistemas baseados em estratégias típicas dos sistemas biológicos humanos de modo a superar os desafios de escala, complexidade, heterogeneidade, confiabilidade e incerteza (Parashar; Hariri, 2005; Horn, 2001).

A Computação autonômica (*Autonomic Computing*) é particularmente inspirada no sistema nervoso humano (Horn, 2001; Manoel et al., 2005), objetivando, para tanto, a construção de sistemas e aplicações que, assim como o sistema nervoso humano, possam ser auto-gerenciáveis – i.e. implementando de forma autônoma o laço de gerenciamento: monitorando mudanças e falhas no ambiente, planejando e agindo de modo a garantir as políticas definidas pelos objetivos do negócio no qual se enquadra a aplicação (Diao et al., 2005; Parashar; Hariri, 2005).

2.5.4.1 Atributos de um sistema autonômico

Os sistemas computacionais autonômicos possuem quatro principais atributos (Horn, 2001; Manoel et al., 2005):

- **Auto-configuração** (***Self-configuring***): capacidade de se adaptar a mudanças no sistema, implicando na habilidade de dinamicamente configurar a si mesmo, com intervenção externa mínima e fazendo com que novos componentes sejam instalados ou trocados em caso de mudanças no ambiente de *TI*.

- **Auto-cura** (***Self-healing***): capacidade de se recuperar de erros detectados, implicando na habilidade de detectar falhas durante as operações e iniciar uma ação corretiva, garantindo a continuidade do serviço. Com isso, procura-se diminuir ou eliminar o impacto das falhas de componentes no cumprimento dos objetivos de negócio da aplicação (Avizienis et al., 2004).

- **Auto-otimização** (***Self-optimizing***): capacidade de melhorar a utilização dos recursos, implicando na habilidade de eficientemente maximizar a alocação e utilização dos recursos de acordo com as necessidades dos usuários com um mínimo de intervenção possível.

- **Auto-proteção** (***Self-protecting***): capacidade de antecipar e sanar intrusões, implicando, não apenas, na capacidade de garantir que pessoas autorizadas tenham acesso aos dados que lhes são permitidos, mas também, na habilidade de automaticamente realizar as ações necessárias para tornar o sistema menos vulnerável a ataques à sua infra-estrutura e aos seus dados durante a sua execução.

As propriedades dos sistemas autonômico refletem a automação das funções de gerenciamento dos sistemas computacionais (ver Seção 2.5.3) – em que as interpretações e decisões de projeto, antes realizadas com intervenção humana, agora são mapeadas em requisitos de entrada e usadas para permitir que os sistemas possam suportar mudanças dinâmicas das características dos ambientes computacionais e dos próprios requisitos relacionados ao negócio. Assim, o administrador ou gestor do sistema deve determinar as políticas (gestão no nível do negócio ou voltadas para o cliente – ver Seção 2.5.2), enquanto que os sistemas se preocupam com as atividades de gestão nos níveis mais baixos, focada nos serviços, nos sistemas, na rede etc.

2.5.4.2 Arquitetura de um sistema autonômico

Em um ambiente autonômico, os componentes trabalham juntos, comunicando entre si e com ferramentas de gerenciamento de alto nível (Jacob et al., 2004; Manoel et al., 2005). Nesse ambiente, tais componentes podem gerenciar e controlar a si mesmos e aos demais.

Entretanto, um componente pode gerenciar a si mesmo até um certo limite. De um ponto de vista mais amplo com relação ao sistema, algumas decisões precisam ser tomadas por componentes de mais alto nível. Sendo assim, nessa relação, o componente de alto nível pode ser visto como *componente gestor* (*autonomic manager*), enquanto que o componente de mais baixo nível pode ser visto como o *recurso gerenciado* (*managed resource*) (Parashar; Hariri, 2005).

A arquitetura do *componente gestor* pode ser dividida em quatro partes que compartilham o conhecimento (Jacob et al., 2004):

- ***Monitor*** – provê os mecanismos que coletam, agregam, filtram, gerenciam e relatam detalhes coletados de um elemento.
- ***Analisador*** – provê mecanismos que correlacionam e modelam situações complexas. Esse componente habilita o *gestor autonômico* a aprender sobre o ambiente e ajuda a predizer situações futuras.
- ***Plano*** – provê mecanismos que estruturam as ações necessárias para alcançar os objetivos. Os mecanismos de planejamento usam as políticas para guiar o trabalho a ser realizado.
- ***Executor*** – provê os mecanismos que controlam a execução de um plano, podendo realizar atualizações nos componentes gerenciados em tempo de execução.

A Figura 2.21, adaptada de (Jacob et al., 2004), mostra as associações dos elementos da arquitetura do *componente gestor*. A interação entre *componente gestor* e *recurso gerenciado* compõe um laço, dito laço de controle autonômico.

O *recurso gerenciado* é o objeto controlado do laço de controle. Tal recurso pode ser um simples componente ou uma coleção de componentes. Para o efetivo controle do *recurso gerenciado*, a arquitetura se utiliza de elementos sensores e atuadores (Figura 2.21). Os elementos

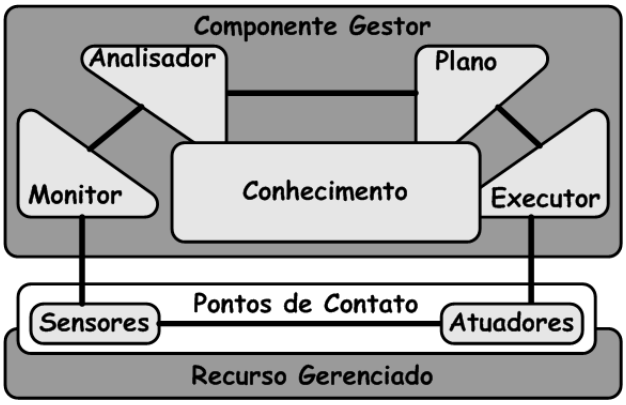


Figura 2.21. Laço de controle autonômico (Horn, 2001)

sensores provêm meios para a coleta de informações sobre o atual estado (configuração) e transições de um componente. Os elementos atuadores, por sua vez, são elementos responsáveis por mudar o estado de um componente.

A arquitetura autonômica não faz exigências em relação à localização de seus componentes (i.e. sensores, *componente gestor*, *recurso gerenciado* e atuador), nem em relação ao mecanismo de comunicação a ser utilizado para a interação entre os mesmos.

2.5.4.3 Níveis de maturidade

Tornar um sistema computacional autonômico não é uma operação instantânea, exige um processo de transição gradual, por novas tecnologias de *hardware* e *software*. Tal transição deve atravessar diversos estágios ou níveis de maturidade (Jacob et al., 2004):

- *Básico* (nível 1) – representa o nível no qual os elementos do sistema são instalados e gerenciados individualmente, requerendo pessoal técnico qualificado que conheça o ambiente e que possa gerenciar, analisar e intervir no sistema.

- *Gerenciado* (nível 2) – representa o nível no qual ferramentas de gerenciamento consolidam informações que são usadas pela equipe de administradores para analisar e intervir sobre o funcionamento do sistema.

- *Preditivo* (nível 3) – um monitor do sistema correlaciona ações e dados de modo a reconhecer padrões de funcionamento e sugerir ações que devem ser aprovadas e iniciadas pelos administradores do sistema.

- *Adaptativo* (nível 4) – um monitor do sistema não apenas monitora, correlaciona e desenvolve planos de ação, mas também decide de acordo com as políticas adotadas.

- *Autonômico* (nível 5) – representa o nível no qual componentes de infraestrutura são integrados e dinamicamente se auto-gerenciam de acordo com as políticas e regras estabelecidas.

Os níveis autonômico e adaptativo são bastante similares. Entretanto, é importante ressaltar que a abordagem autonômica tem como aspecto principal alinhar o gerenciamento dos sistemas computacionais com as políticas e regras do negócio. Nesse sentido, espera-se que um sistema verdadeiramente autonômico possa realizar a adaptação, considerando regras, políticas ou requisitos que podem mudar com a dinâmica dos negócios. Na abordagem adaptativa, por sua vez, essas regras e políticas são assumidas como estáticas, sendo assim, os sistemas adaptativos se ajustam às diversas condições do ambiente, mas assumem que os requisitos determinados durante o projeto não mudam durante a vida útil do sistema.

2.5.5 Gestão autonômica em sistemas distribuídos

Em sistemas distribuídos, a automação das atividades de gerenciamento têm sido focadas na implementação de facilidades de adaptação, i.e. as soluções consideram a adaptação a diferentes condições do ambiente, mas usam políticas ou regras definidas em tempo de projeto – não considerando ou não suportando, portanto, mudanças de tais políticas, regras ou requisitos em tempo de execução. Muitos desses trabalhos têm se auto-denominados autonômicos, mas não consideram o alinhamento dinâmico com os requisitos do negócio – uma demanda básica para a proposição de sistemas autonômicos, como é discutido em Ganek (2007). Como exemplo, podem ser citados Mills et al. (2004), Sivasubramanian et al. (2005), Claudel et al. (2006) e Satzger (2008) etc. Entretanto, esse aspecto não inválida a contribuição desses trabalho, na perspectiva da adaptação, com relação a automação das atividades de gerenciamento de aspectos de sistemas distribuídos.

Na última década, diversos trabalhos têm proposto a implementação das diferentes funções (ou propriedades) da gestão autonômica. De um modo geral, a maioria desses trabalhos focam na relação de compromisso entre custo e benefícios em termos de um ou mais requisitos de qualidade de serviço – principalmente quando variações de carga ou nos próprios requisitos de qualidade de serviço precisam ser consideradas.

Menascé, Barbará e Dodge (2001), por exemplo, propõem um *framework* para a manutenção da QoS de aplicações voltadas para o comércio eletrônico. Esse framework combina uma heurística para otimização e modelos de predição baseados em redes de filas para ajustar dinamicamente os parâmetros de configuração (e.g. número máximo de conexões, políticas e parâmetros para balanceamento de carga etc.) de modo a permitir que os requisitos de QoS sejam atendidos quando existem variações da carga computacional relacionada à demanda de serviço.

Diao, Hellerstein e Parekh (2002), por sua vez, usam um controlador *nebuloso*[22] para regular parâmetros operacionais de servidores web (e.g. número máximo de conexões) de modo a permitir uma melhoria do tempo de resposta quando condições dinâmicas de carga são consideradas.

[22] tradução do inglês, *fuzzy controller*.

Armstrong et al. (2003) propõem um Sistema de Defesa Autonômico (ADS, *Autonomic Defense System*) com o objetivo de atender a relação de compromisso entre o risco potencial da ameaça de infecção de componentes do sistema distribuído versus a precisão e a velocidade do diagnóstico de tais ameaças – combinando, para tanto, algoritmos de otimização e controle realimentado baseado processos de decisão de *Markov*.

Menascé e Bennani (2006) propõem uma abordagem para alocação dinâmica de recursos em ambientes virtualizados, considerando condições variadas de carga computacional. Tal abordagem usa um gestor autonômico para transferir dinamicamente recursos de CPU entre diferentes máquinas virtuais de forma que otimize uma função de custo global definida para o ambiente virtualizado.

Abdelwahed e Kandasamy (2007) propõem uma abordagem para a gestão autonômica do consumo de energia, sendo a mesma implementada usando dois laços de controle: no primeiro nível (i.e. nível mais baixo) um laço de controle realimentado combinado com filtros preditivos são usados para otimizar o desempenho em termos de consumo de energia quando a carga computacional varia; no segundo nível, um controlador global, gerencia as interações entre os componentes de modo a determinar e regular os requisitos a serem usados para os controladores distribuídos no nível mais baixo.

Bezerra e Martins (2009) propõem um modelo baseado em políticas de gestão e com habilidades autonômicas capaz de gerir infra-estruturas de redes, observando para tanto o desempenho em termos de métricas de qualidade. Para tanto, esse modelo permite que os administradores da rede possam introduzir políticas em uma base de conhecimento, de modo a permitir que o gestor autonômico possa determinar de forma cíclica e dinâmica novos planos de decisão.

2.6 CONSIDERAÇÕES FINAIS

A implementação de sistemas distribuídos combina diferentes perspectivas, buscando o atendimento dos requisitos funcionais e não funcionais demandados pelos usuários e pelos objetivos do negócio. Para tanto, essas perspectivas envolvem, entre outras coisas, modelos de sincronia, modelos de arquitetura, além de mecanismos de suporte básicos.

Quando a confiabilidade se apresenta como um requisito básico, o projeto dos sistemas distribuídos deve considerar o uso de diferentes técnicas e mecanismos de tolerância a falhas que assegurem o contingenciamento na presença de falhas. Essas técnicas e mecanismos agregam um custo adicional a implementação e manutenção do sistema que devem ser considerados para permitir o balanceamento adequado entre a confiabilidade e os demais requisitos.

Além disso, durante a vida útil, atividades de gerenciamento devem ser executadas sobre os diversos componentes e mecanismos do sistema distribuído de modo a permitir um funcionamento eficiente e eficaz mediante às mudanças nas características do ambiente ou à introdução de novos requisitos (ou mudanças em requisitos antigos).

Na medida em que as tecnologias de comunicação e processamento evoluem, surgem novas

aplicações e ambientes distribuídos, os quais apesar de trazerem novas facilidades para os negócios, tornam esses ambientes e aplicações mais dinâmicos e complexos. Neste contexto, os modelos de sistemas distribuídos tradicionais baseados em composições estáticas e homogêneas (em termos de sincronia) não capturam todas as nuances dos ambientes e aplicações distribuídas modernas. Além disso, as abordagens de gerenciamento baseadas em procedimentos manuais são completamente ineficientes e ineficazes. Por conta disto, novos modelos de sistemas distribuídos e novas abordagens de gestão são propostos pelos pesquisadores da área, com o intuito de lidar com tal dinamicidade e complexidade. A exemplo disso, estão o modelo de sincronia híbridos e adaptativos de Gorender, Mâcedo e Raynal (2005) e as abordagens de gestão autonômica de Horn (2001) – enquanto os modelos híbridos e adaptativos são usados como base para a implementação de mecanismos de tolerância a falhas com resiliência adaptativa, as abordagens de gestão autonômica capacitam os sistemas com a habilidade de auto-gestão, de modo a alinhar a dinamicidade do ambiente com os requisitos do negócio.

A implementação plena da gestão autonômica em sistemas distribuídos ainda é uma área relativamente nova e os diferentes trabalhos propostos na literatura representam esforços combinados para a implementação de gestão autonômica observando diferentes aspectos dos sistemas distribuídos.

Todas essas questões, em maior ou menor profundidade, foram foco das discussões apresentadas neste capítulo. Os conceitos básicos relacionados a implementação e ao gerenciamento em sistemas distribuídos foram apresentados. os temas abordados foram orientados de modo a possibilitar um entendimento a cerca das definições e questões básicas que são discutidas no restante desta Tese. Sendo assim, as discussões apresentadas não esgotam os diferentes aspectos que podem ser abordados em sistemas distribuídos.

Por fim, no próximo capítulo é apresentado os conceitos básicos da teoria de controle, a qual vem sendo usada como uma ferramenta para a construção de mecanismos de gestão (autonômica) em sistemas distribuídos.

Capítulo 3

Neste Capítulo são discutidos alguns dos conceitos básicos relacionados à teoria de controle realimentado, focando, em representações de sistemas discretos, projeto de sistemas de controle discretos e controle adaptativo.

TEORIA DE CONTROLE DE SISTEMAS DINÂMICOS

3.1 INTRODUÇÃO

Teoria de controle é uma disciplina usada para lidar com o comportamento de sistemas dinâmicos – i.e. sistemas cujo comportamento ao longo do tempo é determinado não apenas pelo estado atual, mas também pela relação deste com os estados anteriores (Morrison, 2008). Para tanto, esta disciplina disponibiliza um conjunto de técnicas de análise e síntese que permitem a construção de modelos, os quais não apenas facilitam o entendimento da dinâmica dos sistemas, mas também possibilitam a construção de estratégias de controle que levem tais sistemas a apresentar um comportamento esperado (Ogata, 2009).

No campo da engenharia, a teoria de controle vem sendo usada para construir sistemas automatizados, ditos sistemas de controle, cujo objetivo é realizar intervenções (ou ações de controle), em um ou mais dispositivos (ditos plantas), de modo a, por exemplo: i) *regular a saída do sistema* para manter a mesma próxima a um valor de referência (*set-point*); ii) *rejeitar distúrbios externos* para certificar que certas perturbações (i.e. interferências) inesperadas não comprometam o funcionamento esperado; iii) *otimizar o sistema* para permitir que o mesmo apresente o melhor desempenho possível. Nesse contexto, vários exemplos podem ser encontrados em, por exemplo, Ogata (1995), Ogata (2009), Astrom e Wittenmark (1995), Goodwin, G. e Salgado (2001), entre outros.

Nas últimas décadas, vários trabalhos têm usado a teoria de controle na resolução de diferentes problemas relacionados à implementação de sistemas computacionais, especialmente em áreas como: Redes de computadores – e.g. Ren, Lin e Wei (2005) e Keshav (1991); Sistemas operacionais – e.g. Carneiro, Sá e Barreto (2009); *Middleware* – e.g. Menascé, Barbará e Dodge (2001), Diao, Hellerstein e Parekh (2002) e Menascé e Bennani (2006); Gerenciamento de energia – e.g. Abdelwahed e Kandasamy (2007) –, entre outros. Uma discussão mais detalhada sobre o uso de teoria de controle na implementação de sistemas computacionais pode ser encontrada em Hellerstein et al. (2004).

Com a crescente demanda por mecanismos relacionados à implementação da habilidade de auto-gerenciamento em sistemas computacionais, cada vez mais, a teoria de controle têm sido foco de estudos voltados ao desenvolvimento de sistemas autonômicos. Por conta disto, neste capítulo são apresentados alguns dos principais aspectos da teoria de controle. Assim, na Seção 3.2 são apresentados alguns conceitos básicos sobre sistemas dinâmicos. Na Seção 3.3, discute-se a modelagem de sistemas de controle, envolvendo, para tanto, as principais representações utilizadas e a modelagem de sistemas de controle realimentados. Na Seção 3.4, são abordadas as questões de projetos de sistemas de controle, focando em requisitos de qualidade e estratégias de controle. Por fim, na Seção 3.5, são apresentadas alguns considerações finais.

3.2 CONCEITOS BÁSICOS

Fundamentalmente, a técnica básica usada na teoria de controle é a implementação de laços (ou malhas) de controle, em que as saídas do sistema dinâmico são continuamente monitoradas e usadas por alguma estratégia (dita lei de controle) para determinar valores de entrada que conduzam o sistema a apresentar as saídas desejadas, mesmo que tais sistemas estão sujeitas a certas perturbações (ver Figura 3.1).

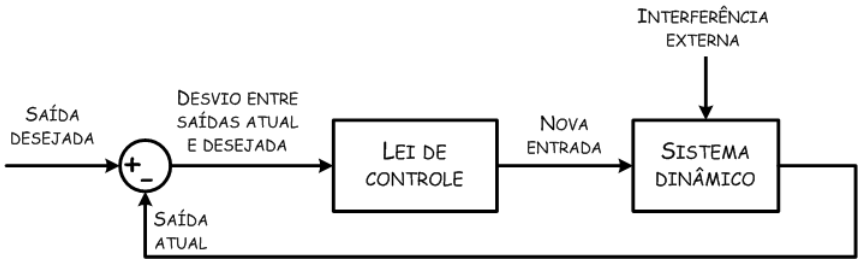


Figura 3.1. Diagrama geral de um Laço de controle

Suponha, por exemplo, um sistema automatizado de aquecimento, cujo objetivo é manter a temperatura de um fluído, em um determinado recipiente, de acordo com a temperatura de referência especificada pelo usuário. Para tanto, um regulador continuamente monitora, através de um termômetro, a temperatura do fluído e compara a mesma com a especificada. Existindo divergências, o regulador aumenta (ou diminui) a potência do aquecedor, de modo a permitir que a temperatura do fluído venha a convergir para a temperatura desejada. Note que, este sistema é dinâmico porque a temperatura do fluído depende não apenas da quantidade de calor imposta pelo aquecedor (entrada), mas também de alguma regra (i.e. função matemática ou dinâmica) que determina como a temperatura do fluído (ou estado) evoluirá ao longo do tempo a partir da temperatura atual.

A partir deste exemplo, é possível identificar os principais componentes do laço de controle (Figura 3.2), isto é: *planta*, *sensor*, *controlador* e *atuador*.

A *planta* é o objeto controlado, no caso do exemplo, o sistema composto pelo aquecedor, recipiente e fluído. A temperatura do fluído é a *variável observada*, monitorada pelo termômetro,

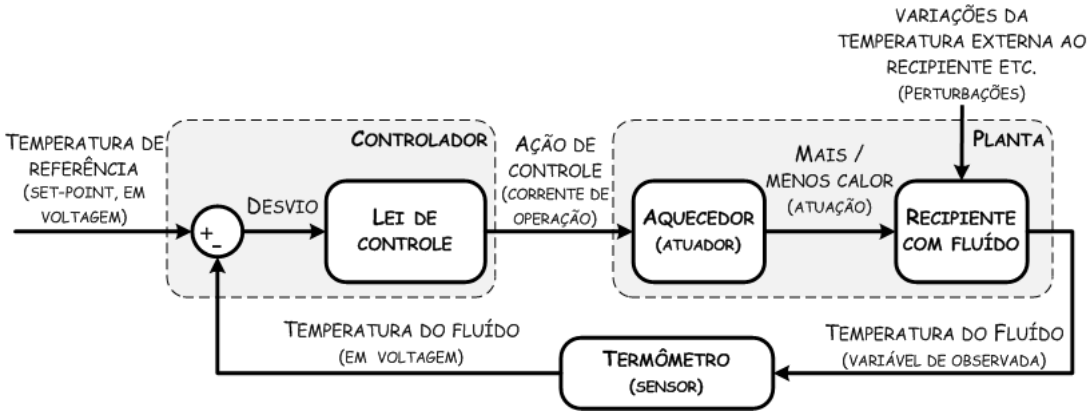


Figura 3.2. Laço (ou malha) de controle do sistema automatizado de aquecimento

o qual faz o papel do dispositivo *sensor*. O *sensor* é responsável por continuamente monitorar e repassar as variáveis observadas para o regulador. O regulador faz o papel do dispositivo *controlador*, o qual tem por responsabilidade calcular o desvio entre os valores desejados e obtidos e então executar a ação de controle – isto é definir a potência do aquecedor. Neste caso, a potência do aquecedor representa a *variável manipulada*. O aquecedor, por sua vez, faz o papel do dispositivo *atuador*, i.e. o dispositivo capaz de traduzir ação de controle em uma atuação efetiva sobre a planta (e.g. aumentar ou reduzir a temperatura), ver Figura 3.2.

Note que, apesar de a temperatura ser a variável observada, é possível que o aquecedor seja implementado a partir de um circuito eletrônico, o qual provavelmente adquire o valor da temperatura a partir de alguma informação em termos da magnitude de um valor de corrente ou de tensão. Assim, o termômetro deve observar a temperatura e traduzi-la para um valor de corrente elétrica ou tensão, de modo que a mesma possa ser manipulada pelo regulador. Esse processo de transformação entre grandezas físicas é denominado *transdução*.

Observe que existem algumas dificuldades durante o processo de regulação da temperatura. Variações na temperatura externa ao recipiente podem influenciar a temperatura do fluído, fazendo com que o mesmo desvie do valor desejado. Essas variações são denominadas *perturbações* ou *distúrbios*. Por fim, a temperatura de referência é dita *ponto de operação* ou *set-point* – i.e. a saída desejada para a planta.

3.2.1 Sinais e sistemas

No contexto da engenharia de controle, os sistemas dinâmicos são sistemas físicos (e.g. motor, caldeira, processo químico etc.) cujas entradas e saídas são representadas por sinais. Um sinal é uma função que representa uma quantidade ou variável física (e.g. temperatura, pressão, velocidade etc.) e contém informações sobre o comportamento ou natureza de algum fenômeno físico associado ao sistema (Hsu, 2004).

Um sinal é dito contínuo quando é definido sobre uma faixa contínua de tempo, caso contrário o mesmo é dito sinal discreto. Sinais discretos podem ser construídos a partir de sinais

contínuos, usando um processo de amostragem, no qual a cada período fixo de tempo T (dito período de amostragem), uma amostra do sinal contínuo é obtida, isto é:

$$x_d(k) = x_c(k * T), \forall k = 0, 1, 2, ..., n \tag{3.1}$$

em que x_c e x_d são, respectivamente, um sinal contínuo e sua versão discreta, obtida a partir do processo de amostragem.

Com isso, é possível classificar os sistemas em relação ao tempo em: a) sistemas de tempo contínuo, ou simplesmente *sistema contínuo* – quando observa e manipula apenas sinais (ou variáveis) contínuos; b) sistemas de tempo discreto, ou simplesmente *sistema discreto* – quando observa e manipula apenas sinais discretos; ou c) *sistemas híbridos*, caso contrário.

Note que os sistemas computacionais são orientados a eventos, disparados por ações internas ou externas. Sendo assim, naturalmente, os modelos de controle realimentados para sistemas computacionais são sistemas discretos (Hellerstein et al., 2004). Nesse sentido, o restante das discussões deste capítulo serão orientadas para esse tipo de modelo.

Outro aspecto importante no estudo dos sistemas é determinar a natureza das relações entre as variáveis do sistema. Sistemas físicos são, em sua grande maioria, não lineares. Um sistema é dito não linear se o relacionamento entre suas variáveis é não linear. Todavia, em muitos casos, tais sistemas podem trabalhar dentro de uma região na qual apresentam um comportamento aproximadamente linear – o que é suficiente para resolução de muitos problemas na prática.

3.3 MODELAGEM DO COMPORTAMENTO DINÂMICO

Na teoria de controle, a análise do comportamento dinâmico se baseia em um procedimento de modelagem, o qual usa representações matemáticas que permitem observar o sistema como uma caixa preta, abstraindo detalhes internos pertinentes à construção dos sistemas e focando apenas no relacionamento entre variáveis (ou sinais) de entrada e de saída. Por estarem relacionadas à modelagem da dinâmica dos sistemas, essas representações matemáticas podem, naturalmente, definir o comportamento das saídas em função das entradas no domínio do tempo (através de equações diferenciais), ou fazer uso de estratégias alternativas (i.e. transformadas), as quais realizam o mapeamento dessas representações temporais para o domínio da freqüência (através de alguma transformação envolvendo números complexos).

No domínio do tempo (discreto), a modelagem do sistema considera o uso de equações de diferenças, na forma:

$$\sum_{i=0}^{n} \alpha_i y[n-i] = \sum_{j=0}^{m} \beta_j u[n-j] \tag{3.2}$$

em que α e β são parâmetros do modelo, enquanto que $y[n]$ representa a n-ésima amostra da variável de saída da planta e $u[n]$ é a n-ésima amostra da variável de entrada.

O tamanho do histórico de entradas ou de saídas necessário para definir o comportamento

de sistema é dito ordem. Assim, na Equação 3.2, a ordem do sistema será n, se $n \geq m$; e m, caso contrário.

Se os parâmetros do modelo (i.e. α e β) variam ao longo do tempo, o sistema é dito *Linear e Variante no Tempo* (LTV, *Linear Time-Varying system*) – caso contrário, o sistema é dito *Linear e Invariante no Tempo* (LTI, *Linear Time Invariant system*).

Em muitos casos, as equações no domínio do tempo são de difícil resolução, sendo relativamente trabalhoso analisar o comportamento do sistema. Por isso, na grande maioria dos casos, os sistemas são analisados no domínio da freqüência.

A transformada $\mathcal{Z}$ (Ogata, 1995) é uma ferramenta bastante utilizada para, a partir de um modelo de sistema no domínio do tempo, encontrar o modelo equivalente no domínio da freqüência. Assim, seja $x[n]$ um sinal discreto, definido para todos $n \geq 0$, então a transformada $\mathcal{Z}$ de x é definida por:

$$X(z) = \mathcal{Z}\{x[n]\} = \sum_{n=0}^{\infty} x[n]z^{-n} \tag{3.3}$$

em que $z = A(\cos\theta + j\sin\theta)$ é um número complexo e, por sua vez, A é o módulo (ou magnitude) de z, j representa a unidade imaginária e θ é o argumento complexo (i.e. ângulo ou fase) em radianos.

Com isso, a Equação 3.2 pode ser reescrita para originar um modelo equivalente em uma representação polinomial em z definida por:

$$\sum_{i=0}^{n} \alpha_i z^{-i} Y(z) = \sum_{j=0}^{m} \beta_j z^{-j} U(z) \tag{3.4}$$

em que $U(z)$ e $Y(z)$ representam a entrada e a saída do sistema no domínio da freqüência.

Usando a transformada $\mathcal{Z}$, também é possível descrever o sistema através de funções de transferência – as quais descrevem como uma entrada $U(z)$ é transformada em (ou está relacionada com) uma saída $Y(z)$, isto é:

$$G(z) = \frac{Y(z)}{U(z)} = \frac{\sum_{j=0}^{m} \beta_j z^{-j}}{\sum_{i=0}^{n} \alpha_i z^{-i}} \tag{3.5}$$

em que $G(z)$ representa a função de transferência.

O diagrama de blocos da Figura 3.3 representa a função de transferência $G(z)$.

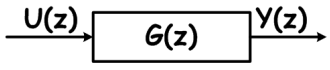


Figura 3.3. Diagrama em bloco da função de transferência $G(z) = \frac{Y(z)}{U(z)}$

A função de transferência G pode ser reescrita na seguinte forma:

$$G(z) = \gamma * \frac{\prod_{j=1}^{m}(z - q_j)}{\prod_{i=1}^{n}(z - p_i)} \tag{3.6}$$

Reescrita dessa forma, $G(z)$ ressalta duas características importantes: pólos e zeros. Os zeros são os valores que tornam $G(z) = 0$ – isto é as raízes de seu numerador, representados na Equação 3.6 por q. Os pólos são os valores que tornam $G(z) = \infty$ – isto é as raízes do denominador, representados na Equação 3.6 por p.

3.3.1 Modelagem de sistemas de controle realimentados

Em sistemas de controle realimentados, a ação de controle é gerada com base no saída atual da planta – i.e. a saída atual é usada para determinar a próxima saída. Desse modo, o sistema de controle pode ser representado por uma malha fechada em blocos com realimentação, conforme indicado na Figura 3.4, em que $P(z)$ e $C(z)$ representam, respectivamente, as funções de transferências da planta e do controlador, enquanto que $U(z)$ e $E(z)$ representam a ação de controle e o desvio entre o *set-point* $R(z)$ e a saída da planta $Y(z)$, respectivamente.

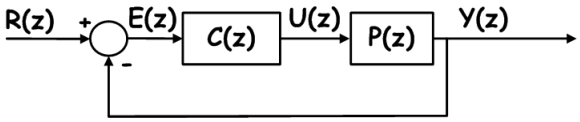


Figura 3.4. Sistema de controle realimentado

A função de transferência em malha fechada $G_F(z)$, a qual determina a relação entre o sinal de referência $R(z)$ e a saída da planta $Y(z)$ é definida por[1]:

$$G_F(z) = \frac{Y(z)}{R(z)} = \frac{P(z).C(z)}{1 + P(z).C(z)} \tag{3.7}$$

As características de desempenho da malha de controle (e.g. estabilidade, precisão etc.) podem ser obtidas a partir de G_F – conforme é discutido mais adiante.

3.4 PROJETO DE SISTEMAS DE CONTROLE

Nesta seção são discutidos os principais aspectos relacionados à implementação de sistemas de controle, focando, principalmente, nos requisitos de qualidade do controle (Seção 3.4.1) e nas principais estratégias tradicionais de controle utilizadas na prática (Seção 3.4.2). Além disso, no final desta seção, são discutidas algumas questões relacionadas à implementação de estratégias de controle adaptativas, usadas para o enfrentamento de problemas nos quais as características do sistema é não linear ou variante no tempo (Seção 3.4.3).

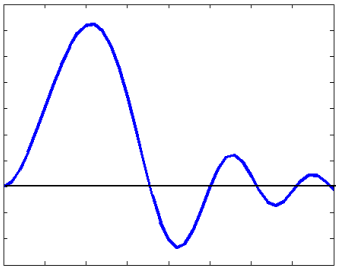
3.4.1 Requisitos de qualidade do controle

O projeto do controlador é uma peça importante na implementação do sistema de controle. Tais projetos devem levar em consideração os requisitos relacionados à qualidade de controle,

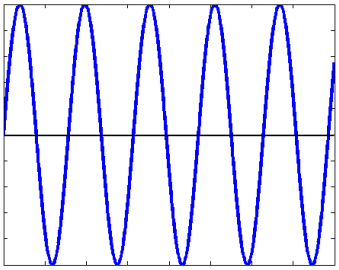
[1] Maiores detalhes (e.g. demonstrações e análises) podem ser encontrados em, por exemplo, Simon (2002), Ogata (2009), Hellerstein et al. (2004) etc.

tais como (Hellerstein et al., 2004): (a) estabilidade (*stability*), diz respeito à habilidade do sistema em convergir para um região de equilíbrio; (b) precisão (*accuracy*), diz respeito ao quanto um sistema consegue se aproximar do ponto de operação desejado; (c) tempo de convergência (*settling time*), diz respeito ao intervalo de tempo que o sistema leva para convergir para o ponto de operação desejado; e (d) máximo sobresinal (*overshoot*), refere-se à diferença máxima, durante a convergência, entre a saída do sistema e a saída desejada.

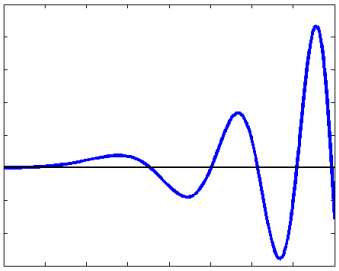
A estabilidade é um requisito fundamental no projeto de um sistema de controle, sendo o mesmo o primeiro a ser perseguido. De acordo com o requisito de estabilidade, o sistema pode ser classificado em (Colom, 2002): i) *estável*, se na ausência de qualquer perturbação o sistema permanece no mesmo estado ou ainda retorna para o estado de equilíbrio quando sujeito a alguma perturbação (ou condição inicial) – ver Figura 3.5(a); ii) *marginalmente estável*, se a saída do sistema não converge para o equilíbrio nem se torna ilimitada – ver Figura 3.5(b); e iii) *instável*, se a saída do sistema torna-se ilimitada – ver Figura 3.5(c).



(a) sistema estável



(b) sistema marginalmente estável



(c) sistema instável

Figura 3.5. Classificação do sistema quanto a estabilidade

Esses requisitos de qualidade do controle podem ser obtidos através da análise da localização dos pólos em malha fechada. Os pólos da malha fechada são as raízes da equação característica $1 + P(z)C(z) = 0$, isto é o denominador da função de transferência de malha fechada G_F, ver Equação 3.7. Os pólos de G_F representam o comportamento autônomo do sistema (Hellerstein et al., 2004), determinando a trajetória que o sistema controlado traçará quando sujeito a perturbações externas ou a incertezas intrínsecas ao modelo matemático proposto (Simon, 2002).

Em um sistema de controle de tempo discreto, a estabilidade é garantida se todos os pólos de

sua função de transferência em malha fechada (G_F) estão contidos no círculo unitário centrado na origem do plano z (Ogata, 2009), ver Figura 3.6.

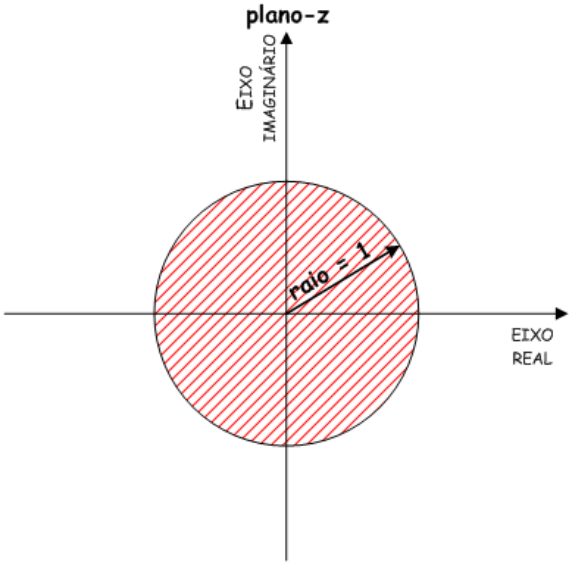


Figura 3.6. Região de estabilidade no plano z

O tempo de estabilização (K_s) e o máximo sobre-sinal (M_P) podem ser definidos usando a análise do pólo dominante. Para tanto, reescreve-se a função de transferência em malha fechada G_F da seguinte forma:

$$G_F(z) = \gamma \frac{(z-q_1)(z-q_2)...(z-q_m)}{(z-p_1)(z-p_2)...(z-p_n)} \tag{3.8}$$

em que p e q representam, respectivamentes os pólos e zeros de G_F.

O pólo dominante p^* é o pólo com a maior magnitude, isto é $p^* = \{p_i : |p_i| \geq |p_j|, \forall i, j\}$. Assim, supondo, na Equação 3.8, que $m \leq n$, então G_F pode ser aproximada por uma função de transferência de primeira ordem G_F^*, como (Hellerstein et al., 2004):

$$G_F^* = \frac{G_F(1)(1-p^*)}{z-p^*} \tag{3.9}$$

Uma vez conhecido o pólo dominante p^*, o tempo de estabilização é calculado usando (Ogata, 1995):

$$K_s = -\frac{4}{\log|p^*|} \tag{3.10}$$

O máximo sobre-sinal é medido como a maior diferença relativa entre o *set-point* (y_r) e a saída máxima (y_{max}), ou seja (Ogata, 1995):

$$M_P = \frac{|y_{max} - y_r|}{|y_r|} \tag{3.11}$$

Durante o projeto do sistema de controle, é possível estimar o máximo sobre-sinal (M_P)

usando o pólo dominante p^*, da seguinte forma (Hellerstein et al., 2004):

i) se o pólo dominante é um número real maior ou igual a zero (i.e. $p^* \geq 0$), então $M_P = 0$;

ii) se o pólo dominante é um número real menor que zero (i.e. $p^* < 0$), então $M_P = |p^*|$;

iii) caso contrário, se o pólo dominante é um número complexo, então:

$$M_P = |p^*|^{\frac{\pi}{\theta}}$$

em que θ é o argumento de p^*.

3.4.2 Estratégias de controle

As estratégias de controle determinam as leis de controle a serem implementadas pelo controlador. Uma vez conhecido o desvio entre os comportamentos esperado e desejado para o sistema dinâmico, as leis de controle definem que ponderações devem ser realizadas para que o controlador corrija tais desvios. Na indústria de controle e automação, as leis de controle *PID* (Proporcional-Integral-Derivativo[2]) e suas variações (*P*, *PD* e *PI*) têm encontrado grande aceitação na automação de processos industriais.

Conforme o nome sugere, a lei de controle PID propõe uma ação de controle que combina o efeito de três componentes: *ação proporcional*, considera um ajuste que é proporcional ao desvio entre a saída atual e o *set-point* (i.e. regula ou direciona a saída em função do desvio atual); *ação integral*, considera um ajuste que é proporcional ao somatório (i.e. integral) dos vários desvios ocorridos (i.e. realiza um ajuste fino observando os desvios no passado); e *ação derivativa*, considera um ajuste que é proporcional a atual variação (i.e. derivada) do desvio (i.e. antecipa o ajuste considerando prováveis desvios futuros). Assim, sendo y_k e y_r a saída atual e o *set-point*, respectivamente, a lei de controle *PID* é definida da seguinte forma:

$$u_k = K_P * error_k + K_I * \sum_{i=0}^{k-1} error_i + K_D * (error_k - error_{k-1}) \tag{3.12}$$

em que K_P, K_I e K_D representam, respectivamente, os ganhos proporcional, integral e derivativo do controlador, enquanto que $error_k = y_R - y_k$ representa o desvio atual.

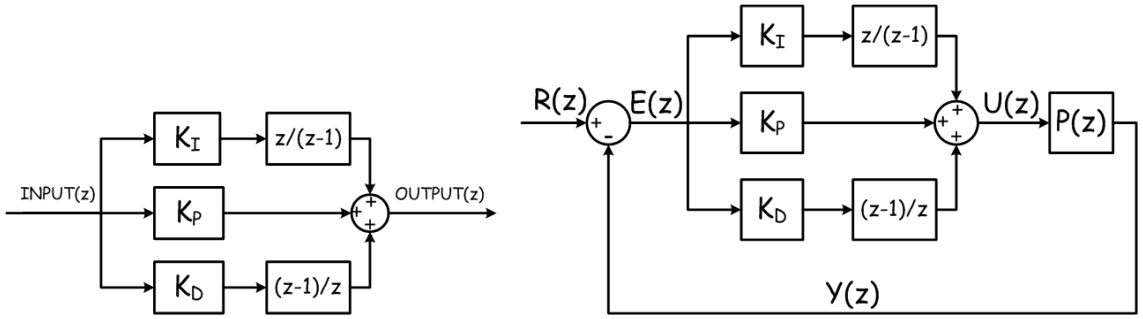
A função de transferência em z do controlador *PID* é dado por:

$$C(z) = K_P + K_I * \frac{z}{z-1} + K_D * \frac{z-1}{z} \tag{3.13}$$

em que $\frac{z}{z-1}$ e $\frac{z-1}{z}$ representam, respectivamente, operações de integração e diferenciação em z.

A Figura 3.7 apresenta, em diagrama de blocos, o controlador PID (Figura 3.7(a)) e a malha de fechada (Figura 3.7(b)) considerando controlador ($C(z)$) e planta ($P(z)$).

[2] ver Ogata (2009).



(a) controlador PID em malha aberta (b) Laço de controle, incluindo o controlador PID

Figura 3.7. Diagramas em blocos, considerando o controlador PID em malha aberta e em malha fechada

As diferentes variações do controlador *PID* são obtidas anulando uma das três componentes da ação de controle, ou seja (Ang; Chong; Li, 2005): um controlador *P* considera apenas o ajuste proporcional (i.e. K_I e K_D são iguais a zero); um controlador *PI* considera apenas os ajustes proporcional e integral (i.e. K_D igual a zero); um controlador *PD* considera apenas os ajustes proporcional e derivativo (i.e. K_I igual a zero); e assim por diante.

Note que, no caso do uso de controladores *PID* na implementação de sistemas computacionais, os controladores *P* e *PI* são, em geral, os mais usados. Devido ao comportamento extremamente aleatório das variáveis que determinam o comportamento dos sistemas computacionais, o uso de controladores com componentes derivativas pode induzir o controlador a realizar ajustes equivocados na *variável manipulada* – assim, controladores *PD* são, em geral, evitados. Nesse mesmo contexto, controladores *PID*, considerando as três componentes (proporcional, integral e derivativa), trazem um problema adicional, que é a dificuldade em obter estratégias eficientes para realizar a sintonia simultânea dos ganhos K_P, K_I e K_D (Hellerstein et al., 2004).

3.4.3 Controle adaptativo

Quando as características[3] do sistema mudam dinamicamente ou quando o mesmo apresenta comportamento não linear, é possível que a eficiência e a eficácia do sistema de controle sejam comprometidas – uma vez que, esses aspectos introduzem incertezas que muitas vezem são desconhecidas ou não podem ser adequadamente eliminadas durante o projeto. Conseqüentemente, pode ser necessário que o controlador também modifique dinamicamente suas características em respostas a tais incertezas (i.e. sistemas com características variantes no tempo ou comportamento não linear).

[3]Em termos de implementação ou modelagem, as características do sistema são traduzidas pelos valores dos parâmetros e pela estrutura de seu modelo matemático (Aguirre, 2007). Por exemplo, quando se define, em tempo de projeto, que a relação entre as entradas e as saídas do sistema pode ser definida em função das m últimas saídas e n últimas entradas, isto define a estrutura do modelo – ou seja, o modelo será composto por pelo menos $w = m + n$ variáveis associadas a tais entradas e saídas. Por outro lado, essas w variáveis precisam ser ponderadas (ou combinadas). Os parâmetros dos modelos são, por exemplo, os pesos (ou coeficientes) atribuídos a cada uma dessas variáveis.

Nesse contexto, a idéia do controle adaptativo é o projeto de mecanismos que permitam o ajuste automático dos parâmetros do controlador, de modo a garantir a eficiência e a eficácia do controle face às incertezas impostas pelas características ou pelo comportamento dos sistemas dinâmicos (Astrom; Wittenmark, 1995).

3.4.3.1 Estratégias de controle adaptativo

De um modo geral, estratégias de controle adaptativo são classificadas de acordo com as abordagens usadas para implementação do mecanismo de ajuste de parâmetros do controlador. Dentre as diferentes abordagens, destacam-se: i) controle adaptativo baseado em *escalonamento de ganho*; ii) controle adaptativo baseado em *modelo de referência*; e iii) controle adaptativo baseado em *auto-reguladores*.

O esquema de controle adaptativo baseado em escalonamento de ganho escolhe parâmetros (ganhos) diferentes para o controlador, de acordo com as condições de operação da planta. A idéia básica é: a planta pode apresentar diferentes condições de operação, então é razoável que sejam definidos diferentes valores de parâmetros para cada condição de operação. A Figura 3.8, adaptada de Astrom e Wittenmark (1995), apresenta a representação, em diagrama de blocos, desse esquema de controle.

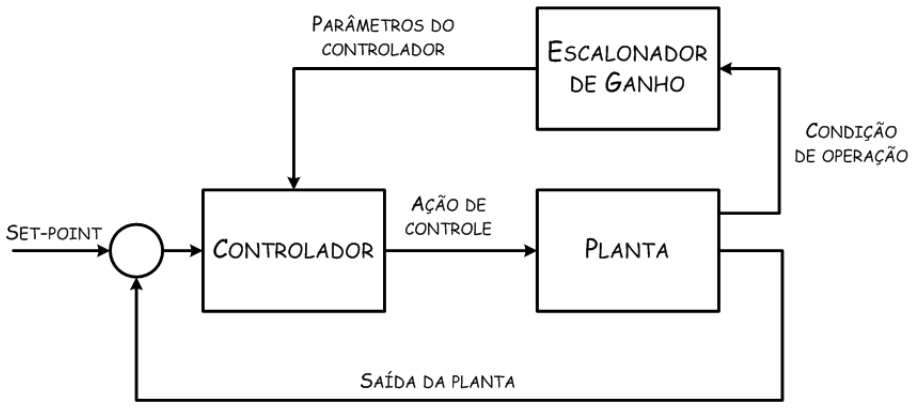


Figura 3.8. Sistema de controle adaptativo baseado em escalonamento de ganho

O esquema de controle adaptativo baseado em modelo de referência adapta os parâmetros do controlador de modo o laço de controle produza saídas (para uma dada entrada) de acordo com o modelo de referência desejado. A idéia básica é implementar um laço de controle de modo que, o modelo que descreve o comportamento da planta possa ser abstraído e a mesma possa se comportar de acordo com um modelo de referência previamente especificado. A Figura 3.9, adaptada de Astrom e Wittenmark (1995), apresenta a representação, em diagrama de blocos, desse esquema de controle.

O esquema de controle adaptativo baseado em auto-reguladores (ou controle auto-sintonizável) visa estimar em tempo de execução o modelo da planta e realizar o ajuste do controle usando o modelo estimado. A idéia básica nesse tipo de esquema é realizar algumas atividades de projeto

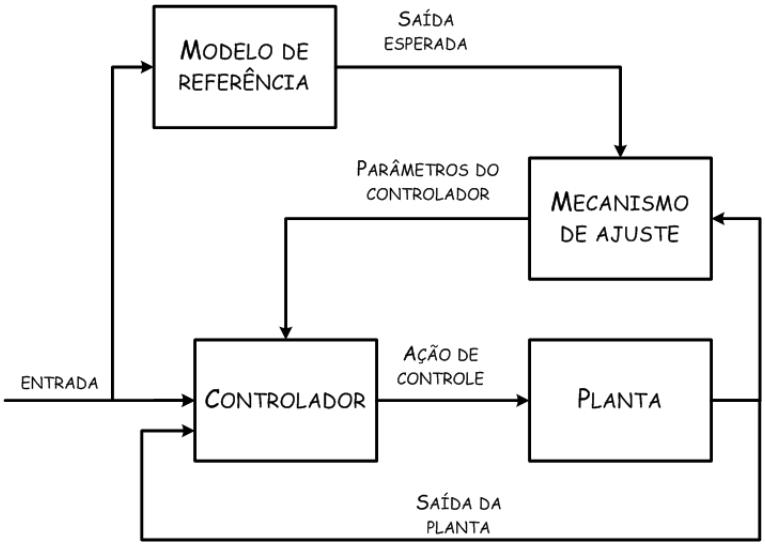


Figura 3.9. Sistema de controle adaptativo baseado em modelo de referência

(i.e. identificação dos parâmetros do modelo e sintonia do controlador) de forma automática, considerando, para tanto, os requisitos de qualidade de controle especificados. A Figura 3.10, adaptada de Astrom e Wittenmark (1995), apresenta a representação, em diagrama de blocos, desse esquema de controle.

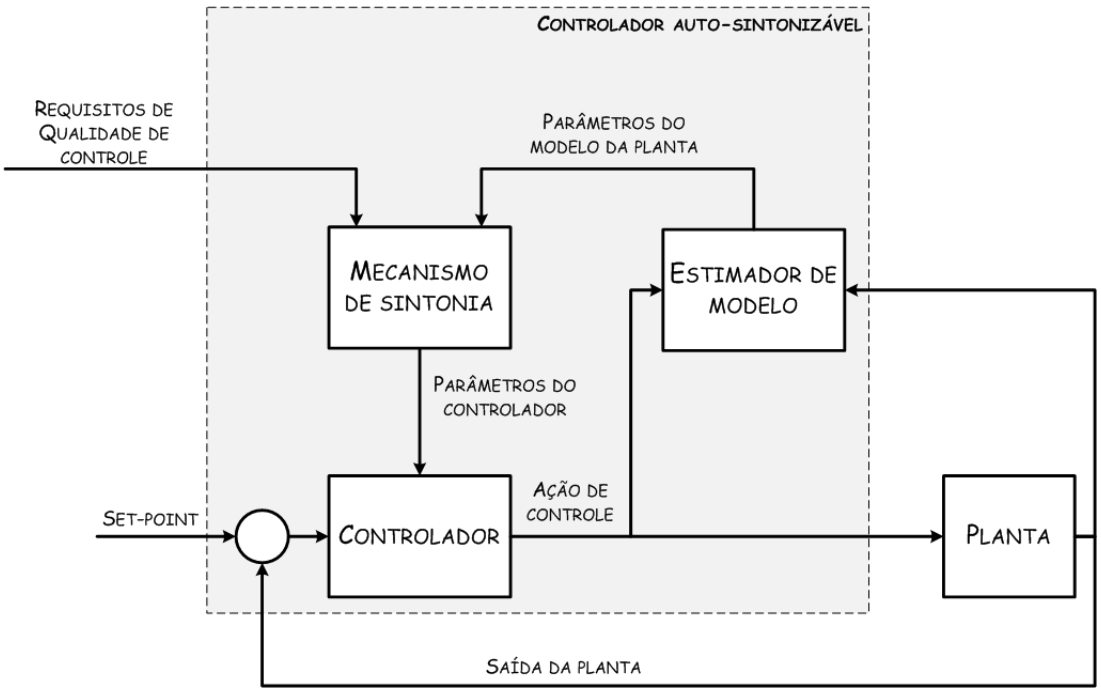


Figura 3.10. Sistema de controle adaptativo baseado em auto-sintonia

Os conceitos básicos e os diversos aspectos de implementações, relacionados aos esquemas de controle adaptativo apresentados, são discutidos com maiores detalhes em Astrom e Wittenmark (1995).

3.5 CONSIDERAÇÕES FINAIS

A teoria de controle é uma disciplina importante, não apenas para o entendimento e modelagem do comportamento dos sistemas dinâmicos, mas também para implementação de estratégias de controle que permitam fazer com que tais sistemas se comportem da forma esperada. Na perspectiva da análise e síntese, a teoria de controle traz um conjunto de ferramentas matemáticas que permitem: i) desenvolver representações variadas do comportamento dinâmico do sistema, possibilitando que tal comportamento seja analisado e entendido em diferentes perspectivas; e ii) sintetizar leis de controle que possibilitem o atendimento de diversos requisitos relacionados à qualidade de controle (i.e. estabilidade, convergência, precisão etc.).

O arcabouço matemático da teoria de controle, predominantemente usado pela engenharia de controle e automação de sistemas industriais, vem sendo adotado por diferentes áreas do conhecimento, incluindo ciência da computação – na qual a teoria de controle recentemente vem sendo usada, não apenas para permitir um melhor entendimento do comportamento dinâmico dos sistemas computacionais, mas também para dotar esses sistemas de facilidades de gestão e de adaptação (ver, por exemplo, Ren, Lin e Wei (2005), Xiong et al. (2006), Diao et al. (2005) etc.).

Sistemas computacionais são sistemas dinâmicos de tempo discreto. Por conta disto, este capítulo comentou questões relacionadas à transformada $\mathcal{Z}$, uma das ferramentas usadas para análise e projeto de estratégias de controle de tempo discreto. Diversas estratégias de controle podem ser encontradas na literatura, entretanto, as discussões foram direcionadas para as estratégias de controle da família PID – as quais são usadas nas implementações dos mecanismos autonômicos propostos nesta Tese.

Quando sistemas dinâmicos não lineares ou com características variantes no tempo são considerados, as abordagens tradicionais de controle podem se mostrar ineficientes e ineficazes. Nesse sentido, abordagens de controle adaptativas são sugeridas na literatura, como uma solução para adequar as estratégias de controle tradicionais para o enfrentamento desse problema. Dada a afinidade dos objetivos dessas estratégias adaptativas de controle, com aqueles considerados nos objetivos desta Tese, o controle adaptativo também é explorado, mais adiante, para a implementação dos mecanismos autonômicos de tolerância a falhas propostos.

PARTE II

CONTRIBUIÇÕES DA TESE

Capítulo 4

Este Capítulo apresenta a proposta de detecção autonômica de defeitos para sistemas distribuídos, contextualizando a mesma com relação ao estado da arte; e discutindo os aspectos de implementação, os experimentos realizados, os resultados obtidos e as perspectivas de trabalhos futuros.

DETECÇÃO AUTONÔMICA DE DEFEITOS PARA SISTEMAS DISTRIBUÍDOS

4.1 INTRODUÇÃO

Detectar defeitos por parada (i.e. *crash*[1]) de componentes é uma questão básica para o funcionamento de muitos protocolos e algoritmos usados na construção de sistemas distribuídos confiáveis. Por exemplo, em um esquema de replicação passiva, o defeito de uma réplica primária precisa ser prontamente detectado para que uma das réplicas secundárias assuma o papel da réplica faltosa com o mínimo de impacto para as aplicações ou serviços distribuídos.

A detecção de defeitos por *crash* geralmente considera que processos monitorados enviam periodicamente mensagens, ditas *heartbeats*, as quais indicam seu estado para processos monitores[2]. Um processo monitor determina um intervalo de tempo (i.e. *timeout* de detecção) durante o qual esperará pela chegada da mensagem de *heartbeat*. Se a mensagem de *heartbeat* não chega dentro do *timeout* de detecção, o processo monitor acreditará que o processo monitorado falhou. Esse modelo de detecção de defeitos depende das restrições temporais relacionadas ao processamento e transmissão das mensagens de monitoramentos trocadas entre os módulos do detector de defeitos.

Em um modelo de sistema distribuído assíncrono, os limites temporais para o processamento e para a transmissão das mensagens são desconhecidos (Lamport; Lynch, 1989), o que torna impossível solucionar certos problemas de tolerância a falhas de forma determinística (Fischer; Lynch; Paterson, 1985).

Para responder a essa impossibilidade, Chandra e Toueg (1996) introduziram o conceito de detectores de defeitos não confiáveis. Esses detectores são ditos não confiáveis, pois podem

[1] ver Seção 2.3.5.
[2] ver Seção 2.4.1.

apontar como defeituosos, processos corretos, e, por outro lado, deixar de apontar processos que efetivamente falharam. Chandra e Toueg (1996) demonstraram como, encapsulando certo nível de sincronia, detectores de defeitos não confiáveis podem ser usados para solucionar problemas fundamentais em sistemas distribuídos assíncronos – e.g. consenso distribuído (Guerraoui et al., 2000) e difusão atômica (Défago; Schiper; Urbán, 2004). Apesar da grande importância do trabalho de Chandra e Toueg (1996) para o entendimento e solução de problemas fundamentais em sistemas distribuídos, a ausência de limites temporais dos modelos assíncronos impõe grandes desafios práticos para a implementação de detectores de defeitos.

Um desses desafios é decidir valores apropriados para o *timeout* de detecção. *Timeouts* muito longos tornam a detecção dos defeitos lenta e compromete a resposta do sistema durante a ocorrência de falhas. Por outro lado, *timeouts* muito curtos podem degradar a confiabilidade do detector de defeitos e prejudicar o desempenho do sistema, uma vez que muitos algoritmos e protocolos, que utilizam a informação do detector de defeitos, podem realizar processamento e troca de mensagens adicionais por conta de falsas suspeitas de falhas. Nesse contexto, Macêdo (2000) propõe um mecanismo denominado CTI (*Connectivity Time Indicator*), o qual é inserido em uma abordagem de detecção de defeitos não confiável, com o intuito de sugerir *timeouts* de detecção dinâmicos, os quais variam de acordo com as condições de carga do ambiente distribuído. Esse trabalho é seguido por muitos outros com o intuito de embutir o uso de estimadores de *timeouts* na implementação de detectores de defeitos – como por exemplo, Bertier, Marin e Sens (2002), Macêdo e Lima (2004), Nunes e Jansch-Pôrto (2004), Falai e Bondavalli (2005), entre outros. O papel desses estimadores é sugerir, em tempo de execução, valores adequados para o *timeout* de detecção, de modo a tornar a detecção dos defeitos mais rápida com o mínimo possível de impacto para a confiabilidade do detector de defeitos. No entanto, tais trabalhos não consideram o ajuste dinâmico do período de monitoramento, outro fator importante para o desempenho dos detectores de defeitos – principalmente quando os custos computacionais da detecção precisam ser considerados.

Outro problema é que a especificação dos detectores de defeitos de Chandra e Toueg (1996) não foca em aspectos pertinentes a qualidade do serviço da detecção de defeitos, definindo propriedades difíceis de serem avaliadas na prática, por exemplo: "*em algum momento as falhas dos processos serão detectadas por algum processo correto*"[3]. Por conta disto, Chen, Toueg e Aguilera (2002) definiram métricas de qualidade de serviço para detecção de defeitos, as quais têm sido usadas para avaliar a velocidade e a precisão de diferentes implementações de detectores de defeitos. Com isso, o trabalho do projetista é definir um período de monitoramento e usar um estimador de *timeout* que consiga entregar um serviço de detecção de defeitos com um nível de qualidade de serviço adequado aos requisitos das aplicações.

Outro aspecto de projeto importante na concepção de detectores de defeitos, é compatibilizar o custo do serviço de detecção com as características dinâmicas dos ambientes computacio-

[3]tradução do inglês, *"eventually every process that crashes is permanently suspected by some correct process"* (Chandra; Toueg, 1996, p. 232).

nais modernos e de suas aplicações. Em ambientes distribuídos modernos, sujeitos a condições de carga variadas, a mudanças dinâmicas de requisitos de qualidade, ou a variações na disponibilidade de recursos, a configuração dos parâmetros operacionais dos detectores de defeitos, considerando métricas de qualidade de serviço e custo computacional da detecção, é uma atividade difícil de ser realizada. Para garantir, por exemplo, uma recuperação rápida na presença de componentes defeituosos, o período de monitoramento deve ser tão curto quanto possível. Todavia, períodos de monitoramento muito curtos podem incrementar demasiadamente o consumo de recursos computacionais, comprometendo o tempo de resposta das aplicações e diminuindo a eficiência e a velocidade dos mecanismos de detecção de defeitos e de recuperação.

Nesse contexto, Chen, Toueg e Aguilera (2002) propõem um procedimento para a configuração *off-line* dos detectores de defeitos. Esses mesmos pesquisadores sugerem que tal procedimento de configuração pode ser re-executado durante o funcionamento do detector, quando as características de carga do ambiente computacional mudam. Todavia, os efeitos dessa re-execução no desempenho dos detectores de defeitos não foram avaliados.

Bertier, Marin e Sens (2003) comentam brevemente um procedimento baseado em consenso para ajustar dinamicamente o período de monitoramento quando certas condições de carga são verificadas. Entretanto, esses autores não detalham a solução, nem avaliam a mesma considerando métricas de qualidade de serviço de detecção.

Mills et al. (2004), Xiong et al. (2006) e So e Sirer (2007) exploram a configuração dinâmica de detectores de defeitos. Entretanto, esses pesquisadores consideram que o comportamento do ambiente computacional não muda e não demonstram como dinamicamente configurar os detectores de defeitos usando métricas de qualidade de serviço, como tempo de detecção, duração da falsa suspeita e intervalo entre falsas suspeitas.

Recentemente, Dixit e Casimiro (2010) propuseram uma abordagem de detecção de defeitos que usa métricas de qualidade de serviço para o ajuste do *timeout* de detecção. Entretanto, esses pesquisadores não consideram o ajuste dinâmico do período de monitoramento, outro aspecto importante para a adequação do custo computacional relacionado ao serviço de detecção de defeitos.

Diferentes dos trabalhos apresentados na literatura até então, o detector de defeitos proposto nesta Tese é o primeiro capaz de auto-configurar seus parâmetros operacionais em tempo de execução, em resposta às mudanças no ambiente computacional ou nas aplicações, observando, para tanto, os requisitos de qualidade de serviço definidos pelo usuário (Sá; Macêdo, 2010a, 2010b). Sistemas com tais características são ditos autonômicos e a auto-configuração é uma de suas propriedades básicas (Huebscher; Mccann, 2008).

A maior dificuldade na implementação de detectores de defeitos auto-configuráveis é a modelagem da dinâmica do sistema distribuído – a qual é difícil de caracterizar usando funções de distribuição de probabilidades quando, por exemplo, ambientes com condições de carga variadas são considerados (e.g. ambientes de computação em nuvem). Para modelar tal comportamento dinâmico dos sistemas distribuídos, este trabalho utiliza a teoria de controle reali-

mentado, comumente aplicada na área de automação de sistemas industriais (Ogata, 1995).

A abordagem de detecção autonômica de defeitos proposta nesta Tese foi completamente implementada e avaliada através de simulações, usando métricas de qualidades de serviço tais como tempo de detecção, intervalo entre falsas suspeitas, taxa de falsas suspeitas e disponibilidade de detecção. Essas métricas permitiram avaliar a velocidade e a precisão das detecções sob condições de carga variadas. Além disso, mesmo sem trabalhos relacionados para uma comparação direta de desempenho, a abordagem foi comparada com abordagens tradicionais de detecção adaptativa, as quais foram manualmente configuradas com diferentes períodos de monitoramento. Os resultados demonstraram que o detector autonômico proposto possui desempenho melhor que os detectores adaptativos considerados.

Assim, o restante deste Capítulo: a) discute o modelo de sistema adotado na concepção da proposta de detecção autonômica de defeitos (Seção 4.2); b) apresenta os detalhes de implementação da detecção autonômica (Seção 4.3); c) descreve os experimentos realizados para verificação do desempenho do detector autonômico, quando comparado com abordagens adaptativas existentes na literatura (Seção 4.4); e d) apresenta algumas considerações sobre as questões discutidas neste Capítulo e sobre as perspectivas de trabalhos futuros associados à proposta de detecção autonômica de defeitos (Seção 4.5).

Além disso, o Apêndice A apresenta maiores detalhes de alguns conceitos e equações relacionados à implementação da proposta de detecção autonômica, os quais foram omitidos, nas discussões deste Capítulo, com o intuito de facilitar a leitura e entendimento da proposta. Mais ainda, algumas avaliações experimentais adicionais também podem ser encontradas no Apêndice A – tais avaliações apresentam: i) o impacto de cada uma das métricas de qualidade de serviço na sintonia do detector autonômico; e ii) o desempenho de implementações alternativas do detector autonômico proposto.

4.2 MODELO DE SISTEMA

O modelo de sistema distribuído considerado pela proposta é constituído por um conjunto finito de n processos $\Pi = \{p_1, p_2, ..., p_n\}$. Esses processos são interconectados através de canais de comunicação não confiáveis, os quais podem duplicar ou perder mensagens. Se uma mensagem enviada por um processo do sistema é corrompida, a mesma será descartada. Além disso, se um processo p_i envia, para um processo correto p_j, uma mesma mensagem sucessivas vezes, em algum momento tal mensagem será recebida com sucesso por p_j – i.e. os processos interagem através de canais de comunicação do tipo *fair-lossy*[4].

Apesar dos processos possuírem acesso aos seus relógios locais, não são assumidos relógios sincronizados e as taxas de desvio desses relógios em relação ao tempo real são valores muitíssimo menores que os tempos de transmissão e processamento das mensagens, portanto, são fatores desprezados nas estimativas dos atrasos. Além disso, não são assumidos limites

[4] ver Lynch (1996, p. 691–732).

temporais para processamento e transmissão das mensagens.

Os processos do sistema podem falhar por *crash* – falhas bizantinas (Lamport; Shostak; Pease, 1982) não são consideradas. Cada processo tem acesso a um módulo local de um serviço de detecção de defeitos, o qual provê informações, possivelmente não confiáveis, sobre o estado dos demais processos do sistema. Este serviço de detecção de defeitos é concebido considerando o estilo de monitoramento *pull* (ver Seção 2.4.1.1). As mensagens de monitoramento trocadas entre os módulos do serviço detecção de defeitos são assinaladas com números seqüenciais.

Os recursos do ambiente computacional e os padrões de carga de suas aplicações podem mudar dinamicamente. Além disso, o serviço de detecção de defeitos não possui qualquer conhecimento sobre as características do ambiente computacional. Para o serviço de detecção, as únicas informações disponíveis são aquelas obtidas através das mensagens de monitoramento trocadas entre os seus módulos.

4.3 A PROPOSTA DE DETECÇÃO AUTONÔMICA DE DEFEITOS

A proposta de detecção autonômica de defeitos considera a implementação de um gestor autonômico (ou controlador), o qual observa o comportamento do serviço básico de detecção de defeitos (elemento gerenciado ou planta). Então, baseado nos requisitos de qualidade de serviço de detecção (i.e. TD^U, TM^U e TMR^L) e nas restrições de consumo de recursos (RC^D) previamente definidos, esse gestor autonômico calcula o período de monitoramento (δ) e o *timeout* de detecção (rto), os quais são usados por um processo monitor p_i para checar o estado de um processo monitorado p_j (ver Figura 4.1).

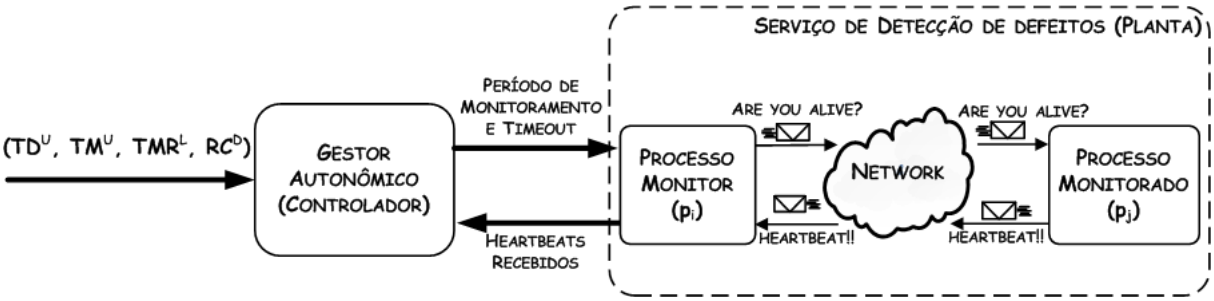


Figura 4.1. Abordagem de detecção autonômica de defeitos

O gestor autonômico proposto executa três tarefas básicas: (i) sensoriamento de características do ambiente computacional e do desempenho do serviço de detecção de defeitos – para observar o atendimento das metas de desempenho definidas pelos usuários ou aplicações; (ii) regulação do *timeout* de detecção – para atender a relação de compromisso entre a precisão e a velocidade da detecções de defeitos; (iii) regulação do período de monitoramento – para atender a relação de compromisso entre a velocidade e o custo computacional das detecções. Os detalhes de implementação de cada uma dessas tarefas são descritos nas subseções a seguir.

4.3.1 Sensoriamento do ambiente e do serviço de detecção

A tarefa de sensoriamento consiste em estimar os atrasos no ambiente computacional e o desempenho do detector em termos de qualidade de serviço de detecção e de suposições a respeito do consumo de recursos. Para tanto, a tarefa de sensoriamento extrai essas informações apartir do comportamento das mensagens de monitoramento trocadas entre os módulos do serviço de detecção de defeitos (ver Figura 4.2).

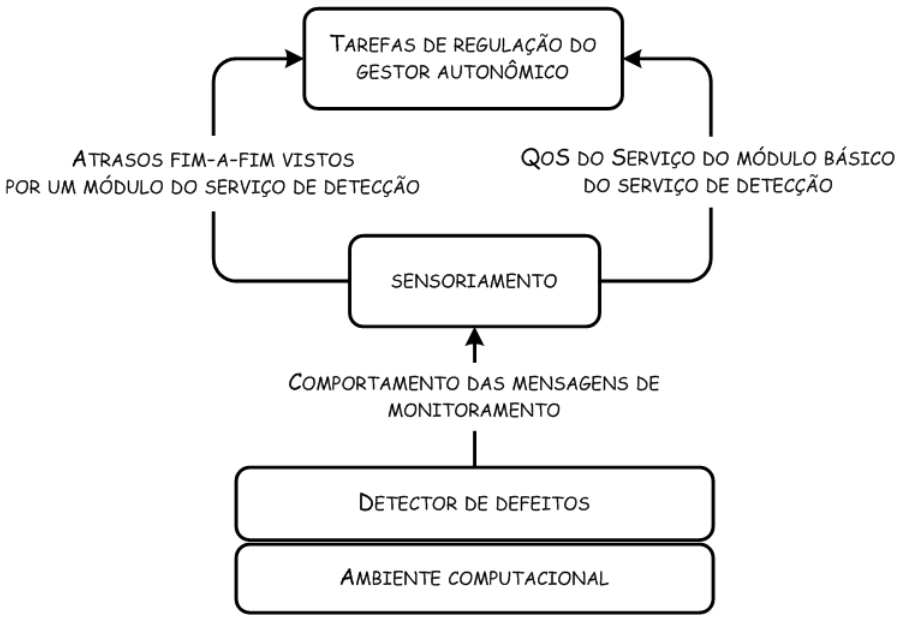


Figura 4.2. Visão geral da atividade de sensoriamento

As estratégias e conceitos definidos e utilizados para a tarefa de sensoriamento são descritos no restante desta subseção.

4.3.1.1 Sensoriamento dos atrasos no ambiente computacional

O sensoriamento do consumo dos recursos do ambiente computacional é o ponto de partida para que o gestor autonômico possa realizar a regulação dos parâmetros do detector de defeitos. Tal atividade é importante para a regulação dos períodos de monitoramentos usados pelos módulos do serviço de detecção de defeitos.

Quando não existem informações disponíveis a respeito das características do ambiente computacional, os atrasos fim-a-fim, observados na transmissão das mensagens de monitoramento do detector de defeitos, podem dar um indicativo do suposto consumo de recursos no ambiente computacional em um dado instante. Para tanto, 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?"* aya_k e de recebimento de seu respectivo *heartbeat* hb_k (ver Figura 4.3).

Perdas de mensagens dificultam a interação entre processos do sistema e impactam direta-

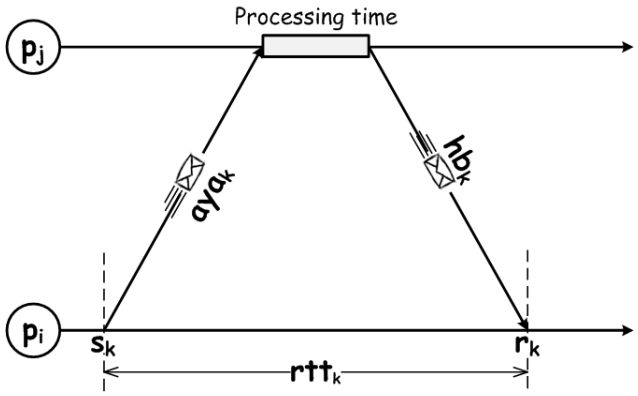


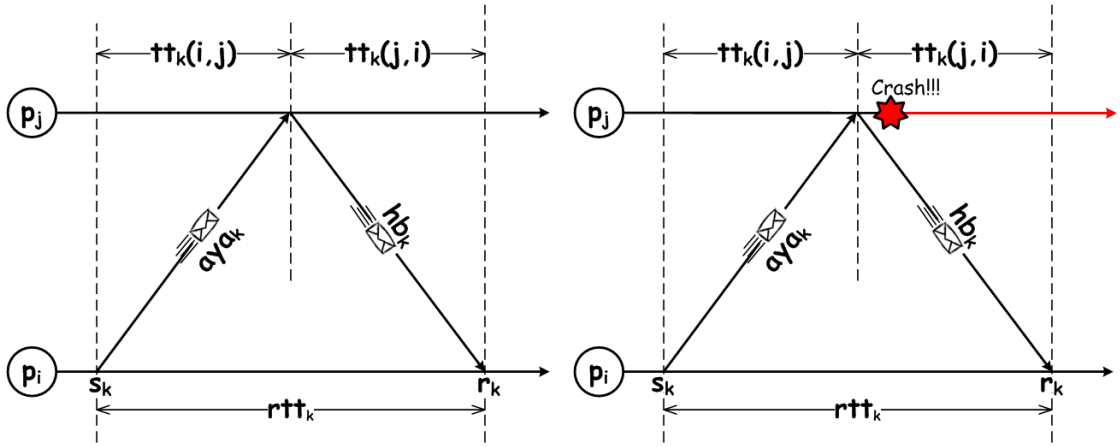
Figura 4.3. Estimativa do atraso de ida-e-volta *rtt*

mente na percepção de um processo monitor p_i, a respeito do estado de um processo monitorado p_j. Por exemplo, p_i pode suspeitar da falha de p_j por causa da perda de uma mensagem de monitoramento (i.e. um *"are you alive?"* ou um *heartbeat*).

No caso da detecção de defeitos, mensurar os atrasos de ida-e-volta (*rtt*) a partir da diferenças entre os instantes de recebimento (r) e de envio (s) pode ser uma tarefa onerosa ou prejudicar a reação do gestor autonômico, quando perdas de mensagens precisam ser consideradas. Na prática, é possível implementar uma estratégia de retransmissão na qual o atraso de ida-e-volta é mensurado considerando o instante de envio do primeiro *aya* até o instante de tempo em que um *hb* é finalmente recebido. Entretanto, esse tipo de implementação tem implicações no custo de monitoramento. Por exemplo, retransmissões desnecessárias podem ocorrer se o processo monitor não sabe se o processo monitorado falhou ou se a mensagem foi perdida – o que é um caso comum em muitos sistemas distribuídos típicos. Além disso, o custo com retransmissões pode ser evitado, uma vez que, o monitoramento de defeitos é periódico, o que significa que a perda de uma mensagem de monitoramento em um ciclo de monitoramento pode ser compensada pelo envio de uma (nova) mensagem em um ciclo de monitoramento posterior. Outra alternativa seria obter uma estimativa da probabilidade de perda de mensagens. Entretanto, esta abordagem pode levar o gestor autonômico a atuar de forma ineficiente, uma vez que, se o ambiente possui características dinâmicas a distribuição de probabilidade não é conhecida a priori e pode mudar dinamicamente. Assim, um ponto importante na construção do gestor autonômico é obter uma medida eficiente e que evite custos desnecessários para o monitoramento das falhas. Para tanto, é necessário analisar, do ponto de vista do serviço de detecção, o atraso na interação entre pares de processos do sistema. O atraso de interação é um conceito introduzido neste trabalho e é apresentado a seguir.

Em sistemas distribuídos, quando um processo monitor p_i recebe um *heartbeat* de um processo monitorado p_j, o mesmo não tem qualquer garantia de que p_j ainda esteja funcionando. Isto porque o recebimento de um *heartbeat* carrega apenas informações sobre o estado de p_j no passado. Portanto, se p_i recebe um *heartbeat* em um instante r_k, o mesmo sabe apenas que p_j esteve funcionando corretamente até o instante $r_k - tt_k(j,i)$, em que $tt_k(j,i)$ é representa o

tempo de viagem de hb_k de p_j para p_i (ver Figura 4.4).



(a) p_j ainda está correto em r_k (b) p_j está falho em r_k

Figura 4.4. Interação entre p_i e p_j

Um fato importante, na análise da interação entre os processos p_i e p_j, é que 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. Com isso, a partir da discussão apresentada, é introduzido, nesta Tese, o conceito de intervalo de incerteza:

Definição 4.1 ***Intervalo de incerteza*** *(uti): é uma medida do envelhecimento da informação obtida por um processo monitor p_i a respeito do estado de um processo monitorado p_j, podendo ser calculada, em dado instante t, por: $uti(t) = t - [r_u - tt_u(j,i)]$, em que r_u e $tt_u(j,i)$ são, respectivamente, o instante de chegada e tempo de viagem do último heartbeat recebido hb_u.* ∎

A partir da Definição 4.1, é possível obter a primeira propriedade importante a respeito do intervalo de incerteza:

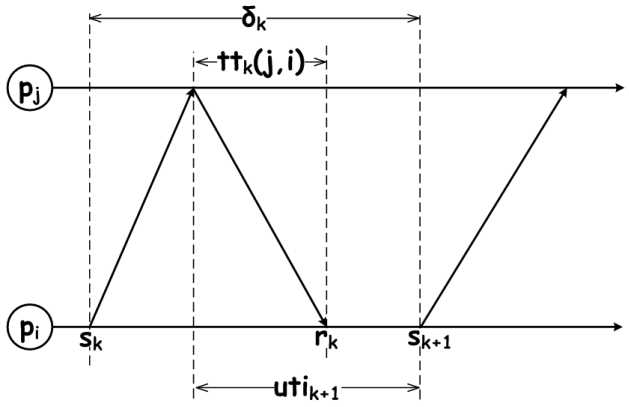
Propriedade 4.1 ***Influência do atraso fim-a-fim em*** *uti: A magnitude do intervalo de incerteza, estimada por um processo monitor p_i, depende diretamente da magnitude do atraso fim-a-fim (tt) entre um processo monitorado p_j e o processo p_i.* ∎

O gestor autonômico estima *uti* nos instantes de envio dos "*are you alive?*". Assim, a cada intervalo k, o mesmo calcula $uti_k = uti(s_k)$, ou:

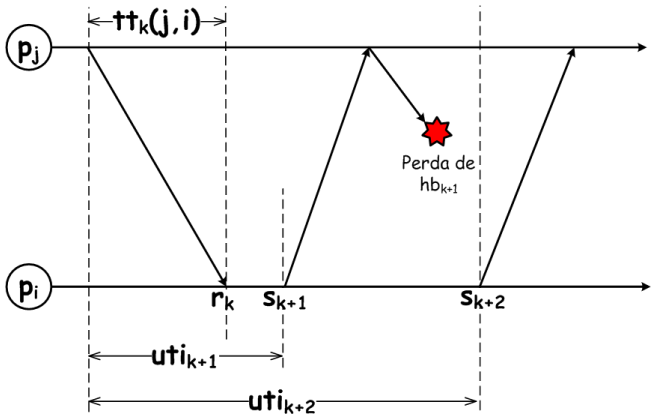
$$uti_k = s_k - [r_u - tt_u(j,i)] \tag{4.1}$$

A Figura 4.5 ilustra *uti* para (a) o caso no qual mensagens não são perdidas e (b) o caso no qual *heartbeats* são perdidos.

O conceito de intervalo de incerteza possui mais três propriedades importantes:



(a) *uti* sem perda de mensagens



(b) *uti* com perda de mensagens

Figura 4.5. Interação entre p_i e p_j

Propriedade 4.2 ***Influência do período de monitoramento em*** uti*: A magnitude do intervalo de incerteza, estimada por um processo monitor* p_i*, depende diretamente da magnitude do período de monitoramento* (δ) *usado por* p_i *para verificar o estado de um processo monitorado* p_j*.*

Demonstração. Suponha que hb_u foi recebido por p_i. Uma vez que $rtt_u = r_u - s_u$ e $rtt_u = tt_u(i,j) + tt_u(j,i)$, então, $r_u - tt_u(j,i) = s_u + tt_u(i,j)$. Assim, no instante s_{u+1}, o gestor autonômico estimará $uti_{u+1} = s_{u+1} - [s_u + tt_u(i,j)]$. Dado que $\delta_u = s_{u+1} - s_u$, então $uti_{u+1} = \delta_u - tt_u(i,j)$. ∎

Propriedade 4.3 ***Influência de perdas de mensagens de monitoramento em*** uti*: A magnitude do intervalo de incerteza, estimada por um processo monitor* p_i*, cresce na medida em que mensagens de monitoramento são perdidas.*

Demonstração. Suponha que após hb_u ter sido recebido por um processo monitor p_i, w mensagens de monitoramento consecutivas foram perdidas. Então, usando a Equação 4.1, $uti_{u+1} = s_{u+1} - [r_u - tt_u(j,i)]$, da mesma forma, $uti_{u+2} = s_{u+2} - [r_u - tt_u(j,i)]$, ..., $uti_{u+w} = s_{u+w} - [r_u - tt_u(j,i)]$. Dado que $s_u < s_{u+1} < ... < s_{u+w}$, então $uti_{u+1} < uti_{u+2} < ... < uti_{u+w}$. ∎

Propriedade 4.4 ***Influência da falha de um processo monitorado*** p_j ***em*** uti*: A magnitude do intervalo de incerteza, estimada por um processo monitor* p_i*, cresce indefinidamente quando um processo monitorado* p_j *falha.*

Demonstração. Suponha que hb_u é o último *heartbeat* recebido por um processo monitor p_i após a falha de um processo monitorado p_j. Deste modo, p_i não receberá, de p_j, qualquer *heartbeat* hb_x, com $x > u$. Sendo assim, na medida em que $x \to \infty$, então $s_x \to \infty$ e, conseqüentemente, $uti_x \to \infty$. ∎

Conforme demonstrado nas propriedades 4.1 e 4.3, o intervalo de incerteza permite que o gestor autonômico possa observar o impacto do atraso e da perda de mensagens sem um custo adicional para o monitoramento. Entretanto, para que este conceito possa ser utilizado, é necessário que se obtenha uma estimativa do atraso fim-a-fim (tt). Para tanto, em um dado intervalo k, estima-se $tt_k(i,j) = tt_k(j,i) = \dfrac{rtt_k}{2}$.

Apesar de, na prática, as mensagens de *"are you alive?"* e de *"heartbeat"* poderem ter tempos de viagem diferentes, assume-se, por questões de simplicidade, $rtt/2$ como uma estimativa para o tempo de viagem de uma mensagem de monitoramento. Observe que, para o caso da estimativa do suposto percentual consumo de recursos, um aumento ou diminuição de rtt será refletido em $rtt/2$. Dessa forma, o fato de as mensagens *aya* e *hb* poderem ter tempos de viagem diferentes tem pouca importância no contexto da gestão autonômica do serviço de detecção de defeitos.

Um inconveniente no uso de uti, na atividade de regulação do período de monitoramento, é o fato de, em cenários sem perda de mensagens ou falhas, o mesmo ser influenciado pelo período de monitoramento determinado pelo gestor autonômico, ver Propriedade 4.2. Isto porque, nesses cenários, a informação fornecida pela tarefa de sensoriamento para a tarefa de regulação de período estará *contaminada* com o valor do período de monitoramento – necessitando, assim, que tal tarefa de regulação realize um tratamento antes que a informação possa ser efetivamente usada. Para solucionar tal inconveniente, define-se , nesta Tese, o conceito de atraso de interação:

Definição 4.2 ***Atraso de interação****: é uma medida do atraso, imposto pelo ambiente computacional, para que o estado de um processo monitorado* p_j *seja obtido por um processo monitor* p_i*, podendo ser calculado em um dado ciclo de monitoramento* k *por:*

$$d_k = |uti_k - \delta_{k-1}| \tag{4.2}$$

em que δ *representa o período de monitoramento.* ∎

Na ausência de falhas, o atraso de interação apresenta uma propriedade importante:

Propriedade 4.5 ***Atraso de interação em ciclos de monitoramento livres de falhas****: Quando não existem perdas de mensagens ou falha do processo monitorado, o atraso de interação,*

observado por um processo monitor p_i, representa uma estimativa do atraso fim-a-fim – i.e. $d_k = \frac{rtt_{k-1}}{2}$.

Demonstração. Sendo $d_k = |uti_k - \delta_{k-1}|$ e $uti_u = \delta_{k-1} - tt_{k-1}(i,j)$, então $d_k = |-tt_{k-1}(i,j)|$. Uma vez que, $tt_{k-1}(i,j) = \frac{rtt_{k-1}}{2}$ e $rtt \geq 0$, então $d_k = \frac{rtt_{k-1}}{2}$. ∎

Em cenários com perdas de mensagens de monitoramento ou falha do processo monitorado, o atraso de interação herda, do intervalo de incerteza, duas propriedades importantes:

Propriedade 4.6 ***Influência de perdas de mensagens de monitoramento em*** d*:* *A magnitude do atraso de interação, estimada por um processo monitor p_i, cresce na medida em que mensagens de monitoramento são perdidas na interação com um processo monitorado p_j.*

Demonstração. Segundo a Propriedade 4.3, se após o recebimento de um *heartbeat* hb_u, uma seqüência de w mensagens de monitoramento é perdida, tem-se que $uti_{u+1} < uti_{u+2} < ... < uti_{u+w}$. Se $d_x = |uti_x - \delta_{x-1}|$, então $d_{u+1} < d_{u+2} < ... < d_{u+w}$. ∎

Propriedade 4.7 ***Influência da falha de um processo monitorado*** p_j ***em*** d*:* *A magnitude do atraso de interação, estimada por um processo monitor p_i, cresce indefinidamente quando um processo monitorado p_j falha.*

Demonstração. A partir da Propriedade 4.4, se na falha p_j, $uti \to \infty$, na medida em que $s \to \infty$, então por definição $d \to \infty$. ∎

A Propriedade 4.5 é importante porque permite ao gestor autonômico usar o atraso fim-a-fim como uma suposta estimativa para o consumo de recursos. Observe que o aumento dos atrasos fim-a-fim associados às mensagens de monitoramento não necessariamente representam um aumento do suposto uso de recursos no ambiente computacional. Isto pode se causado, por exemplo, por defeitos nos componentes do sistema (e.g. interferência nos canais, falhas de gestão de memória etc.). Entretanto, seja por conta do uso efetivo dos recursos, seja por conta de defeitos em componentes do sistema, um aumento nos atrasos fim-a-fim (ou nos intervalos entre chegadas) representam uma suposta diminuição do percentual de recursos disponíveis.

Através da Propriedade 4.6 o gestor autonômico, durante a ocorrência de perda de mensagens, percebe atrasos maiores e conseqüentemente estima que existem menos recursos disponíveis. Com isso, o mesmo pode fornecer períodos de monitoramento maiores nestes casos – o que representa um aspecto importante, uma vez que, a ocorrência de perda de mensagens pode ser um indicativo de possíveis congestionamentos ou altas demandas por recursos computacionais.

A Propriedade 4.7, por sua vez, é importante para permitir que o gestor autonômico leve o período de monitoramento para o máximo possível, após a ocorrência de falhas do processo monitorado. Este é um aspecto relevante para assegurar que o custo do serviço de detecção autonômica é reduzido quando processos monitorados falham.

Observe que o atraso de interação, além de possuir propriedades favoráveis a gestão autonômica do detector de defeitos, representa uma métrica operacional, não demandando, portanto, o uso de qualquer função de distribuição de probabilidade ou qualquer outra informação definida a priori – um aspecto importante para a regulação do período de monitoramento quando se considera ambientes com características dinâmicas.

A partir das definições acima, é importante descrever que informações são extraídas do atraso de interação para que o mesmo possa realizar uma estimativa do percentual de uso (ou de disponibilidade) de recursos no ambiente computacional, como é discutido na Seção 4.3.3. Para tanto, o gestor autonômico caracteriza o ambiente computacional através de d, de seus valores máximos e mínimos, ditos d^U e d^L, e de sua máxima variação, dita j^U. Os valores de d^L, d^U e j^U podem ser estimados assumindo o menor e o maior valor de d e sua maior variação durante a execução do detector, respectivamente.

Entretanto, se as características do ambiente variam, é possível que os valores de d^L, d^U e j^U observados pelo gestor autonômico também mudem. Com isso, considera-se um fator de esquecimento $f \in [0,1]$ na estimativa de tais valores (f é descrito mais adiante). Assim, o gestor autonômico estima os valores de d^L, d^U e j^U usando o procedimento descrito no Algoritmo 4.1. Nesse Algoritmo, antes de estimar os atrasos, o gestor autonômico assume $d_0^L = d_0^U = rtt_0/2$ e $j_0^U = 0$ (Linhas 1–3). Em seguida, antes do envio de cada *heartbeat* hb_k, o gestor autonômico calcula d_k^L, d_k^U e j_k^U (Linhas 7–18). Contudo, para realizar tais estimativas, primeiramente o gestor autonômico calcula os valores do fator de esquecimento f e do atraso de interação d (Linhas 5–6).

Definir f arbitrariamente pode levar o gestor autonômico a realizar ajustes inadequados em δ, por conta da velocidade da variação de d^L, d^U e j^U. Assim, f é definido a partir do tempo máximo de detecção (TD^U) e do atraso de interação mínimo (d^L):

$$f_k = \frac{max[0, (TD^U - d_k^L)]}{TD^U} \tag{4.3}$$

em que $f_0 = 1$ e TD^U é definido pelo usuário (ou pelas aplicações).

Se o tempo máximo de detecção definido pelo usuário é aproximadamente igual à estimativa do atraso de interação mínimo (i.e. $TD^U \approx d^L$), então o fator de esquecimento f tende a 0 (esquecimento máximo), assim o gestor autonômico não memoriza os valores de d^L, d^U e j^U e os mesmos variarão a cada observação. Observe que, neste caso, o gestor autonômico não tem liberdade para a regulação do período de monitoramento – uma vez que, períodos de monitoramento muito menores que d^L pode implicar em um uso demasiado de recursos do ambiente, enquanto que períodos de monitoramento maiores que d^L violam o tempo máximo de detecção definido pelo usuário.

Por outro lado, se o tempo máximo de detecção definido pelo usuário é muito maior que a estimativa do atraso de interação mínimo (i.e. $TD^U >> d^L$), então o fator de esquecimento f tende a 1 (esquecimento nulo), significando que a atualização dos valores de d^L, d^U e j^U

Algoritmo 4.1: *Sensoriamento do ambiente computacional*

```
1  define d_0^L = rtt_0/2;
2  define d_0^U = rtt_0/2;
3  define j_0^U = 0;
4  before event hb_k sending do
5      use the Equation 4.3 to compute f_k;
6      use the Equation 4.2 to compute d_k;
7      if d_{k-1}^L > d_k then
8          d_k^L = d_k;
9      else
10         d_k^L = f_k * d_{k-1}^L + (1 - f) * d_k;
11     end if
12     if d_{k-1}^U < d_k then
13         d_k^U = d_k;
14     else
15         d_k^U = f_k * d_{k-1}^U + (1 - f) * d_k;
16     end if
17     compute j_k = |d_k - d_k^L|;
18     if j_{k-1}^U < j_k then
19         j_k^U = j_k;
20     else
21         j_k^U = f_k * j_{k-1}^U + (1 - f) * j_k;
22     end if
23 end event
```

não é relevante e os mesmos serão atualizados de forma mais conservadora pelo gestor autonômico (i.e. desconsiderando o fator de esquecimento). Neste caso, o gestor autonômico possui liberdade para ajustar o valor do período de monitoramento de modo a atender a relação de compromisso, da detecção de defeitos, entre custo (i.e. suposto percentual de consumo de recursos), velocidade (i.e. tempo de detecção) e precisão (i.e. quantidade e duração das falsas suspeitas).

Uma última questão a ser considerada pelo gestor autonômico no processo de sensoriamento do ambiente são os ajustes desnecessários no período de monitoramento, provocados por variações espúrias (ou repentinas) nos atrasos fim-a-fim. Observe que, apesar do período de monitoramento usado pelo serviço de detecção de defeitos contribuir para o suposto percentual de consumo de recursos (observado a partir dos atrasos), as aplicações que compõem o ambiente computacional possuem uma forte influência no atraso fim-a-fim, fazendo com que o mesmo possa variar aleatoriamente – incluindo, desse modo, vários picos espúrios (ou perturbações), provocadas, por exemplo, pelo início de uma nova transmissão ou chegada de um novo processo ao sistema.

Nesse sentido, usa-se de um filtro para evitar que o gestor autonômico ajuste desneces-

sariamente o período de monitoramento. Desse modo, após encontrar d^L, d^U e j^U, o gestor autonômico realiza uma filtragem em d de modo a eliminar variações espúrias. Assim, sendo d^F a versão filtrada de d, com $d_0^F = d_0$, calcula-se:

$$d_k^F = f_k * d_{k-1}^F + (1 - f_k) * d_k$$

As variáveis d, d^L, d^U, d^F representam informações do passado, obtidas a partir das mensagens de *heartbeats* recebidas. Para predizer o valor do atraso de interação, dito d^E, uma margem de segurança (j^U) é adicionada à versão filtrada de d:

$$d_k^E = d_k^F + j_k^U$$

A variável d^E é usada durante a regulação (ou sintonia) do período de monitoramento. Uma discussão mais detalhada do uso dessa e das demais variáveis estimadas e coletadas, durante o sensoriamento do ambiente, é apresentada na Seção 4.3.3.

4.3.1.2 Sensoriamento da QoS de detecção

O sensoriamento da qualidade do serviço básico de detecção é um aspecto chave para que o gestor autonômico possa compatibilizar o desempenho do serviço de detecção de defeitos com o desempenho demandado pelas aplicações. Para tanto, consiste em estimar, em tempo de execução, o desempenho do serviço de detecção de defeitos em termos de: (a) *tempo de resposta* – o qual é representado pelo tempo de detecção (TD); e (b) *confiabilidade* – a qual é representada pela duração da falsa suspeita (TM), pelo intervalo entre falsas suspeitas (TMR)[5] e pela disponibilidade de detecção (AV).

A disponibilidade de detecção é uma métrica operacional introduzida nesta Tese, ver Sá e Macêdo (2010b), e é discutida mais adiante.

4.3.1.2.1 Sensoriamento do tempo de detecção

O sensoriamento do tempo de detecção é de particular interesse, pois é um elemento que permite ao gestor autonômico conciliar, durante a regulação do período, o custo computacional associado ao suposto consumo de recurso do serviço de detecção com a velocidade da detecção. Todavia, uma estimativa mais precisa do tempo de detecção demanda uma análise estatística das falhas e certo conhecimento a respeito do ambiente de execução – o que pode não ser apropriado, uma vez que o ambiente possui características dinâmicas e, nesses casos, o de uso funções de distribuições de probabilidades adequadas não é uma tarefa simples.

Sendo assim, a implementação do gestor considera como alternativa uma análise do tempo de detecção supondo que a falha sempre em um cenário de pior caso, isto é: um processo monitorado p_j falha imediatamente depois de enviar um *heartbeat* para um processo monitor p_i.

[5] Ver Seção 2.4.1.3 para uma descrição mais detalhada de TD, TM e TMR.

Neste caso, p_i suspeitará de p_j quando alcançar o *timeout* de detecção do ciclo de monitoramento seguinte (ver Figura 4.6).

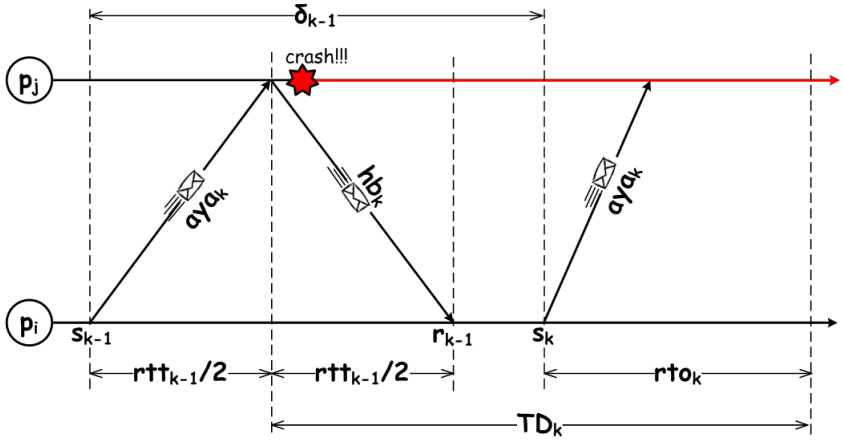


Figura 4.6. Cenário de pior caso para a estimativa de TD

Usando essa hipótese na estimativa do tempo de detecção, o gestor autonômico estima a falha de p_j em:

$$tcrash_k = r_{k-1} - \frac{rtt_{k-1}}{2}$$

em que $tcrash$ representa o suposto instante da falha do processo p_j.

Assim, p_i suspeitará da falha p_j em:

$$tsuspect_k = s_k + rto_k$$

Com isso, pode-se estimar o tempo de detecção por:

$$TD_k = tsuspect_k - tcrash_k \tag{4.4}$$

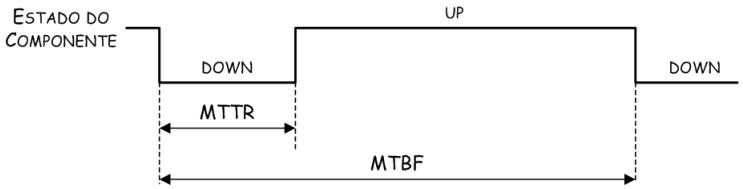
Observe, novamente, que a Equação 4.4 representa um modo simples para estimar o tempo de detecção – no qual o gestor autonômico também não precisa utilizar qualquer conhecimento a priori sobre o ambiente ou sobre o comportamento, em termos de estatísticas de falhas, do processo que está sendo monitorado.

4.3.1.2.2 Sensoriamento da confiabilidade da detecção

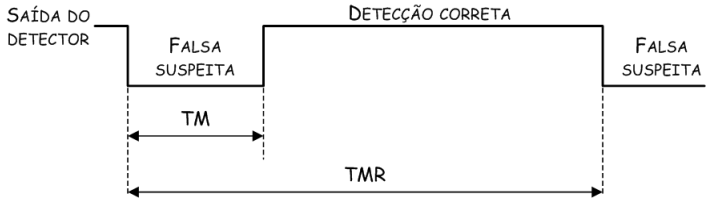
O sensoriamento da confiabilidade é importante para que o gestor autonômico concilie, durante a regulação do *timeout* de detecção, a relação de compromisso entre velocidade e a confiabilidade do serviço de detecção. Nesse sentido, o gestor precisa garantir *timeouts* curtos, mas ao mesmo tempo assegurar a confibilidade do detector mediante a imprecisão das estimativas do estimador de *timeout* quando condições dinâmicas do ambiente são consideradas.

Note que, o intervalo entre falsas suspeitas (TMR) e a duração das falsas suspeitas caracterizam de formas distintas a confiabilidade do detector. Enquanto TMR verifica a capacidade do detector em evitar falsas suspeitas (i.e. manter detecções livre de erros), TM avalia a capacidade do detector em corrigir rapidamente as falsas suspeitas cometidas (i.e. se recuperar de erros durante na realização das detecções). Um aspecto importante é conciliar essas duas medidas de modo a facilitar a atuação do gestor autonômico.

Fazendo um analogia entre TMR e TM e as métricas usadas na análise da confiabilidade de sistemas, é possível observar que TMR e TM se assemelham, respectivamente, ao tempo médio entre defeitos (i.e. $MTBF$, *Mean Time Between Failures*) e ao tempo médio de reparo (i.e. $MTTR$, *Mean Time To Repair*), ver Figura 4.7.



(a) Métricas de confiabilidade de componente



(b) Métricas de confiabilidade de detecção

Figura 4.7. Analogia entre (TMR, TM) e ($MTBF$ e $MTTR$)

Na análise da confiabilidade de sistemas, uma métrica que concilia ambas as métricas de confiabilidade é a disponibilidade. Assim, introduzimos nesta Tese o conceito de disponibilidade de detecção:

Definição 4.3 ***Disponibilidade de Detecção*** *(AV): é uma métrica operacional da confiabilidade do detector que avalia a capacidade de um módulo monitor do serviço de detecção de defeitos em, quando requisitado, entregar informações livres de falsas suspeitas.* ∎

Em Chen, Toueg e Aguilera (2002) é definida uma métrica similar, denominada Probabilidade de consulta correta (PA, *query accuracy probability*), entretanto, essa métrica demanda um conhecimento a priori da função de distribuição de probabilidade para que a mesma seja calculada corretamente. A disponibilidade de detecção, por sua vez, é uma métrica operacional, sendo dessa forma uma métrica que avalia o desempenho do serviço em tempo de execução. A rigor, TM e TMR também precisam de distribuições de probabilidade definidas para serem calculadas. Entretanto, por conta de uma coerência com o jargão adotado na literatura, a denominação dessas métricas são mantidas – contudo, nesta Tese, utiliza-se uma metodologia

diferente, da definida por Chen, Toueg e Aguilera (2002), e que permite que o cálculo de TM e TMR em tempo de execução. A seguir é discutido como a disponibilidade de detecção pode ser obtida.

A disponibilidade de um componente é determinada por:

$$AV = \frac{MTBF - MTTR}{MTBF}$$

De forma análoga, em um dado intervalo k, a disponibilidade média do módulo básico do serviço de detecção pode ser calculada pelo gestor autonômico usando:

$$AV_k = \frac{TMR_k - TM_k}{TMR_k} \tag{4.5}$$

O problema então é definir como usar a analogia adotada entre a disponibilidade do componente e a disponibilidade de detecção para obter TM e TMR e, dessa forma, calcular AV. Para tanto, observa-se que uma falsa suspeita pode ser comparada a um defeito de um módulo do serviço de detecção. Além disso, a cada intervalo k, o gestor autonômico calcula a duração de uma falsa suspeita (sd_k) e o número de falsas suspeitas (nf_k) por: $sd_k = r_k - (s_u - rtt_u/2)$ e $nf_k = nf_{k-1} + 1$, quando uma suspeita acontece; e $sd_k = 0$ e $nf_k = nf_{k-1}$, caso contrário. A variável s_u representa o instante de envio do último aya_u, para o qual um respectivo hb_u não foi recebido por um processo monitor p_i dentro do *timeout* de detecção rto_u.

As variáveis sd e nf podem ser associadas ao tempo de reparo de um defeito no serviço de detecção e ao número de reparos realizados no mesmo. Sendo assim, associando o valor médio de TM ao tempo médio para reparo $MTTR$, o gestor autonômico pode estimar TM em um dado intervalo k por:

$$TM_k = \frac{1}{nf_k}\left(\sum_{x=0}^{k} sd_x\right) \tag{4.6}$$

Se até o instante s_k, um módulo monitor cometeu nf_k falsas suspeitas, então o intervalo médio entre falsas suspeitas (ou o $MTBF$ do detector) é dado por:

$$TMR_k = \frac{s_k}{nf_k} \tag{4.7}$$

Desse modo, usando equações 4.6, 4.7 e 4.5 a disponibilidade de detecção pode ser encontrada.

Um último aspecto a ser tratado é conciliar os requisitos de confiabilidade de detecção definidas pelo usuário, usando TM^U e TM^L. Para tanto, é definido nesta Tese o conceito de disponibilidade mínima de detecção:

Definição 4.4 ***Disponibilidade mínima de detecção*** *(AV^L): é uma métrica de confiabilidade do detector que determina a menor expectativa do usuário em termos de disponibilidade de detecção, podendo ser calculada da seguinte forma:*

$$AV^L = \frac{TMR^L - TM^U}{TMR^L} \tag{4.8}$$

Observe que, a disponibilidade de detecção depende das magnitudes dos intervalos entre falsas suspeitas e das durações das falsas suspeitas. Assim, usando como base a Equação 4.5, quando o intervalo entre falsas suspeitas diminui, a disponibilidade de detecção também diminui. Da mesma forma, quando a duração da forma suspeita aumenta, a disponibilidade de detecção diminui. Uma vez que as métricas TMR^L e TM^U representam, respectivamente, a expectativa do usuário em termos do intervalo mínimo entre falsas suspeitas e da duração máxima da falsas suspeitas, esses limites implicam na disponibilidade mínima esperada pelo usuário – conforme definido na Equação 4.8.

4.3.2 Regulação do timeout de detecção

4.3.2.1 Discussão preliminar

Na detecção de defeitos em sistemas distribuídos, o *timeout* de detecção é um elemento chave, pois determina o prazo para que o processo monitorado responda antes que se torne suspeito de falha (ver Seção 2.4.1). Além disso, em ambientes com atrasos variados, qualquer estratégia de ajuste de *timeout* de detecção deve atender à relação de compromisso entre *velocidade* e *precisão* na detecção de defeitos. Isto é, *timeouts* de detecção (i.e. prazos) muito longos diminuem a ocorrência de falsas suspeitas, mas implicam em altas latências de detecção. Por outro lado, *timeouts* muito curtos, implicam em baixas latências de detecção, mas aumentam o número de falsas suspeitas.

Evidentemente que, em um ambiente com características dinâmicas e atrasos variados, qualquer estimador de atraso, usado no ajuste dos *timeouts* de detecção, cometerá erros – seja sugerindo estimativas maiores que o atraso observado (prejudicando o tempo de detecção), seja sugerindo estimativas menores que o atraso observado (prejudicando a confiabilidade do detector).

Com isso, as abordagens de detecção de defeitos, existentes na literatura, vêm privilegiando o atendimento ao requisito de confiabilidade[6], uma vez que, na grande maioria dos casos, as estratégias de ajuste de *timeout* usam estimadores que se adaptam as variações do atraso, mas utilizam alguma margem de segurança para prevenir falsas suspeitas – o que prejudica a velocidade da detecção.

Naturalmente, é razoável beneficiar a confiabilidade do detector, pois falsas suspeitas podem implicar em altos custos computacionais, oriundos da ativação de procedimentos de reconfiguração ou de recuperação, que demandam processamento e troca de mensagens adicionais.

[6] ver, por exemplo, Macêdo (2000), Chen, Toueg e Aguilera (2002), Bertier, Marin e Sens (2002), Falai e Bondavalli (2005), Satzger et al. (2007) etc.

Para reduzir o prejuízo ao tempo de detecção, alguns trabalhos têm usado margens de segurança adaptativas, em que a adaptação é realizada de acordo com as variações do atraso. A abordagem de Jacobson (1988), por exemplo, usa uma estratégia baseada em médias móveis para estimar (ambos) o atraso e sua variação. Então, a mesma ajusta o *timeout* de detecção como sendo a média do atraso mais duas vezes sua variação[7] – o que pode garantir uma taxa de acerto de cerca de $95,45\%$, contando ainda com latências de detecção menores em momentos de estabilidade (isto é, nos quais as variações do atraso são menores). A mesma consideração vale para estratégias como a de Bertier, Marin e Sens (2002), por exemplo. Esta estratégia também trabalha com média móvel (para a estimativa do atraso e da margem de segurança), considerando, entretanto, uma mobilidade de média menor que a usada em Jacobson (1988). Mais precisamente, enquanto Bertier, Marin e Sens (2002) consideram o histórico dos últimos 1000 atrasos, Jacobson (1988) baseia sua estimativa observando os dois últimos atrasos (ver Seção 2.4.1.4). Note que o tamanho do histórico (i.e. a mobilidade da média) determina a velocidade com que a estimativa converge, acompanhando as variações dos atrasos.

Quando as características do ambiente podem mudar, o desafio dessas estratégias é encontrar uma mobilidade de média que permita uma convergência adequada, atendendo ao compromisso entre velocidade e precisão das detecções. Isto é, se a estratégia confia em um histórico muito longo de atrasos (i.e. baixa convergência), a mesma pode demorar muito para responder às mudanças nas características do atraso (podendo, a depender da situação, prejudicar a velocidade ou a confiabilidade do detector)[8]. Por outro lado, se a estratégia confia em um histórico limitado de atrasos (i.e. convergência alta), a mesma estará mais suscetível a variações esporádicas do atraso (podendo cometer um maior número de falsas suspeitas) – como é verificado em muitos experimentos que usam o estimador de Jacobson (1988) para implementar o detector adaptativo.

Mais ainda, tentar estimar as características (i.e. distribuições) dos atrasos em tempo de execução, como é feito em Dixit e Casimiro (2010), por exemplo, pode implicar em soluções complexas e que, em alguns casos, desperdiçam recursos computacionais e não detectam a distribuição – ou quando detectam, a distribuição detectada não corresponde mais a distribuição que caracteriza o comportamento do ambiente.

4.3.2.2 O algoritmo proposto para a regulação de timeout

A proposta de regulação de *timeout*, apresentada nesta Tese, não apenas aproveita os benefícios trazidos pelas abordagens de adaptação de *timeout* existentes, mas também realiza correções nos *timeouts* sugeridos, considerando as mudanças dinâmicas nas características do ambiente e nos requisitos definidos pelo usuário.

[7] Estatisticamente, se os dados são normalmente distribuídos, a média mais duas vezes o desvio padrão abrange $95,45\%$ das dados – isto é conhecido como uma *regra de ouro*, nomeada *regra dos três sigma* ou *regra "*68-95-$99,7$*"*, ver Dai e Wang (1992).

[8] Neste caso, a confiabilidade da detecção é prejudicada quando o comportamento dos atrasos muda de uma variabilidade menos acentuada para uma variabilidade mais acentuada; e prejudica o tempo de detecção, caso contrário.

Para isto, esta proposta de regulação usa um controlador integral que monitora a confiabilidade do detector, a partir da disponibilidade de detecção (ver Seção 4.3.1.2). Então, sugere uma margem de segurança a ser adicionada ao *timeout* de detecção, de modo a atender à disponibilidade mínima definida pelo usuário, com um menor prejuízo para o tempo de detecção. Mais precisamente, se o detector é impreciso (i.e. a disponibilidade AV é menor que a mínima), então a margem de segurança α é incrementada para tornar a detecção mais precisa. Por outro lado, se a disponibilidade AV é alta (disponibilidade observada maior que a disponibilidade mínima), então a margem de segurança α é decrementada para tornar a detecção mais rápida.

Para não comprometer o desempenho do detector, é importante estabelecer os limites para a margem de segurança (α) sugerida pelo regulador de *timeout*. No estilo de monitoramento *Pull*[9], o menor *timeout* de detecção (i.e. prazo) possível é o atraso de comunicação mínimo (d^L). Então, α deve ser menor ou igual a $TD^U - d^L$, de modo a não extrapolar o tempo máximo de detecção (TD^U) definido pelo usuário. Além disso, α deve ser maior ou igual a zero, para não induzir o detector a cometer falsas suspeitas.

Algoritmo 4.2: *Regulação do timeout de detecção*

1 *define* $\alpha_0 = 0$;
2 ***before event*** aya_k *sending* ***do***
3 $\quad$ *compute* $AV^L = \frac{TMR^L - TM^U}{TMR^L}$;
4 $\quad$ *compute* $AV_k = \frac{TMR_k - TM_k}{TMR_k}$;
5 $\quad$ *obtain* $error_k = AV^L - AV_k$;
6 $\quad$ *compute* $\alpha_k = \alpha_{k-1} + \delta_{k-1} * error_k$;
7 $\quad$ ***if*** $\alpha_k < 0$ ***then*** $\alpha_k = 0$;
8 $\quad$ ***if*** $\alpha_k > TD^U - d^L$ ***then***
9 $\quad\quad$ $\alpha_k = TD^U - d^L$;
10 $\quad$ ***end if***
11 $\quad$ *obtain* rto_k^C;
12 $\quad$ *compute* $rto_k = rto_k^C + \alpha_k$;
13 ***end event***

O Algoritmo 4.2 apresenta o procedimento usado na regulação do *timeout* de detecção. Antes do envio de cada mensagem de monitoramento, a disponibilidade mínima (AV^L) e a disponibilidade atual (AV_k) do detector são calculadas (Linhas 3–4). Então, baseado no desvio (*error*) entre AV^L e AV_k, o regulador calcula a ação de controle integral (Linhas 5–6), isto é:

[9] ver Seção 4.2.

$$\alpha_{k+1} = \sum_{i=0}^{k} (\Delta t * error_i) = \alpha_k + \Delta t * error_k \tag{4.9}$$

em que Δt representa o intervalo de tempo entre duas ativações consecutivas do procedimento (i.e. $\Delta t = \delta_{k-1}$).

Em seguida, o regulador limita a margem de segurança entre seus valores máximo e mínimo (Algoritmo 4.2, Linhas 7–9). Então, uma estratégia de adaptação de *timeout* é ativada para estimar o *timeout* (rto^C) de acordo com o atraso observado (Linha 11). Por fim, o procedimento define o *timeout* de detecção (rto) a ser usado pelo detector, i.e. a soma do *timeout* (rto^C), sugerido a partir dos atrasos, com a margem de segurança (α), obtida a partir da disponibilidade (Linha 12).

Note que o mecanismo de regulação de *timeout* não faz restrições com relação à estratégia de adaptação de *timeout*, baseada em atraso, que será usada para determinar rto^C. Nesse sentido, o regulador de *timeout* encapsula a estratégia de adaptação de *timeout* usada por um detector adaptativo.

O diagrama em blocos da Figura 4.8 representa o laço de controle implementado pelo mecanismo de regulação do *timeout* – em que: (i) as entradas do regulador são, respectivamente, os requisitos de qualidade de serviço de detecção definidos pelo usuário (i.e. TMR^L, TM^U e TD^U); (ii) os requisitos TMR^L e TM^U são usados para definir o *set-point* do controlador, em termos da disponibilidade mínima de detecção (AV^L); (iii) o tempo máximo de detecção (TD^U) é usado para limitar a margem de segurança; (iv) o desempenho do detector, em termos da disponibilidade de detecção, é obtido a partir de um componente de sensor que implementa a atividade de sensoriamento da QoS de detecção (ver Seção 4.3.1.2); e (v) o *timeout* de detecção, sugerido pelo mecanismo de regulação de *timeout*, é implantado no monitor do detector de defeitos.

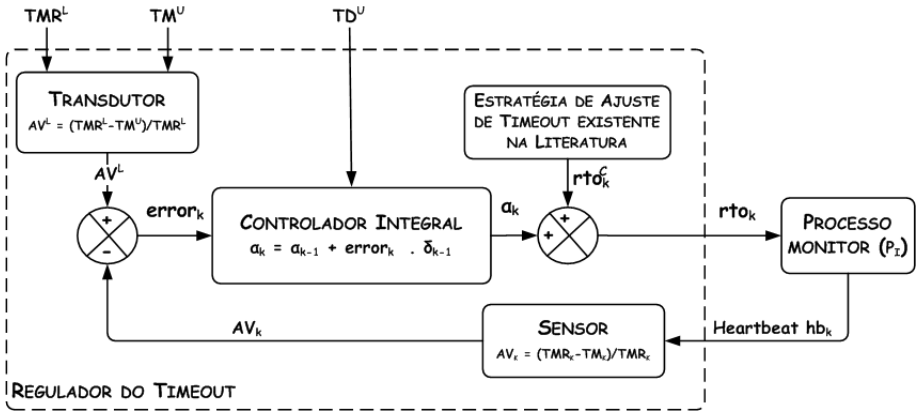


Figura 4.8. Diagrama em blocos do mecanismo de regulação do *timeout* de detecção

4.3.3 Regulação do período de monitoramento

4.3.3.1 Discussão preliminar

Na detecção de defeitos em sistemas distribuídos, o período de monitoramento é importante para determinar o intervalo entre as verificações do estado dos processos. Em ambientes com condições de carga variadas, o período de monitoramento deve ser escolhido de modo a atender à relação de compromisso entre custo computacional e desempenho do detector de defeitos.

Períodos de monitoramento muito longos reduzem o custo da detecção, uma vez que menos mensagens de monitoramento são transmitidas e processadas. Por outro lado, isto implica em detecções mais lentas e menos confiáveis. É evidente que quanto mais longo o período, menor é o número de falsas suspeitas, pois menos vezes o detector de defeitos realiza suposições a respeito do estado do processo monitorado. Entretanto, neste contexto, a detecção dos defeitos se baseia em uma informação de estado muito envelhecida e menos provável de corresponder ao estado mais recente. Períodos de monitoramento muito curtos podem elevar demasiadamente o custo computacional das detecções, podendo exaurir os recursos disponíveis e implicar em atrasos mais longos e variados. Isto também compromete o tempo de resposta das aplicações e diminui a eficiência e a velocidade dos detectores e dos demais mecanismos de tolerância a falhas.

Os períodos de monitoramento também comprometem a qualidade do ajuste dos *timeouts* de *detecção*. As estratégias de adaptação de *timeout* precisam de amostras dos atrasos de comunicação (ou de outra informação do ambiente) para realizar suas estimativas. Portanto, menos precisos são os *timeouts* estimados, quando períodos longos são usados – uma vez que, isto implica em um número menor de amostras coletadas.

Por tudo isso, a escolha de períodos de monitoramento apropriados demanda um conhecimento dos recursos disponíveis (Mills et al., 2004) ou algum conhecimento sobre a distribuição de carga de trabalho das aplicações (Chen; Toueg; Aguilera, 2002) – o que é um desafio em ambientes distribuídos dinâmicos ou abertos. Isto é, em ambientes abertos é muito difícil determinar a quantidade de recursos disponíveis em um dado instante, como ocorre em Mills et al. (2004). Além disso, se as características do ambiente mudam, é muito difícil determinar a quantidade de recursos disponível a partir de distribuições de probabilidade específicas, como é feito em Chen, Toueg e Aguilera (2002).

4.3.3.2 As abordagens de regulação de período propostas

A regulação de período de monitoramento, proposta nesta Tese, visa possibilitar detecções mais rápidas, confiáveis e com custo computacional adequado, considerando ambientes com características dinâmicas e requisitos de usuário que podem mudar.

Dada a dificuldade em se determinar com precisão os recursos disponíveis a cada instante, a proposta de regulação de período explora o atraso de interação fim-a-fim como uma metáfora

para a determinação do percentual de uso (ou consumo) dos recursos disponíveis a cada instante.

O conceito básico, associado a esta metáfora, é que quanto maior a carga das aplicações, dos mecanismos de tolerância a falhas ou do próprio detector de defeitos maior é o consumo dos recursos, significando maior troca de mensagens, maior processamento, mais uso de memória e, conseqüentemente, atrasos computacionais maiores. Do mesmo modo, uma mudança nas características do ambiente que leva a uma variação (i.e. aumento ou redução) do poder de processamento, da quantidade de memória ou da capacidade de transferência na rede, também tende a implicar em variações no percentual do uso dos recursos disponíveis[10] e nos atrasos computacionais.

Mais especificamente, a regulação do período usa um controlador o qual observa as variações nos atrasos de interação entre os processos. Em seguida, baseado nas variações de atrasos observadas, usa uma relação previamente definida para estimar o percentual de consumo de recursos (*RC*) no ambiente. Então, a partir do desvio entre os percentuais de consumo de recursos estimado e o desejado pelo usuário, o controlador usa uma lei de controle para determinar que variação deve ser realizada no período de monitoramento. Esse ajuste no período é realizado de modo a adequar o consumo de recursos do detector à fatia de recursos disponível no ambiente em um dado instante – buscando, assim, o menor período que não interfira no desempenho das aplicações ou dos demais mecanismos de tolerâncias a falhas.

4.3.3.3 Visão geral da implementação da proposta

Para atender ao objetivo proposto, o gestor autonômico implementa um laço de controle que considera três atividades:

(i) *caracterização do consumo de recursos no ambiente computacional* – refere-se a atividade de estabelecer relações que ajudem o gestor autonômico a inferir o consumo de recursos no ambiente em um dado instante. Uma vez que os atrasos das mensagens de monitoramento são a única informação disponível para o gestor autonômico, deve-se estabelecer uma relação entre tais atrasos e o consumo de recursos e deste último com o período de monitoramento.

(ii) *definição da lógica do laço de controle* – é uma atividade na qual se determina quais ações de controle (ou ajustes) devem ser realizadas sobre o período de monitoramento quando o consumo de recursos no ambiente se afasta do desejado.

(iii) *projeto e sintonia do controlador* – é uma atividade relacionada à definição do algoritmo de controle e seus respectivos parâmetros.

Conforme discutido anteriormente, assume-se que as características do ambiente são desconhecidas e mudam dinamicamente. Desse modo, o projeto do mecanismo de regulação de

[10]Note que, se a carga computacional se mantém constante, mas acontece uma variação na quantidade nominal dos recursos disponíveis, também existirá uma variação relativa no percentual de consumo de recursos.

período trata o ambiente como um sistema em caixa preta e usa as variações nos atrasos[11] como uma estimativa para o percentual do consumo de recursos do ambiente computacional.

Apesar de os atrasos poderem variar de forma não determinística, a proposta de regulação de período usa equações lineares para modelar o consumo de recursos e estimar o relacionamento entre o atraso e o período de monitoramento. Esse modelo baseado em equações lineares é uma aproximação e não descreve de forma apropriada o problema da caracterização do comportamento do ambiente computacional em termos de consumo de recursos. Entretanto, tal modelo é uma boa ferramenta para a descrição das questões relacionadas ao problema de controle, à definição do comportamento dinâmico do ambiente distribuído e ao projeto da lei de controle.

Para contornar as limitações impostas pela modelagem baseada em equações lineares, foi projetada uma lei de adaptação para a sintonia dos parâmetros do modelo, a qual permite que o modelo considerado se ajuste às mudanças nas características do ambiente computacional.

Para a regulação de período de monitoramento do detector de defeitos, duas propostas foram implementadas, usando as atividades e considerações descritas acima:

- *RBL* (***R**egulation **B**ased on **L**ittle's Laws*) – se baseia em algumas leis operacionais da teoria das filas propostas e originadas das leis de Little (1961) para modelar o comportamento dinâmico do ambiente computacional[12].

- *RBS* (***R**egulation **B**ased on **S**imple Linearization*) – realiza a modelagem baseada na linearização do comportamento dos recursos percebidos a partir dos atrasos[13].

Os aspectos relacionados ao projeto e a implementação das propostas de regulação de período de monitoramento (i.e. *RBL* e *RBS*) são discutidos com maiores detalhes nas subseções a seguir.

4.3.3.4 Proposta 1 - *RBL*: Regulação baseada nas leis de *Little*

4.3.3.4.1 Caracterização do consumo de recursos no ambiente

A proposta de regulação de período *RBL* abstrai o ambiente como uma caixa preta e estima o que seria o número médio de mensagens em fila a partir dos atrasos fim-a-fim observados pelo processo de sensoriamento do gestor autonômico (ver Figura 4.9).

O número médio de mensagens em fila ajuda o gestor autonômico a estimar o quanto de recurso está sendo consumido pelas aplicações que compartilham o ambiente em conjunto com o módulo do detector de defeitos que está sendo controlado. Para tanto, uma mensagem de monitoramento possui tamanho fixo e precisa de no mínimo $rtt^L = 2 * d^L$ unidades de tempo para ser processada pelo ambiente computacional (i.e. envio de *aya* e recebimento de um *hb*), ver Figura 4.9. Se é observado um atraso de $rtt^E = 2 * d^E$ no processamento de uma mensagem

[11] i.e. d, d^L, d^U e j^U – ver Seção 4.3.1.

[12] A abordagem *RBL*, proposta neste trabalho, foi apresentada em (Sá; Macêdo, 2010a)

[13] A abordagem *RBS*, proposta neste trabalho, foi apresentada em (Sá; Macêdo, 2010b).

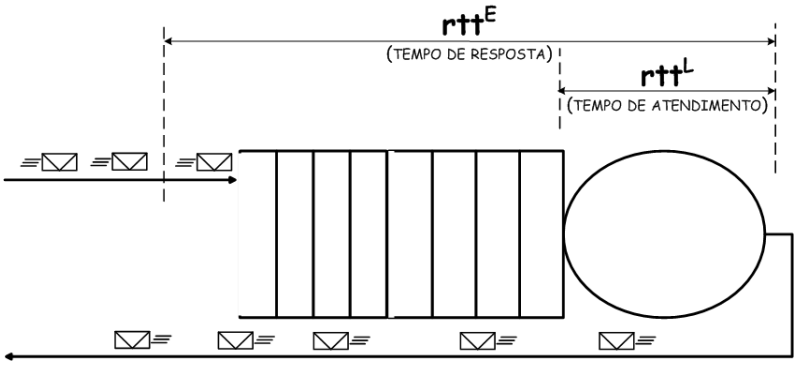


Figura 4.9. Modelo de filas suposto pelo gestor autonômico

de monitoramento (Figura 4.9), então, o número médio de mensagens sendo processados no sistema pode ser estimada, usando a lei de Little (1961), por:

$$M = \frac{rtt^E}{rtt^L} \tag{4.10}$$

Se quando um processo p_i insere uma mensagem no sistema, o mesmo observa um atraso médio (na resposta) equivalente a M mensagens, então essa mensagem esperou em fila $Q = M - 1$ mensagens serem processadas. Assim, usando a Equação 4.10, é possível obter:

$$Q = \frac{rtt^E}{rtt^L} - 1 \tag{4.11}$$

ou,

$$Q = \frac{rtt^E - rtt^L}{rtt^L} \tag{4.12}$$

Com isso, o percentual de utilização (RC) do sistema pode ser estimada usando:

$$RC = \frac{Q}{M^U} \tag{4.13}$$

em que M^U representa o número máximo de mensagens no sistema em um intervalo de observação T.

Uma vez que o tempo de resposta mínimo do sistema é rtt^L, então assumindo um intervalo de observação $T = \delta$, estima-se que o sistema é capaz de processar:

$$M^U = \frac{\delta}{rtt^L} \tag{4.14}$$

Assim, o percentual de utilização do sistema (RC) pode ser reescrita usando as equações 4.12 e 4.14:

$$RC = (rtt^E - rtt^L) * \lambda \tag{4.15}$$

em que $\lambda = 1/\delta$ é a freqüência de monitoramento.

Observe que a utilização, tal qual apresentada na Equação 4.15, não representa a utilização real, mas apenas uma estimativa vista pelo gestor autonômico instalado em um módulo monitor do detector de defeitos. Entretanto, tal informação serve como um indicativo para o consumo de recursos. Isto porque a Equação 4.15 consegue estabelecer uma relação entre a carga no ambiente a partir dos atrasos fim-a-fim estimados. Isto é, possíveis variações na carga no ambiente computacional implicam em possíveis variações no atraso fim-a-fim estimado rtt^E, logo, segundo a Equação 4.15, o percentual consumo de recursos varia. Além disso, possíveis variações na freqüência de monitoramento implicam em variações na carga do detector, portanto o percentual de consumo de recursos também varia.

Usando a Equação 4.15, a taxa de variação (i.e. derivada) de *RC* em função de λ pode ser calculada por:

$$\frac{\Delta RC}{\Delta \lambda} = (rtt^E - rtt^L) \tag{4.16}$$

Do ponto de vista de um processo monitor p_i, se o gestor autonômico observa a variação de *RC* em função da variação de λ, então é possível calcular o relacionamento entre a entrada ($u = \Delta\lambda$) e a saída ($y = \Delta RC$) do sistema usando um modelo linear ARX[14]:

$$y_{k+1} = y_k + (rtt^E - rtt^L) * u_k \tag{4.17}$$

Observe que o efeito da variação da freqüência de monitoramento em um intervalo k só tem efeito sobre a variação dos recursos, vistos pelo gestor autonômico em p_i, em $k+1$.

O mecanismo de regulação de período observa a saída da planta y e atua sobre a entrada u e estas são as únicas variáveis vistas pelo mesmo. Entretanto, conforme ilustrado na Figura 4.10, desde a entrada (u) a até a saída (y) existem vários elementos envolvidos. De acordo com a Figura 4.10, mensagens de monitoramento são coletadas pelo elemento sensor, o qual extrai os atrasos d^L e d^E, conforme discutido na Seção 4.3.1, e então um dispositivo transdutor usa a Equação 4.15 para determinar o consumo de recursos.

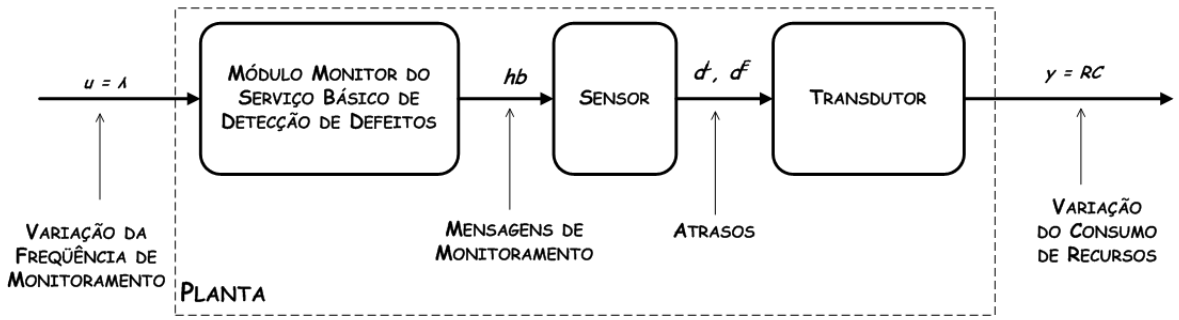


Figura 4.10. Diagrama em blocos da planta, em malha aberta, vista pelo regulador de período

[14]Para uma discussão mais detalhada do uso do modelo ARX em teoria de controle, ver Hellerstein et al. (2004) e Astrom e Wittenmark (1995).

4.3.3.4.2 Lógica do laço de controle

O problema de controle é regular λ de modo a obter um menor tempo de detecção, usando uma fração do percentual de recursos disponíveis no ambiente a cada instante. Assim, seja $RC^D \in (0,1]$ o percentual de recursos que pode ser consumida por um módulo monitor do detector de defeitos – em que RC^D é definido pelo usuário. Então, o percentual de consumo de recursos atual RC_k é comparada a RC^D e a diferença $error_k = RC^D - RC_k$ é calculada. Quando $error_k > 0$, a utilização está abaixo do limite especificado, logo λ é incrementado para permitir uma detecção mais rápida. Se $error_k < 0$, a utilização está acima do limite desejado, logo λ é decrementado.

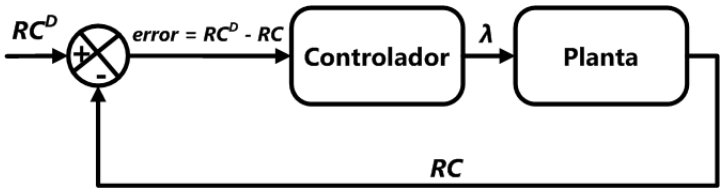


Figura 4.11. Laço de controle implementado pelo regulador *RBL*

A Figura 4.11 ilustra o laço de controle (i.e. malha fechada) implementado pelo mecanismo de regulação *RBL*.

4.3.3.4.3 Projeto e sintonia do controlador

O gestor autonômico utiliza um controlador P (Hellerstein et al., 2004), no qual a ação de controle (variação de λ) é proporcional ao desvio entre o valor desejado (RC^D) e o valor obtido (RC) para o consumo de recursos, ou seja:

$$\Delta\lambda = Kp * (RC^D - RC_k)$$

ou,

$$u_k = Kp * error_k \tag{4.18}$$

A sintonia do controlador diz respeito a encontrar o valor do ganho (ou parâmetro) proporcional Kp de modo a atender aos seguintes requisitos de desempenho(Hellerstein et al., 2004): estabilidade; precisão; tempo de convergência (*settling time*); e máximo sobre-sinal (*overshoot*).

Uma vez que, o comportamento do ambiente muda, os requisitos de desempenho do controlador não podem ser garantidos com um controlador linear (P). Além disso, o projeto do controlador considera a escolha de intervalos fixos de tempo, dito período de amostragem, nos quais o controlador observa o valor do consumo de recursos e então atua sobre a freqüência de monitoramento (λ). 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, o projeto tradicional baseado em funções de transferências descritas pela transformada $\mathcal{Z}$ é uma ferramenta válida apenas se o período de amostragem é fixo, portanto, qualquer análise utilizando funções de transferência em $\mathcal{Z}$ também não será válida. Para contornar tal problema, utiliza-se uma configuração inicial do controlador P, usando funções de transferência em $\mathcal{Z}$, e então aplica-se uma lei de adaptação para o ganho Kp, de modo a atender ao menos o requisito de estabilidade. Sendo assim, a configuração inicial usa as equações 4.17 e 4.18 para obter as funções de transferências:

$$P(z) = \frac{rtt^E - rtt^L}{z-1} \quad e \quad C(z) = K_p$$

em que $P(z)$ e $C(z)$ representam o comportamento da planta e a lei de controle P, respectivamente.

Então, usa-se $P(z)$ e $C(z)$ para definir a função de transferência em malha fechada:

$$F_r(z) = \frac{C(z) * P(z)}{1 + C(z) * P(z)}$$

A estabilidade é garantida se $1 + C(z) * P(z) = 0$, ou seja: $z - 1 + Kp * (rtt^E - rtt^L) = 0$. Usando a técnica de localização dos pólos em malha fechada (Hellerstein et al., 2004) e definindo um máximo sobre-sinal $M_p = 1 - RC^D$, é possível calcular:

$$Kp_k = \begin{cases} 1, & se\ rtt_k^E = rtt^L \\ \dfrac{RC^D}{rtt_k^E - rtt_k^L}, & se\ rtt_k^E > rtt^L \end{cases} \tag{4.19}$$

O Apêndice A apresenta como a função de transferência para o modelo da planta é obtida (Seção A.1.1) e como máximo sobre-sinal (M_p) e a lei de adaptação para Kp são encontrados (Seção A.1.2).

4.3.3.4.4 Algoritmo de regulação de período

Finalmente, o procedimento de regulação de período, usando a abordagem *RBL*, é descrito no Algoritmo 4.3.

No Algoritmo 4.3, λ^L e λ^U representam as freqüências de monitoramento mínima e máxima, respectivamente. Estes limites são usados para evitar que os valores das freqüências determinem períodos de monitoramento que extrapolem os limites desejados. Tais limites são calculados a partir do tempo de detecção mínimo (TD^L) estimado e do tempo de detecção máximo (TD^U) especificado pelo usuário. Uma discussão mais detalhada sobre os valores limites para λ é encontrada na Seção A.1.5 do Apêndice A.

O diagrama em blocos da Figura 4.12 representa o laço de controle implementado pelo mecanismo de regulação de período *RBL*. Nesta figura: (i) o regulador *RBL* recebe como entrada RC^D e TD^U; (ii) a partir do comportamento dos *heartbeats*, o sensor calcula o percentual

Algoritmo 4.3: *Regulação de período de monitoramento (proposta RBL)*

1 ***before event*** hb_k *sending* ***do***
2 *use the Equation* 4.19 *to compute* Kp_k*;*
3 *use the Equation* 4.15 *to compute* RC_k*;*
4 *obtain* RC^D*;*
5 *obtain* $error_k = RC^D - RC_k$*;*
6 *compute* $\lambda_k = \lambda_{k-1} + Kp * error_k$*;*
7 ***if*** $\lambda_k < \lambda^L$ ***then*** $\lambda_k = \lambda^L$*;*
8 ***if*** $\lambda_k > \lambda^U$ ***then*** $\lambda_k = \lambda^U$*;*
9 $\delta_k = \frac{1}{\lambda_k}$*;*
10 ***end event***

de consumo de recursos atual (RC_k); (iii) o desvio entre o percentual atual e o desejado (i.e. *error*) é enviado para o controlador proporcional, o qual estima o período de monitoramento; (iv) o tempo de detecção máximo (TD^U) é usado para determinar o limite superior do período de monitoramento.

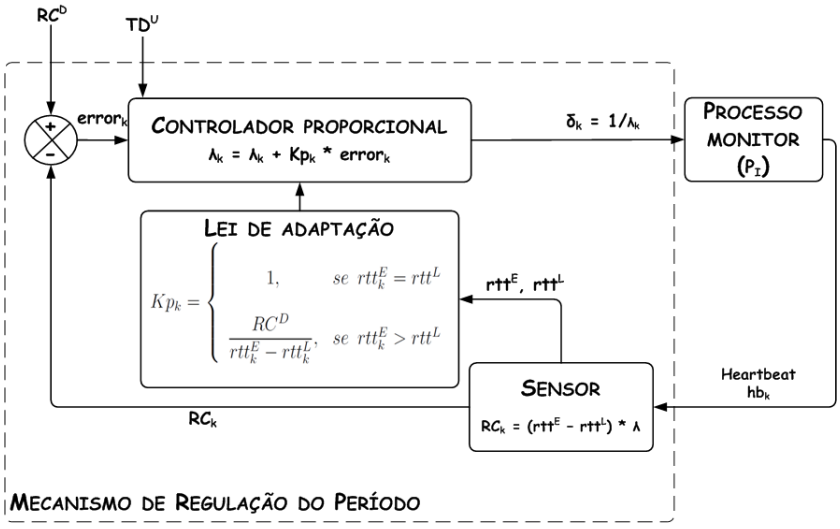


Figura 4.12. Diagrama em blocos do mecanismo de regulação de período *RBL*

4.3.3.5 Proposta 2 - *RBS*: Regulação baseada em linearização simples

4.3.3.5.1 Caracterização do consumo de recursos no ambiente

A abordagem *RBS* abstrai o ambiente como um sistema caixa preta (BBS, *Black-Box System*). Assim, assumindo essa modelagem de serviço baseado em caixa-preta, pode-se considerar que processo monitor submete uma requisição de serviço (i.e. uma mensagem de "*are you*

alive?") e recebe uma resposta a requisão (i.e. uma mensagem de *heartbeat* com o estado do processo monitorado) de BBS[15].

A idéia básica nesta abordagem é determinar a relação entre os tempos de respostas de BBS e os períodos de monitoramento do detector, usando equações lineares. Para tanto, considera-se que uma variação no tempo de resposta equivale a uma variação proporcional no percentual de consumo de recursos em BBS. Da mesma forma, uma variação no período de monitoramento equivale a uma variação proporcional no percentual do consumo de recursos relacionado ao custo da detecção em BBS. O gestor autonômico conhece apenas BBS, ignorando a existência dos demais processos ou aplicações que usem o serviço de BBS. Portanto, o mesmo supõe que o percentual de consumo de recursos estimados a partir da variação dos tempos de respostas e aquele estimado a partir da variação do período de monitoramento sejam equivalentes. Então, a partir dessa equivalência, obtém-se a relação entre os períodos de monitoramento e os tempos de respostas de BBS. Todos esses pontos são apresentados com mais detalhes a seguir.

Para realizar uma estimativa da capacidade de execução de BBS, o gestor autonômico assume o atraso mínimo (d^L) como uma indicação do tempo de serviço de BBS. Este tempo de serviço é mais tarde usado como uma referência para estimar o percentual de consumo de recursos do sistema. Similarmente à abordagem *RBL*, nesta modelagem baseada em caixa preta, o gestor autonômico assume d^E como uma indicação do tempo de resposta atual de BBS, e como tal, o tempo de resposta varia de acordo com o número de requisições de serviço.

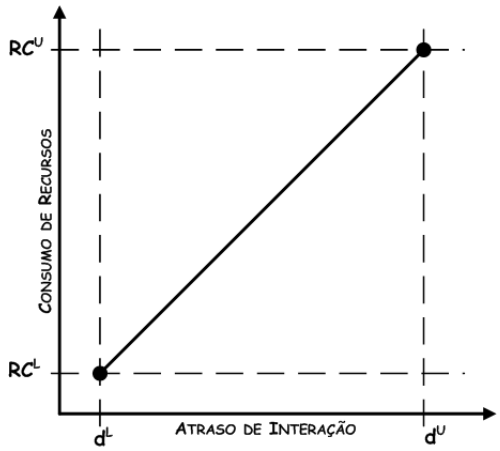


Figura 4.13. Relação entre o atraso de iteração e o consumo de recursos

Assim, o gestor autonômico supõe que a variação do percentual de consumo de recursos em função da variação do atraso (i.e. tempo de resposta) é linear, podendo ser descrita conforme a reta (ver Figura 4.13):

$$RC_k = \frac{RC^U - RC^L}{d^U - d^L} * (d_k^E - d^L) + RC^L \tag{4.20}$$

em que RC^U e RC^L representam, respectivamente, os percentuais de consumo de recursos má-

[15]Note que essa abstração é bastante similar aquela usada na abordagem *RBL*.

ximo e mínimo. Observe que, uma vez que $RC \in [0,1]$, é possível assumir $RC^U = 1$ e $RC^L = 0$.

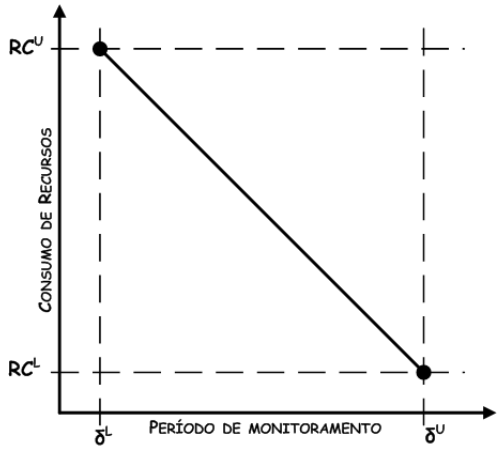


Figura 4.14. Relação entre o período de monitoramento e o consumo de recursos.

Além disso, o gestor autonômico também assume que a variação do percentual de consumo do recursos influenciado pelo período de monitoramento também é uma reta (ver Figura 4.14). Deste modo, o percentual de consumo de recursos em função do período é dado por:

$$RC_k = \frac{RC^U - RC^L}{\delta^U - \delta^L} * (\delta_{k-1} - \delta^L) + RC^L \tag{4.21}$$

em que δ^U e δ^L são, respectivamente, os períodos de monitoramento máximo e mínimo – ambos obtidos a partir do tempo de detecção mínimo estimado e do tempo de detecção máximo especificado pelo usuário[16].

Igualando as equações 4.20 e 4.21:

$$d_k^E = \frac{\delta^U - \delta_{k-1}}{\delta^U - \delta^L} * (d^U - d^L) + d^L \tag{4.22}$$

Calculando a taxa de variação (i.e. derivada) de d^E em função de δ, tem-se:

$$\frac{\Delta d^E}{\Delta \delta} = -\frac{d^U - d^L}{\delta^U - \delta^L} \tag{4.23}$$

Em relação a um processo monitor p_i, quando se observa o comportamento da variação de d no BBS em relação a variação de δ, é possível calcular o relacionamento entre a entrada ($u = \Delta\delta$) e a saída ($y = \Delta d^E$) do BBS usando um modelo linear ARX:

$$y_{k+1} = y_k - \left(\frac{d^U - d^L}{\delta^U - \delta^L}\right) * u_k \tag{4.24}$$

Na Seção A.1.3 do Apêndice A podem ser encontrados maiores detalhes sobre o modelo ARX apresentado na Equação 4.24.

[16] Veja o Apêndice A para uma discussão sobre estes limites de δ.

4.3.3.5.2 *Lógica do laço de controle*

Sendo $RC^D \in (0,1)$ o percentual de consumo máximo de recursos desejado em relação a p_i. Então, o gestor autonômico calcula a diferença entre RC^D e RC_k (Equação 4.20), isto é $error_k = RC^D - RC_k$. Quando $error_k > 0$, RC_k está abaixo do esperado, então δ é decrementado para permitir uma detecção mais rápida. Quando $error_k < 0$, RC_k é maior que o esperado, então δ é incrementado para consumir menos recursos e evitar falsas suspeitas.

4.3.3.5.3 *Projeto e sintonia do controlador*

A regulação do período na proposta *RBS* é realizada usando um controlador Proporcional-Integral (PI) (Hellerstein et al., 2004). Este controlador produz uma ação de controle (variação de δ) proporcional à diferença entre RC^D e RC_k, somada a uma ação de controle referente à integração de tal diferença no tempo. Um controlador PI usa a seguinte lei de controle (Ogata, 1995):

$$u_k = Kp * error_k + Ki * \Delta t * \sum_{x=0}^{k-1} error_x \tag{4.25}$$

em que Δt é o intervalo de tempo entre a ativação atual e a última ativa do controlador e Kp e Ki são os ganhos proporcional e integral do controlador, respectivamente.

Assim como no caso da proposta *RBL*, a sintonia do controlador PI consiste em encontrar os valores dos ganhos Kp e Ki de modo a atender aos requisitos de desempenho do controle. As mesmas considerações feitas para a sintonia do controlador *P*, são válidas também para a sintonia do controlador *PI*. Portanto, para este controlador também utiliza uma lei de adaptação para os ganhos Kp e Ki.

A configuração inicial usa as equações 4.24 e 4.25 para obter as funções de transferência em $\mathcal{Z}$:

$$P(z) = \frac{\left(\frac{d^U - d^L}{\delta^U - \delta^L}\right)}{z-1} \quad e \quad C(z) = K_P + K_I * \Delta t * \frac{z}{z-1}$$

em que $P(z)$ e $C(z)$ representam o comportamento de BBS e a lei de controle PI, respectivamente – usa-se $P(z)$ e $C(z)$ para definir a função de transferência em malha fechada $F_r(z) = \frac{C(z)*P(z)}{1+C(z)*P(z)}$.

A partir destas definições, assume-se que o laço de controle atinge a saída desejada em um tempo de estabilização $K_s = d^U$. Mais ainda, assume-se um máximo sobre-sinal de 10% (i.e., $M_p = 0,1$) – o que representa uma escolha convencional para o sobre-sinal (Hellerstein et al., 2004).

Então, usa-se a técnica de localização dos pólos em malha fechada[17] para estimar os pólos complexos (cp) de $F_r(z)$ como:

[17] Ver Hellerstein et al. (2004).

$$cp = g * exp(\pm j\theta)$$

em que $g = exp(\frac{-4}{K_s})$ e $\theta = \pi * \frac{log(g)}{log(M_p)}$ são, respectivamente, a magnitude e o ângulo de cp.

Os ganhos K_P e K_I do controlador, usados na abordagem *RBS*, são definidos usando a lei de adaptação definida no Algoritmo 4.4.

Algoritmo 4.4: *Lei de adaptação dos parâmetros Kp e Ki (proposta RBS)*

1 ***if*** $d_k^L = d_k^U$ ***then***
2 $\quad \phi = 0;$
3 ***else***
4 $\quad \phi = \frac{1}{d_k^U - d_k^L};$
5 ***end if***
6 ***if*** $\delta_k^U = \delta_k^L$ ***then***
7 $\quad \psi = 1;$
8 ***else***
9 $\quad \psi = \frac{1}{\delta_k^U - \delta_k^L};$
10 ***end if***
11 *compute* $K_P = \frac{\phi - g^2}{\psi};$
12 *compute* $K_I = \frac{g^2 - 2 * g * cos(\theta) + 1}{\psi};$

Conforme discutido anteriormente, d^L e d^U variam em função do tempo, sendo assim, o posicionamento dos pólos cp mudam também. A lei de adaptação proposta ajusta Kp and Ki para manipular essas variações e melhorar o desempenho do laço de controle.

O Apêndice A apresenta maiores detalhes sobre a lei de adaptação para os ganhos Kp e Ki (Seção A.1.4) e sobre o modelo matemático da planta (Seção A.1.3).

4.3.3.5.4 Algoritmo de regulação do período

O procedimento de regulação de período, usado na abordagem *RBS*, é descrito no Algoritmo 4.5. Neste algoritmo, a função *currentTime*, linhas 2 e 4, retorna o instante atual considerando o relógio local do processo monitor p_i. Então, regulador de período usa o procedimento 4.4 para calcular K_P e K_I e usa a Equação 4.21 para obter a estimativa atual para o percentual de consumo de recursos (Linhas 7–9).

Em seguida, obtêm o percentual de consumo de recursos máximo desejado (RC^D) e calcula o desvio ($error_k$) entre RC^D e o consumo atual RC^k (Linhas 10 e 11). A partir do cálculo *error*, o gestor autonômico calcula as ações de controle proporcional (up) e integral (ui) – Linhas 12 e 13.

Por fim, o gestor autonômico faz uso das ações de controle para calcular o valor do período de monitoramento e, em seguida, limita tal período entre os valores máximo (δ^U) e mínimo (δ^L) permitidos – ver Linhas 14–16.

Algoritmo 4.5: *Regulação de período de monitoramento (proposta RBS)*

1 *define* $ui_0 = 0$;
2 *obtain* $last = currentTime()$;
3 ***before event*** hb_k *sending* ***do***
4 *obtain* $current = currentTime()$;
5 *compute* $\Delta t = current - last$;
6 *assign* $last = current$;
7 *use the Algorithm 4.4 to compute* Kp_k;
8 *use the Algorithm 4.4 to compute* Ki_k;
9 *use the Equation* 4.21 *to compute* RC_k;
10 *obtain* RC^D;
11 *obtain* $error_k = RC^D - RC_k$;
12 *compute* $up_k = Kp_k * error_k$;
13 *compute* $ui_k = ui_{k-1} + \Delta t * Ki_k * error_k$;
14 *compute* $\delta_k = \delta_k^L + (ui_k + up_k)$;
15 ***if*** $\delta_k > \delta_k^U$ ***then*** $\delta_k = \delta_k^U$;
16 ***if*** $\delta_k < \delta_k^L$ ***then*** $\delta_k = \delta_k^L$;
17 ***end event***

O diagrama em blocos da Figura 4.15 apresenta o laço de controle implementado pelo mecanismo de regulação de período *RBS*. Nesta Figura: (i) o regulador *RBS* recebe como entrada RC^D e TD^U; (ii) a partir do comportamento dos *heartbeats*, o sensor calcula o percentual de consumo de recursos atual (RC_k); (iii) o desvio entre o percentual atual e o desejado (i.e. *error*) é enviado para o controlador PI, o qual estima o período de monitoramento; (iv) o tempo de detecção máximo (TD^U) é usado para determinar o limite superior do período de monitoramento.

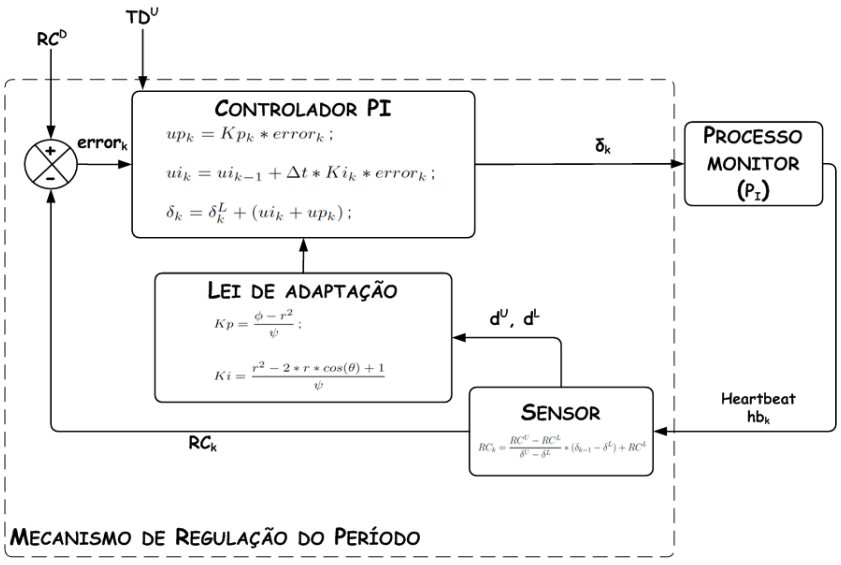


Figura 4.15. Diagrama em blocos do mecanismo de regulação de período *RBS*

4.4 AVALIAÇÃO DE DESEMPENHO

4.4.1 Descrição do ambiente de simulação

Os experimentos foram realizados por simulação com o auxílio do pacote *Matlab/Simulink/TrueTime 1.5*(Henriksson; Cervin, 2007). Para os experimentos, foram configurados 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 descartadas. As mensages trocadas entre os processos do ambiente têm tamanho fixo e igual a $1536bits$ – equivalente a três vezes o tamanho mínimo de um quadro *Ethernet*.

O processo em c_1 monitora defeitos do processo em c_2. Em c_3, um processo (denominado p_3) é utilizado para gerar rajadas aleatórias de trafego na rede. Essas rajadas são geradas de tal modo que a utilização média da rede é incrementada de 10% a cada $1000ms$ – sendo que a mesma retorna a zero após atingir 90%.

Para tanto, p_3 é ativado periodicamente a cada $0,1536ms$, i.e. 1536 bits dividido por $10Mbps$ – o que equivale ao tempo necessário para transmitir um quadro de 1536 bits. Uma variável $bw \in [0,1)$ é mantida por p_3, sendo a mesma inicializada com zero, incrementada de $\frac{1}{10}$ a cada $1000ms$ e retorna à zero após atingir $0,9$. Uma função geradora de números pseudo-aleatórios, dita *rand()* é chamada por p_3 a cada ativação. A função *rand()* retorna um valor uniformemente distribuído no intervalo $(0,1)$. Um quadro é enviado por p_3 quando o valor retornado pela função *rand()* é menor que *bw*.

O Algoritmo 4.6 ilustra o procedimento implementado por p_3 na geração das rajadas.

Algoritmo 4.6: *Geração de rajadas aleatórias*

```
1  τ = 1536 / (10 * 10^6);
2  bw = 0;
3  foreach (τ milliseconds) do
       /* t is the current time in milliseconds                    */
4      t = currentTime();
       /* mod(x, y) finds the remainder of division of x by y      */
5      bw = (mod(floor(t), 1000)) / 10;
6      r = rand();
7      if r < bw then
8          send(m);
9      end if
10 end
```

Essa geração de rajadas aleatórias permite avaliar os detectores de defeitos sob diferentes

condições de carga.

Por fim, as simulações são executadas até que tenham sido transferidas 10^4 mensagens de monitoramento.

4.4.2 Métricas de desempenho

O desempenho dos detectores foram verificados considerando as métricas: *Tempo de Detecção* (TD), *Duração da Falsa Suspeita* (TM), *Intervalo entre Falsas Suspeitas* (TMR) e *Disponibilidade de Detecção* (AV), vide Seções 2.4.1.3 e 4.3.1.2.

Além disso, utiliza-se também, como métrica de desempenho, a *Taxa de Falsas Suspeitas* (*RM*, *Rate of Mistakes*), a qual é obtida dividindo o número de falsas suspeitas (nf) pelo número total de verificações que o detector realiza a respeito do processo monitorado. Essa última métrica é mais justa que o uso do número de falsas suspeitas somente. Isto porque, detectores de defeitos com períodos de monitoramento mais longos verificam menos vezes o estado do processo monitorado. Portanto, tendem a cometer menos suspeitas por intervalo de tempo.

4.4.3 Configuração dos detectores

Nos experimentos realizados são considerados quatro detectores de defeitos:

- Detector autonômico baseado na abordagem *RBL*, denominado *AFD-RBL*;
- Detector autonômico baseado na abordagem *RBS*, denominado *AFD-RBS*;
- Detector adaptativo baseado no estimador de *timeout* de Jacobson (1988), denominado *Jacobson*;
- Detector adaptativo baseado no estimador de *timeout* de Bertier, Marin e Sens (2002), denominado *Bertier*;

Conforme discutido na Seção 4.3.2, as abordagens de detecção autonômicas *AFD-RBL* e *AFD-RBS* usam um estimador de *timeout* e realizam correções nas estimativas realizadas pelo mesmo quando necessário. Assim, nos experimentos, ambas as abordagens autonômicas usam o estimador de *timeout* de Jacobson (1988) e as correções são realizadas usando os procedimento apresentado na Seção 4.3.2 (ver Algoritmo 4.2). Além disso, *AFD-RBL* e *AFD-RBS* foram configurados da seguinte forma: $RC^D = 0,5$ (i.e. 50% do percentual dos recursos não utilizados pelas aplicações); $TD^U = 50ms$ (i.e. aproximadamente 20 vezes a latência mínima de transmissão de um quadro na rede); e $TM^U = 1ms$ e $TMR^L = 10000ms$ (i.e. $AV^L = 0,9999$). A Seção A.2 do Apêndice A apresenta um análise do impacto desses parâmetros no desempenho do detector autonômico.

Os detectores adaptativos, usando os estimadores de Bertier, Marin e Sens (2002) e de Jacobson (1988), foram configurados conforme originalmente proposto pelos autores. Nas com-

parações, consideram-se quatro configurações para os detectores adaptativos. Essas configurações usam período de monitoramento fixo e equivalentes a 1, 3 e 5 milissegundos, respectivamente. Esses valores de período de monitoramento foram selecionados experimentalmente, considerando os seguintes aspectos observados: (a) períodos de monitoramentos inferiores a $1ms$ foram utilizados, mas os mesmos apenas aumentaram a variabilidade dos atrasos na rede e o número de falsas cometidas pelos detectores adaptativos – sem trazer qualquer benefício para o tempo de detecção; (b) períodos superiores a $5ms$ apenas incrementaram o tempo de detecção, sem melhorar a precisão dos detectores adaptativos. Portanto, os períodos de monitoramento foram fixados dentro da faixa de 1 a $5ms$ para garantir uma comparação mais justa entre as abordagens de detecção autonômica e as abordagens de detecção adaptativa com períodos pré-fixados.

4.4.4 Resultados obtidos

As figuras de 4.16 a 4.20 apresentam o desempenho dos detectores de defeitos em termos das métricas consideradas e em diferentes condições de carga. Nessas figuras, os eixos x e y dos gráficos representam o tempo em milissegundos e a métrica considerada, respectivamente. Nas figuras 4.16, 4.17 e 4.19, TD, TM e TMR são representados em milissegundos. Nas figuras 4.17 e 4.19, os eixos y, referentes a TM e TMR, estão em escala logarítmica. Os gráficos da Figura 4.20, os quais se referem ao desempenho em termos da disponibilidade de detecção, possuem eixos y com escalas diferentes.

Em todas as figuras, os gráficos intitulados Bertier($\delta = x\,ms$) referem-se ao detector adaptativo implementado com o algoritmo de Bertier, Marin e Sens (2002) e configurado com período de monitoramento (δ) igual a x milissegundos. Da mesma forma, os intitulados Jacobson($\delta = x\,ms$) referem-se ao detector adaptativo implementado com o algoritmo de Jacobson (1988) e configurado com período de monitoramento igual a x milissegundos.

Na Figura 4.16, referente ao desempenho em termos do tempo de detecção, todos os eixos y dos gráficos estão na mesma escala (variando entre 0 e $8\,ms$), com exceção dos gráficos pertinentes a Bertier($\delta = 1\,ms$) e Jacobson($\delta = 1\,ms$), os quais possuem eixo y variando de 0 a $80\,ms$ – pois os tempos de detecção dos mesmos foram muito superiores aos demais e, usar uma escala comum, em todos os gráficos, não permitiria que os demais resultados fossem comparados de forma adequada.

Por fim, ao final desta seção, a Tabela 4.1 apresenta um resumo dos desempenhos médios obtidos, pelos detectores, durante os experimentos realizados.

4.4.4.1 Desempenho em termos do tempo de deteção (TD)

Em termos do tempo de detecção, Figura 4.16, Bertier($\delta = 1\,ms$) e Jacobson($\delta = 1\,ms$) apresentam baixos tempos de detecção para baixas condições de carga, mas eleva drasticamente o tempo de detecção quando considerado cenários com carga maiores que 70% – apresentando

tempos de detecção médios de $2,7ms$ e $3,0ms$, respectivamente.

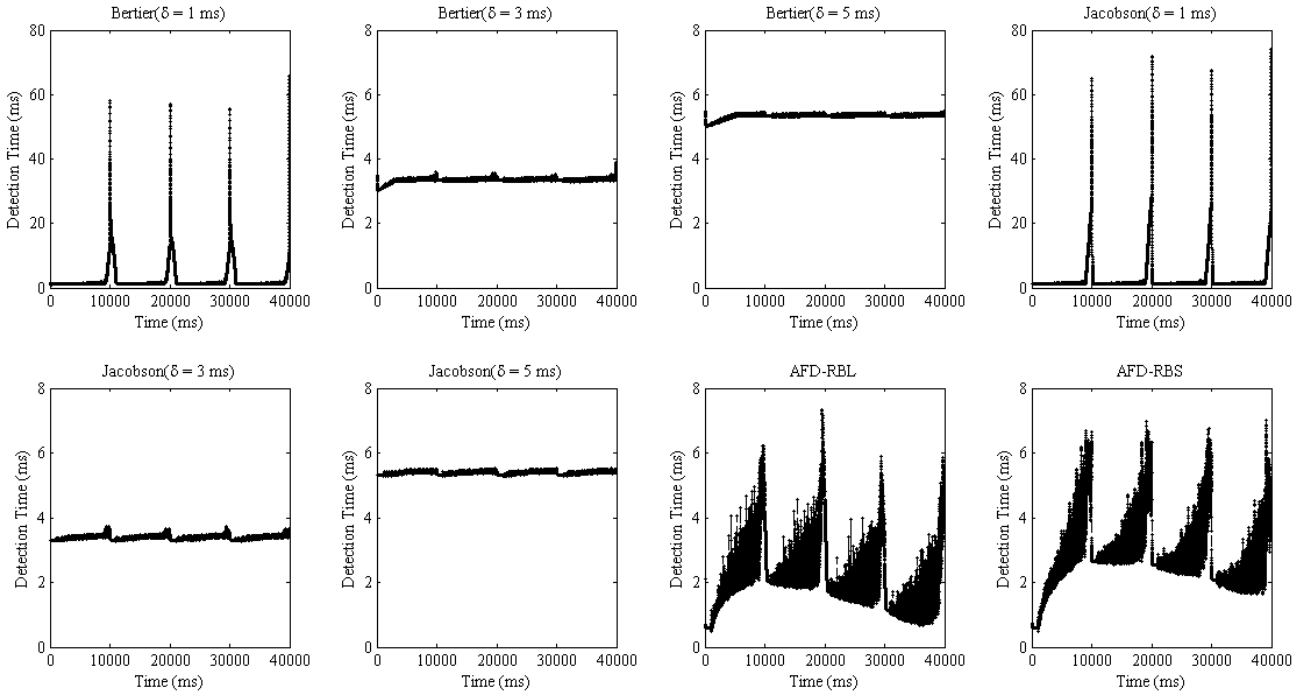


Figura 4.16. Desempenho em termos do tempo de detecção

Note que os detectores adaptativos com $\delta = 1ms$ consomem $\approx 30,72\%$ da banda da rede (isto é, uso de rede durante $\frac{1536bits}{10Mbs}$ segundos vezes 2 mensagens a cada $1ms$). Isto significa que, nos intervalos $[7000ms, 9000ms]$, $[17000ms, 19000ms]$, $[27000ms, 29000ms]$ e $[37000ms, 39000ms]$, a demanda de recurso de rede é maior que 100% (o que justifica os picos nos valores de TD observados em tais intervalos para esses detectores).

Os detectores adaptativos com $\delta = 3ms$ e $\delta = 5ms$, por sua vez, possuem tempos médios de detecção muito próximos aos seus respectivos valores de período de monitoramento – i.e., $3,4ms$ e $5,3ms$ para Bertier($\delta = 3\,ms$) e Bertier($\delta = 5\,ms$) e $3,4ms$ e $5,9ms$ para Jacobson($\delta = 3\,ms$) e Jacobson($\delta = 5\,ms$).

Os detectores autonômicos (*AFD-RBL* e *AFD-RBS*) possuem tempos de detecção muitos similares, sendo que estes tempos de detecção variam entre $0,5ms$ e $7,0ms$ dependendo da carga, apresentando valores médios de TD aproximadamente iguais a $1,7ms$ (*AFD-RBL*) e $2,5ms$ (*AFD-RBS*). Sendo assim, os detectores autonômicos *AFD-RBL* e *AFD-RBS* possuem desempenho médio em termos do tempo de detecção superior a todos os detectores adaptativos considerados, tendo *AFD-RBL* um desempenho médio um pouco melhor que *AFD-RBS*, em termos desta métrica.

O detector *AFD-RBS* possui um tempo médio de detecção superior ao *AFD-RBL* por conta do efeito da ação integral usada pelo mecanismo de regulação de período *RBS*. Tal ação integral faz com que o detector memorize o efeito das variações de carga passadas e sugira períodos de

monitoramento um pouco mais elevados que a abordagem de regulação de período *RBL*.

Os resultados apresentados demonstram que o ajuste dinâmico do período de monitoramento, realizado pelas abordagens de detecção autonômicas, permite acomodar adequadamente a carga gerada pelo detector aos recursos disponíveis a cada instante. Além disso, estas abordagens permitem tempos de detecção mais curtos em cenários com menor carga de aplicação. Nenhum desses aspectos é possível nas abordagens de detecção adaptativas com períodos de monitoramento fixos.

4.4.4.2 Desempenho em termos da duração da falsa suspeita (TM)

Em termos da duração das falsas suspeitas, Figura 4.17, os detectores adaptativos com $\delta = 1ms$ apresentaram o pior desempenho, com TM médios de $2,40ms$ e $0,49ms$ para $Bertier$ e *Jacobson*, respectivamente. Mais ainda, o detector Bertier($\delta = 1\,ms$), sob as condições de carga impostas, tem o seu desempenho prejudicado por conta do efeito de memória provocado pelo tamanho do histórico de mensagens utilizado em tal abordagem – i.e. $ws = 1000$, conforme originalmente proposto por Bertier, Marin e Sens (2002). Note que, nos intervalos de maior carga (i.e. $[7000ms, 9000ms]$, $[17000ms, 19000ms]$, $[27000ms, 29000ms]$ e $[37000ms, 39000ms]$, respectivamente), esse detector demora a perceber a mudança dos atrasos, pois a componente principal do *timeout* confia principalmente na média. Assim, no início desses intervalos, o mesmo dependerá basicamente do ajuste fino que é feito pela margem de segurança, calculada segundo a estratégia de Jacobson (1988) e que também sofre com a mudança de carga (ver Figura 4.16 e 4.17). Isto explica o comportamento observado no gráfico de TM para o detector adaptativo de $Bertier$ com $\delta = 1ms$.

Na medida em que o período de monitoramento cresce, a influência do mesmo sobre a variação da carga de trabalho na rede diminui. Assim, os estimadores de *timeout* podem realizar predições mais acuradas e, conseqüentemente, os detectores adaptativos corrigem as suas falsas suspeitas mais rapidamente – observe que ambos os estimadores de Jacobson (1988) e de Bertier, Marin e Sens (2002) usam margens de segurança adaptativas para sobreestimar os atrasos, desse modo, se a variabilidade da carga diminui, a influência dessas margens de segurança é menor.

Com isso, os detectores adaptativos com períodos $\delta = 3ms$ e $\delta = 5ms$ têm tempos de detecção e durações de falsas suspeitas menores que aqueles com $\delta = 1ms$. Sendo assim, Bertier($\delta = 3\,ms$) e Bertier($\delta = 5\,ms$) apresentam valores médios de TM de $0,10ms$ e $0,11ms$, respectivamente. Enquanto que, a média de TM para Jacobson com $\delta = 3ms$ e $\delta = 5ms$ foi de $0,04ms$ para ambas as configurações, sendo, desta forma, melhor que o detector adaptativo de Bertier em termos de tal métrica.

Os detectores autonômicos possuem períodos de monitoramento que se adaptam às diferentes condições de carga, de modo a minimizar o efeito de tal período sobre a carga da rede. Sendo assim, os mesmos induzem menos variações na carga, o que permite que os mesmos

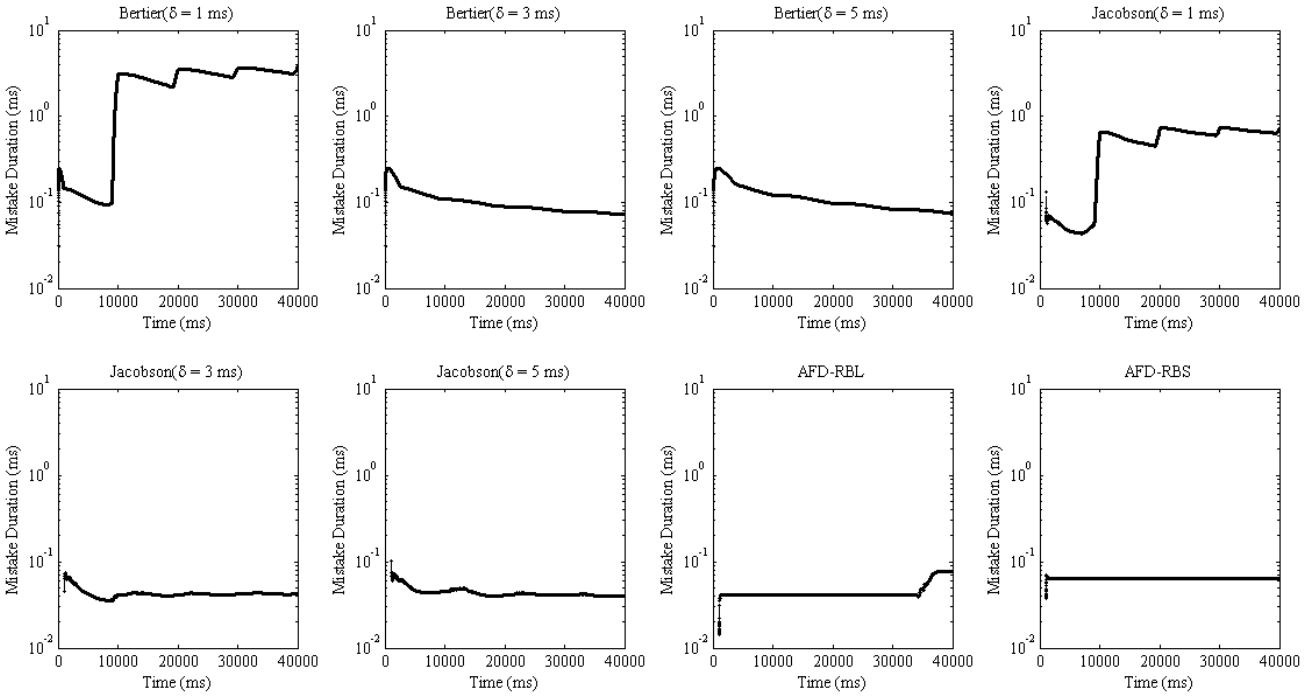


Figura 4.17. Desempenho em termos da duração da falsa suspeita

apresentem durações de falsas suspeitas mais estáveis. Com isso, os detectores autonômicos *AFD-RBL* e *AFD-RBS* apresentam durações média de falsas suspeitas de $0,04ms$ e $0,06ms$, respectivamente. Sendo que, conforme comentado anteriormente, o detector *AFD-RBS*, por conta da ação integral, responde mais lentamente, possuindo, assim, um desempenho médio um pouco inferior ao *AFD-RBL* em termos de TM.

Os resultados apresentados demonstram que o ajuste dinâmico do período de monitoramento, realizado pelas abordagens de detecção autonômicas, permite acomodar adequadamente a carga gerada pelo detector aos recursos disponíveis a cada instante. Conseqüentemente, estas abordagens também permitem durações de falsas suspeitas mais estáveis e curtos em cenários com menor carga de aplicação. Da mesma modo que observado para caso do tempo de detecção, nenhum desses aspectos é possível nas abordagens de detecção adaptativas com períodos de monitoramento fixos.

4.4.4.3 Desempenho em termos da taxa de falsas suspeitas (*RM*)

Em termos da taxa de falsas suspeitas, Figura 4.18, os detectores autonômicos apresentam um desempenho muito superior ao dos demais. Tais detectores apresentaram taxas de falsas suspeitas muito próximas de zero, mais precisamente $0,0013$ e $0,0007$ para *AFD-RBL* e *AFD-RBS*, respectivamente.

O bom desempenho dos detectores autonômicos, em termos da taxa de falsas suspeitas, se deve a estratégia de ajuste da margem de segurança, combinada com o uso de períodos de

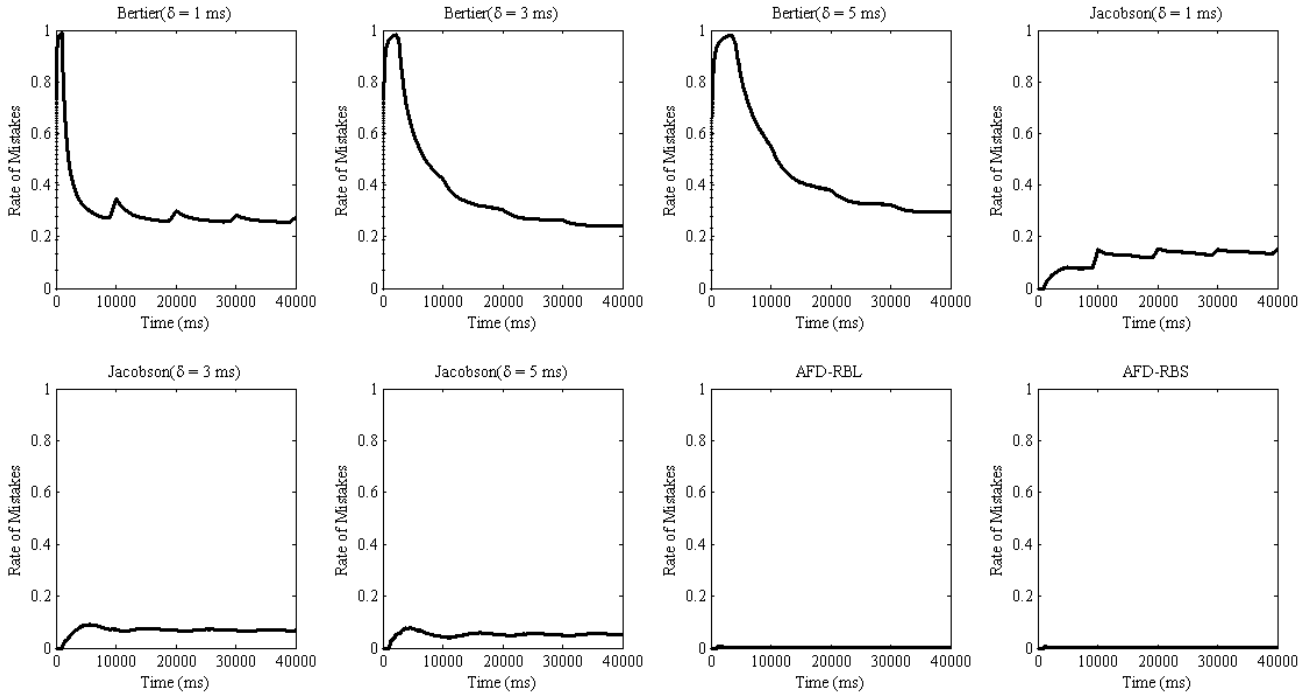


Figura 4.18. Desempenho em termos da taxa de falsas suspeitas

monitoramento mais longos e dinamicamente calculados para altas condições de carga na rede. Tal combinação resulta em atrasos de comunicação mais estáveis.

Os detectores adaptativos com o estimador de Bertier, Marin e Sens (2002) sempre apresentaram uma taxa de falsas suspeitas acima de $0,3$, para qualquer configuração considerada. Para tal detector, a taxa de falsas suspeitas média foi de $0,46$, $0,38$ e $0,30$ para δ igual a 1, 3 e $\delta = 5ms$, respectivamente.

Os detectores adaptativos com o estimador de Jacobson (1988), por sua vez, apresentaram taxas de falsas suspeitas abaixo de $0,2$, para todas as configurações consideradas. Para tal detector, a taxa de falsas suspeitas média foi de $0,11$, $0,06$ e $0,05$ para $\delta = 1ms$, $\delta = 3ms$ e $\delta = 5ms$, respectivamente.

4.4.4.4 Desempenho em termos do intervalo entre falsas suspeitas (TMR)

Em termos do intervalo entre falsas suspeitas, Figura 4.19, os detectores autonômicos obtiveram o melhor desempenho quando comparados a todos os demais detectores adaptativos considerados. As rampas apresentas no gráfico para os detectores autonômicos indicam que os mesmo levam muito mais tempo para cometer uma falsa suspeita.

Além disso, os valores médios de TMR estão na ordem de 10^3 milissegundos, sendo os mesmos $548,78ms$ e $927,28ms$ para $AFD\text{-}RBL$ e $AFD\text{-}RBS$, respectivamente. O bom desempenho dos detectores autonômicos em termos de TMR também se deve à combinação da abordagem usada para o cálculo da margem de segurança combinada com maiores períodos de monitora-

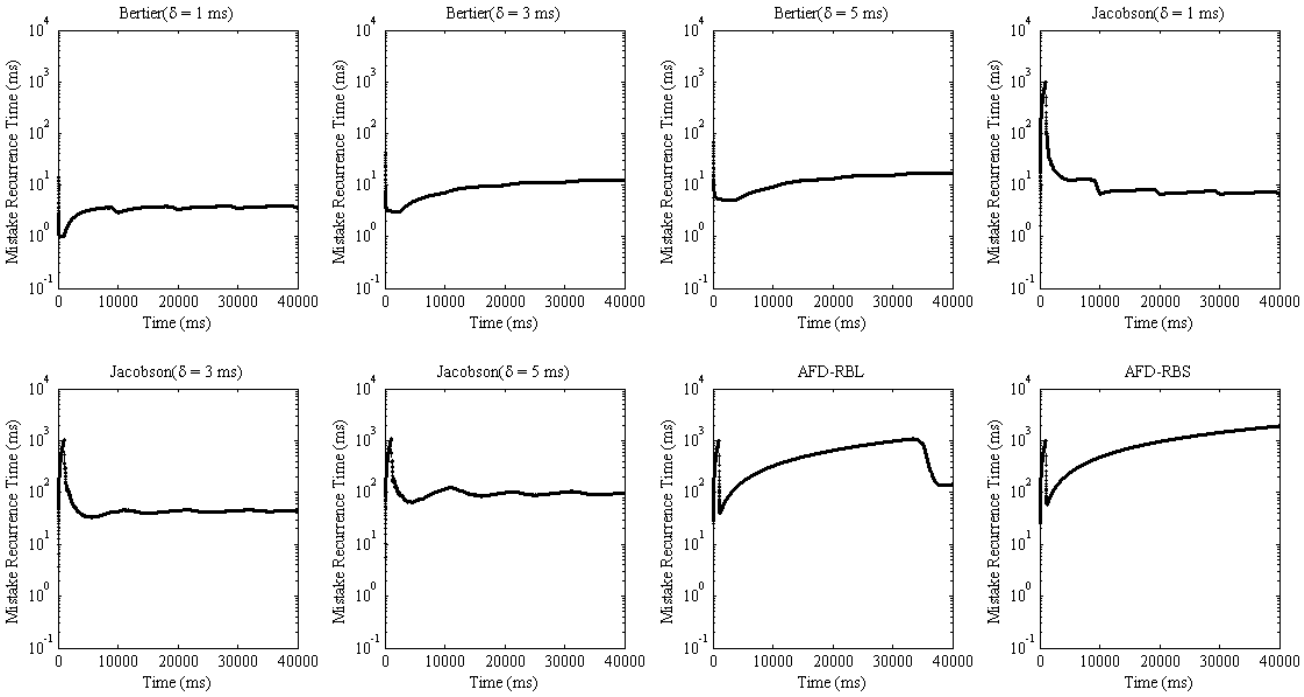


Figura 4.19. Desempenho em termos do intervalo entre falsas suspeitas

mento dinamicamente calculados para altas condições de carga na rede.

O detector *Jacobson* obteve o segundo melhor desempenho, em termos de TMR, sendo os valores médios de TMR iguais a $22,51ms$, $58,42ms$ e $109,70ms$ para δ igual a 1, 3 e $5ms$, respectivamente.

Os detectores *Bertier* teve o pior desempenho em termos de TMR quando comparado com os demais. Sendo que para tal detector, as médias de TMR foram $3,47ms$, $9,28ms$ e $12,55ms$, respectivamente.

No casos dos detectores adaptativos o baixo desempenho se deve aos períodos de monitoramento fixados. E no caso do detectores de *Bertier*, o desempenho muito ruim se também ao efeito de memória provocado pelo tamanho do histórico de mensagens utilizado pelo estimador de *timeout* usado por tal abordagem. Note que, o histórico longo faz com que o mesmo demore mais para responder a mudanças repentinas na carga, ocasionando um número maior de falsas suspeitas e, conseqüentemente menores valores médios de TMR.

4.4.4.5 Desempenho em termos da disponibilidade de detecção (AV)

Por fim, a Figura 4.20 apresenta o desempenho dos detectores em termos da disponibilidade de detecção.

A disponibilidade do detector Bertier($\delta = 1\,ms$) decai até atingir valores de $0,3398$ na média, sendo este o pior desempenho em termo desta métrica. O detector *Bertier* com $\delta = 3ms$ e $\delta = 5ms$ apresenta desempenho médio em termos de AV de $0,9849$ e $0,9881$, respectivamente.

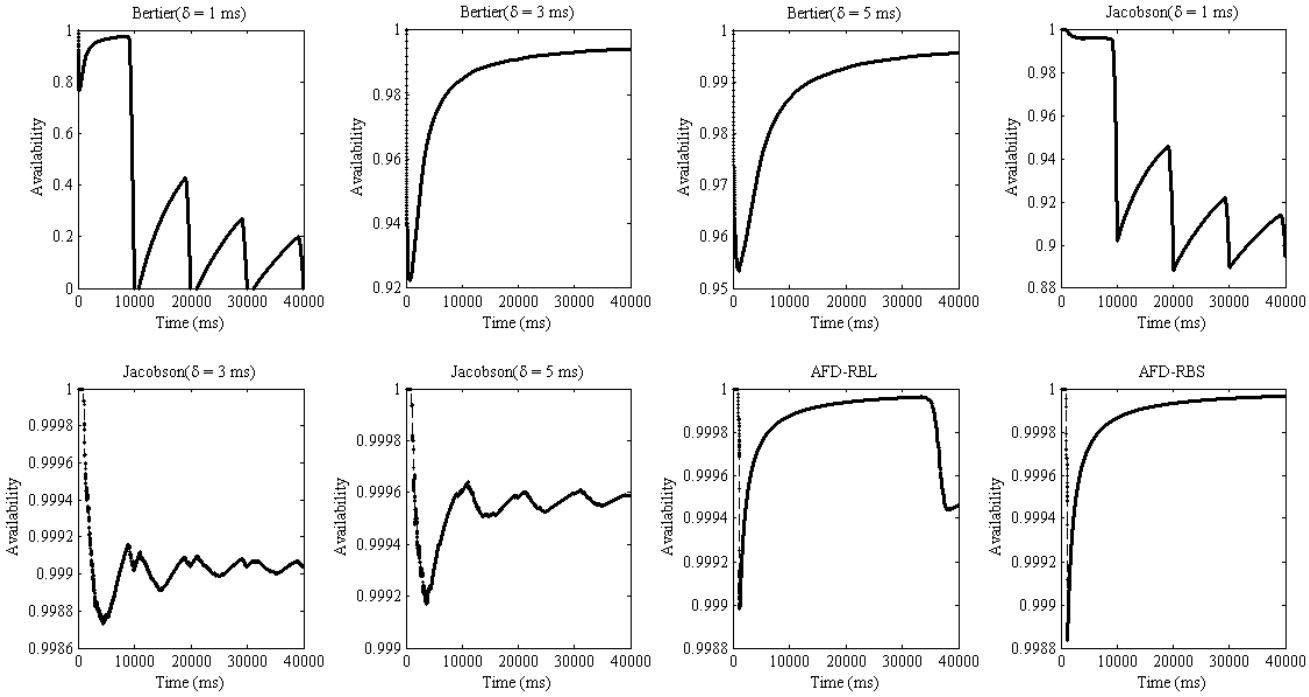


Figura 4.20. Desempenho em termos da disponibilidade de detecção

Ainda assim, o mesmo apresenta desempenho inferior aos detectores autonômicos e ao detector *Jacobson*. Observe que, no melhor caso o detector *Bertier* apresenta disponibilidade média de dois 9 (i.e., $0,99$ aproximadamente).

O detector *Jacobson* apresentaram disponibilidades médias de $0,9334$, $0,9991$ e $0,9995$ para δ igual a 1, 3 e 5ms, respectivamente. Deste modo, o detector *Jacobson* apresenta, no melhor caso, disponibilidade média de três 9 (i.e., $0,999$ aproximadamente).

Os detectores autonômicos *AFD-RBL* e *AFD-RBS* apresentaram disponibilidade média de $0,9998$ e $0,9999$, respectivamente. Sendo assim, os detectores autonômicos possuem desempenho em termos desta métrica muito melhor que os detectores adaptativos considerados. Observe que, no melhor caso, os detectores autonômicos possuem disponibilidade média de quatro 9 (i.e., $0,9999$ aproximadamente).

Nas figuras 4.17, 4.19 e 4.20, observa-se uma queda de QoS para *AFD-RBL*, no instante 35000ms das simulações. Isto porque, neste ponto, *AFD-RBL* está com δ decrescendo (verifique a Figura 4.16), chegando ao ponto de interferência máxima (menor δ), até o ponto em que o período estabiliza e começa a aumentar como indicam os gráficos no fim da curva. O mesmo não acontece com *AFD-RBS* por conta da ação integral empregada, por tal detector, no ajuste do período de monitoramento.

4.4.4.6 Sumário dos resultados.

A Tabela 4.1 apresenta o desempenho médio dos detectores em termo das métricas consideradas.

Tabela 4.1. Sumário do desempenho médio dos detectores de defeitos em termos das métricas

	Bertier et. al. (2002)			Jacobson (1988)			AFD-RBL	AFD-RBS
	$\delta = 1ms$	$\delta = 3ms$	$\delta = 5ms$	$\delta = 1ms$	$\delta = 3ms$	$\delta = 5ms$		
TD (ms)	2,69	3,35	5,34	3,04	3,39	5,90	1,72	2,54
TM (ms)	2,40	0,10	0,11	0,49	0,04	0,04	0,04	0,06
TMR (ms)	3,47	9,28	12,55	22,51	58,42	109,70	548,78	927,81
RM	0,46	0,38	0,30	0,11	0,06	0,05	0,00	0,00
AV	0,3398	0,9849	0,9881	0,9334	0,9991	0,9995	0,9998	0,9999

4.5 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 capítulo apresentou contribuições inéditas para o projeto e implementação de detectores autonômicos baseados em teoria de controle e com habilidade de configurar, de forma dinâmica, o período de monitoramento e o *timeout* de detecção em função de demandas de QoS definidas pelo usuário.

As avaliações realizadas, através de simulação, verificaram o desempenho dos detectores em função da taxa de falsas suspeitas, tempo de detecção, disponibilidade, intervalo entre falsas suspeitas de detecção e duração das falsas suspeitas. Tais métricas 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 comparações com detectores adaptativos disponíveis na literatura e configurados (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 dos detectores autonômicos nunca é inferior a dos detectores avaliados.

Os detectores autonômicos implementados consideraram o encapsulamento do detector adaptativo de Jacobson. Entretanto, os detectores autonômicos foram projetados para encapsular qualquer detector adaptativo e não apenas o usado na implementação. Isto torna a solução geral o bastante para aproveitar qualquer outro detector adaptativo que tenha desempenho melhor que o considerado na implementação. A Seção A.3 do Apêndice A ilustra o desempenho do detector autonômico usando o estimador de *timeout* de Bertier, Marin e Sens (2002).

As avaliações foram realizadas para redes locais, típicas de aplicações como *cluster computing*. Atualmente, as abordagens de detecção autonômicas estão sendo implementadas em um ambiente federado de computação em nuvem – a qual considera a interação entre componentes distribuídos em nuvens computacionais de diferentes provedores de TI. Além disso, as mesmas serão integradas a um gestor de aplicações autonômicas em desenvolvimento no LaSiD (Andrade; Macêdo, 2009).

Do ponto de vista prático, a contribuição apresentada nesta proposta ressalta que a combinação de ajuste dinâmico de período de monitoramento e *timeout* de detecção trazem grandes benefícios ao desempenho dos detectores de defeitos para sistemas distribuídos. Observe que, mesmo considerando métricas de qualidade de serviço na configuração do detector, o atendimento das demandas de QoS nem sempre podem ser garantidas, uma vez que o atendimento dessas demandas depende das garantias fornecidas pelo ambiente computacional no qual o detector está inserido. Entretanto, o uso dessas demandas de QoS na configuração do detector permite que o serviço autonômico de detecção tente fornecer a QoS desejada nos momentos em que o ambiente fornece as garantias necessárias, sem a intervenção do usuário.

Um outro aspecto importante do ponto de vista prático, é apresentar, aos administradores, engenheiros e desenvolvedores de sistemas, como os parâmetros de configuração (i.e. TD^U, TM^U, TMR^L e RC^D) do detector autonômico impactam no desempenho do mesmo em termos de qualidade de serviço de detecção. Assim, uma avaliação experimental foi realizada e o impacto de cada um dos parâmetros de configuração foram verificados e discutidos na Seção A.2 do Apêndice A. Ainda assim, fica pendente como trabalho futuro apresentar como o mapeamento horizontal e vertical de métricas de qualidade de serviço de detecção podem ser realizadas no caso da detecção autonômica, isto é: [a] como acomodar diferentes demandas de qualidade de serviço de detecção, oriundas de diferentes aplicações, na configuração do detector autonômico (mapeamento horizontal de QoS); e [b] como as métricas de qualidade de serviços diretamente ligadas ao desempenho das aplicações (e.g. tempo de resposta, disponibilidade, vazão de transações por segundo etc.) podem ser traduzidas nos parâmetros de configuração do detector autonômico (mapeamento vertical de QoS).

Capítulo 5

Este Capítulo apresenta a proposta de protocolo autonômica de comunicação em grupo para sistemas distribuídos, contextualizando a mesma com relação ao estado da arte; e discutindo os aspectos de implementação, os experimentos realizados, os resultados obtidos e as perspectivas de trabalhos futuros.

PROTOCOLOS AUTONÔMICOS DE COMUNICAÇÃO EM GRUPO

5.1 INTRODUÇÃO

Por muitos anos, a comunicação em grupo vem sendo usada como uma abstração poderosa para a construção de aplicações distribuídas confiáveis – uma extensa discussão desses protocolos pode ser encontrada em, por exemplo, Birman (1993), Cristian (1996), Chandra et al. (1996), Chockler, Keidar e Vitenberg (2001) e Défago, Schiper e Urbán (2004). Atualmente, as diferentes propostas de protocolos de comunicação em grupo possuem uma grande variabilidade propriedades, dependendo dos ambientes e das aplicações para os quais foram projetados. De um modo geral, a idéia básica por traz da comunicação em grupo é ocultar, das aplicações, a complexidade e a incerteza dos sistemas de comunicação subjacentes, tais como falhas, problemas com ordenação de mensagens etc. Além disso, a comunicação em grupo provê, aos processos da aplicação, garantias associadas às seqüências de configurações ou visões (i.e. *membership view*), relacionadas aos processos ativos e que podem ser usadas, por exemplo, para gerenciar de forma consistente um grupo de servidores replicados.

Para prover, de forma continuada, as aplicações com entregas de mensagens e informações atualizadas das mudanças nas composições de grupo (impostas por falhas, ou entrada e saídas espontâneas de processos), os protocolos de comunicações em grupo devem trocar mensagens e continuamente monitorar todos os membros do grupo – o que pode, em certas condições de carga, implicar em uma sobrecarga inaceitável de mensagens no sistema de comunicação subjacente. Conseqüentemente, o projeto de tais protocolos devem determinar parâmetros operacionais de modo a lidar com a relação de compromisso entre os requisitos de desempenho, tais como velocidade (e.g. latência de entrega) e custo (e.g. sobrecarga de mensagens). Entretanto, quando o comportamento do ambiente computacional é desconhecido e pode mudar com o

tempo, ou quando os requisitos das aplicações mudam dinamicamente, a auto-configuração dos parâmetros operacionais de tais protocolos é um aspecto básico que é normalmente ignorado pelas abordagens de comunicação em grupo existentes.

Nesse contexto, este capítulo apresenta o projeto, implementação e avaliação de um protocolo autonômico de comunicação em grupo capaz de auto-configurar, em tempo de execução, seus parâmetros operacionais, a partir de requisitos definidos pelo usuário (e.g. consumo de recursos), ver Macêdo, Freitas e Sá (2011). Para tanto, a proposta estende o protocolo de comunicação em grupo baseada em *Blocos Causais*[1] de Macêdo (1994) e introduz um mecanismo autonômico (baseado em teoria de controle) para reconfigurar dinamicamente tal protocolo para certos objetivos, definidos de forma estática ou dinâmica.

Macêdo (2007, 2008a) e Macêdo e Freitas (2009, 2010) também propõem extensões do protocolo de comunicação em grupo baseado em blocos causais, ambas as propostas focam em aspectos de adaptação voltadas para o suporte da comunicação em grupo considerando ambientes distribuídos híbridos e dinâmicos – i.e. ambientes nos quais as características de sincronia dos processos e canais são heterogêneas e podem mudar com o tempo. O protocolo autonômico de comunicação em grupo proposto nesta Tese, por outro lado, foca apenas em modelos de sistemas distribuídos parcialmente síncronos e utiliza uma abordagem autonômica (baseada em teoria de controle) para atender aos objetivos definidos pelo usuário, o que é um aspecto não considerado em Macêdo e Freitas (2009, 2010). Contudo, o gestor autonômico proposto pode ser facilmente adaptado para trabalhar com as abordagens de Macêdo e Freitas (2009, 2010) – tal adaptação não foi considerada nesta Tese para garantir uma maior simplicidade na exposição da abordagem autonômica.

Na literatura, algumas propostas têm focado em estratégias adaptativas para o atendimento da relação de compromisso entre requisitos de custo e de latência dos protocolos de comunicação em grupo, mas em diferentes perspectivas da apresentada nesta Tese.

Macêdo, Ezhilchelvan e Shrivastava (1993), Ezhilchelvan, Macêdo e Shrivastava (1995), por exemplo, propõem uma plataforma de comunicação em grupo com garantias de ordenação total (dita Newtop). Esta plataforma se baseia em blocos causais e suporta o chaveamento entre esquemas de ordenação baseado em componente centralizado (i.e. seqüenciador) e baseado em consenso distribuído. Entretanto, este chaveamento (ou adaptação) é disparado pela aplicação ou pelo usuário final.

Litiu e Prakash (1998) e Chockler, Huleihel e Dolev (1998) propõem abordagens adaptativas de comunicação em grupo, baseadas em prioridades dinamicamente definidas, para permitir que aplicações obtenham garantias de ordenação que determinem latências de entrega adequadas quando diferentes grupos de processos são considerados.

Em Litiu e Prakash (1998), a abordagem baseada em prioridades é usada para que aplicações de *groupware* com perfil interativo de tráfego (e.g. *chat*) sejam privilegiadas em relação às aplicações com perfil de tráfego por rajadas (e.g. multimídia) – determinando, assim, perfis

[1] Tradução do inglês, *Causal Blocks*.

diferenciados em termos de latência de entrega de mensagens. Entretanto, a abordagem de Litiu e Prakash (1998) não considera que os requisitos de usuários possam mudar com o tempo e, diferente da proposta autonômica apresentada nesta Tese, implementa a adaptação usando prioridades e utilizando um protocolo de comunicação em grupo baseado em componente centralizado (dito, *Corona Server*), responsável tanto pela ordenação das mensagens quanto pelo *membership*.

Chockler, Huleihel e Dolev (1998), por sua vez, propõem um protocolo adaptativo de comunicação em grupo com garantias de ordenação total, baseada em consenso. Neste protocolo (dito ATOP), as mensagens são assinaladas no envio com prioridades dinâmicas baseadas na taxa de envio da aplicação. Então, essas prioridades são usadas durante o consenso, pelos membros do grupo, para determinar a ordem da entrega. A abordagem apresentada nesta Tese usa *timeouts* adaptativos que também refletem a taxa de envio da aplicação. Entretanto, diferente de Chockler, Huleihel e Dolev (1998), a adaptação é realizada pelo gestor autonômico de modo a atender a relação de compromisso entre latência e custo, considerando mudanças dinâmicas no ambiente e expectativas de desempenho definidas dinamicamente pelas aplicações (ou usuários). Além disso, a abordagem de adaptação de Chockler, Huleihel e Dolev (1998) é implementada considerando ambientes de redes locais, a abordagem autonômica apresentada nesta Tese, entretanto, considera a implementação do protocolo em ambientes abertos típicos, como a Internet – contudo, sem prejuízo em seu uso em redes locais.

Outras propostas existentes na literatura lidam com a relação de compromisso entre custo e latência de entrega através de abordagens adaptativas que realizam o chaveamento dinâmico entre diferentes implementações de protocolos de comunicação em grupo na medida em que as condições de carga no ambiente computacional variam – ver, por exemplo, Renesse et al. (1998), Liu et al. (2001), Rütti, Wojciechowski e Schiper (2006), Mocito e Rodrigues (2006) e Karmakar e Gupta (2007). Essas abordagens, diferente da proposta autonômica apresentada nesta Tese, não consideram demandas dinâmicas definidas pelos usuários no processo de chaveamento. Além disso, a proposta autonômica não considera qualquer tipo de chaveamento, principalmente, por que a maioria dessas abordagens é baseada em algum tipo de acordo durante o chaveamento, implicando, em certas condições de carga no ambiente, em custos em termos de sobrecarga de mensagens e processamento, que podem degradar ainda mais o desempenho das aplicações e dos próprios mecanismos de comunicação em grupo.

Neste contexto, considerando o estado da arte, a proposta apresentada nesta Tese é a primeira a enfrentar o problema da relação de compromisso entre custo e latência de entrega, considerando uma abordagem autonômica (e baseada em teoria de controle). Observe que a idéia principal da proposta, discutida mais adiante, é demonstrar como a gestão autonômica pode ajudar no atendimento de tal relação de compromisso. Nesse sentido, diferentes protocolos de comunicação em grupo poderiam ser usados. No entanto, o desenvolvimento de uma versão autonômica do protocolo de comunicação em grupo baseado em blocos causais foi considerada porque tal protocolo disponibiliza, em um framework integrado, mecanismos de detecção de

defeitos, ordenação de mensagens e reconfiguração do grupo – o que facilita a construção da abordagem autonômica. Por fim, para demonstrar o desempenho da abordagem autonômica de comunicação em grupo proposta, a mesma foi avaliada considerando diferentes condições de carga, incluindo diferentes cenários com e sem falhas de processos. Além disso, considerou-se o desempenho do protocolo quando os requisitos do usuário variam durante a execução. Os resultados obtidos foram comparados com o desempenho do protocolo de Macêdo (1994), manualmente configurado para os diferentes cenários considerados.

Assim, o restante deste Capítulo: a) discute o modelo de sistema adotado na concepção da proposta (Seção 5.2); b) apresenta o serviço básico de comunicação em grupo usado(Seção 5.3); c) descreve os detalhes de implementação do mecanismo autonômico proposto (Seção 5.4); d) discute os experimentos realizados para verificação do desempenho da proposta autonômica (Seção 5.5); e, por fim, apresenta algumas considerações finais e propostas de trabalhos futuros (Seção 5.6).

5.2 MODELO DE SISTEMA

O modelo de sistema considerado consiste em um conjunto finito Π de $n > 1$ processos, definido por $\Pi = \{p_1, p_2, ..., p_n\}$. Esses processos do sistema se comunicam e sincronizam entre si a partir do envio e recebimento de mensagens usando um sistema de comunicação subjacente. Ao longo do tempo, cada processo executa uma série de passos. Cada passo corresponde a uma ação, ocasionada por eventos internos ou externos, que provoca uma mudança de estado, como, por exemplo: (i) o envio ou o recebimento de mensagens; ou, ainda, o processamento ou armazenamento local. Além disso, cada processo possui acesso ao hardware de seu relógio local, o qual é caracterizado por uma taxa de desvio (*drift*), em relação ao tempo real, limitada por ρ. Os processos do sistema podem falhar por parada (i.e. crash)[2]. Processos que não falham por *crash* são ditos corretos.

Cada par de processos $(p_i; p_j)$ é conectado usando canais FIFO bidirecionais, os quais não criam, alteram, ou perdem mensagens. Assim, se p_i envia uma mensagem para p_j, então se p_i é correto, em algum momento p_j receberá a mensagem, a menos que o mesmo falhe.

O modelo de sistema distribuído considerado é livre de tempo (*time-free*). Desse modo, não existem limites para a transmissão ou processamento das mensagens – o que torna o protocolo proposto mais portável e menos sensível a condições operacionais. Por conta disto, atrasos longos e imprevisíveis na transmissão das mensagens, por exemplo, não afetam a correção (i.e. *safety properties*) do protocolo e de seus subprotocolos associados. Contudo, os limites superior (d_{max}) e inferior (d_{min}) para a transmissão das mensagens nos mecanismos autonômicos, desenvolvidos nesta Tese, são continuamente estimados a partir dos atrasos observados, sendo os mesmos usados para suspeitar de falhas de processos do sistema.

O protocolo de comunicação proposto requer que uma seqüência única de visões seja insta-

[2]Falhas bizantinas não são consideradas.

lada. Com isso, um protocolo de consenso é necessário para que os membros do grupo possam entrar em acordo sobre a seqüência das visões de grupo que são instaladas a cada momento.

Por outro lado, é sabido que o problema do consenso distribuído não pode ser solucionado em modelos de sistemas livres de tempo (Fischer; Lynch; Paterson, 1985). Assim, um modelo parcialmente síncrono é assumido de modo a permitir que o consenso seja obtido nos momentos em que o sistema apresente um comportamento estável (Chandra; Toueg, 1996) – isto é, apresente períodos de sincronia suficientemente longos para que rodadas do consenso possam ser finalizadas.

Apesar de tais considerações, as propriedades relacionadas à ausência de limites temporais para as primitivas do grupo são garantidas. Mais ainda, assume-se a existência de tal protocolo de consenso e de um respectivo detector de defeitos do tipo $\diamond S$, dito FD – em que $FD(p) = true$ denota que o detector suspeita da falha de um processo p. Essas condições são necessárias para resolver a maioria dos problemas fundamentais de tolerância a falhas em sistemas distribuídos (Dolev; Dwork; Stockmeyer, 1987; Hadzilacos; Toueg, 1993).

5.2.1 Propriedades do grupo

Os processos do sistema formam um grupo g, cuja configuração inicial é $g = \Pi$ – por questões de simplicidade múltiplos grupos não são considerados.

Um processo p_i de um grupo g instala visões, denotadas por $v_i(g) \subseteq \Pi$. Uma visão representa um conjunto de membros do grupo que mutuamente se consideram operacionais. Esse conjunto muda dinamicamente na ocorrência de falhas (suspeitas) de processos ou quando os processos deixam ou associam-se a g – através de primitivas de associação (*join*) e dissociação (*leave*), as quais não são discutidas neste Trabalho.

A todo momento, pode ocorrer uma mudança de visão e cada nova visão instalada é associada a um número que incrementa monotonicamente. Para tanto, $v_i^k(g)$ denota o número da visão k instalada por um processo p_i – entretanto, quando adequado, a identidade do processo ou do grupo será omitida na identidade da visão, podendo ser usado, por exemplo, $v^k(g)$, v_i^k ou v^k. Um processo p_i difunde mensagens apenas para os processos de sua visão atual.

Em geral, um protocolo de comunicação em grupo deve satisfazer a propriedades de *safety* e *liveness*, relacionadas ao conjunto de mensagens entregues e às visões instaladas pelos processos. Tais propriedades variam de uma implementação para outra, de acordo com o ambiente computacional considerado (Chockler; Keidar; Vitenberg, 2001; Cristian, 1996). O protocolo de comunicação em grupo proposto nesta Tese visa, entre outras aplicações, a implementação de mecanismos de replicação ativa de servidores. Conseqüentemente, as propriedades especificadas para o protocolo proposto devem satisfazer não apenas a propriedade de ordenação total (e causalidade) na entrega das mensagens, mas também a propriedade de acordo em um histórico linear das visões do grupo (Schneider, 1990; Lamport, 1978) – essas propriedades são brevemente descritas a seguir.

Para assegurar ***liveness* na entrega das mensagens**, a propriedade de *validade* determina que um processo correto entregará, no instante $t + D_1$, uma mensagem enviada pelo mesmo no instante t. Para garantir ***safety* na entrega das mensagens**, as seguintes propriedades devem ser satisfeitas: a) *acordo uniforme* – se um processo entrega uma mensagem m em uma visão, então todos os processos corretos devem entregar m na mesma visão; e b) *ordenação total uniforme* e *ordenação causal* – os processos seguem a mesma ordem de entrega das mensagens, respeitando a causalidade potencialmente existente (Lamport, 1978).

Para assegurar ***liveness* na entrega das visões**, a propriedade de detecção de defeitos garante que se um processo falha em um instante t, então todos os processos corretos detectarão tal falha no instante $t + D_2$ e instalarão uma nova visão que exclui o processo defeituoso. Para assegurar ***safety* na entrega das visões**, os processos corretos devem concordar em uma visão, seguindo a propriedade de seqüência de visões única (***unique sequence of views** property*). Além disso, as exclusões de processos de um grupo devem ser justificadas por falhas ou suspeitas, isto é: se um processo não pertence a uma nova visão, então o mesmo falhou ou foi suspeito de ter falhado (exclusão justificada – *exclusion justification*). Mais ainda, um processo p instala uma nova visão $v(g)$ se, e somente se, o p pertence à $v(g)$, i.e. a instalação de uma nova visão deve atender a propriedade de auto-inclusão (*self-inclusion*).

Finalmente, os limites D_1 e D_2, considerados acima, são desconhecidos e assumem valores finitos, mas arbitrários.

5.3 O PROTOCOLO BÁSICO DE COMUNICAÇÃO EM GRUPO

O protocolo básico de comunicação em grupo utilizado é derivado das implementações do modelo de blocos causais propostas em Macêdo (1994, 2007, 2008a) e extendidas em Macêdo e Freitas (2009, 2010). Este protocolo é dito básico, pois será especializado mais adiante com a introdução das facilidades de gestão autonômica. Uma descrição mais completa das questões relacionadas ao mecanismo de blocos causais e a facilidade de acordo uniforme podem ser encontrada em Macêdo (1994, 2008a). A descrição do protocolo básico de comunicação em grupo proposto, a partir de tal combinação, é brevemente apresentada a seguir.

5.3.1 A abordagem de blocos causais

Cada processo p_i mantém um relógio lógico (Lamport, 1978), dito *Block Counter* (denotado por BC_i), e as mensagens enviadas são assinaladas (*timestamped*) com o valor atual de tal relógio lógico (i.e. *Block counter*). Um processo p_i usa um bloco causal (*Causal Block*) para representar mensagens concorrentes, as quais foram enviadas ou recebidas com um mesmo número de bloco. O conjunto de blocos causais ordenados pelos seus respectivos números de bloco permite construir uma matriz de blocos *BM*, como é mostrado na Figura 5.1 para um grupo de processos com 6 membros. A matriz *BM* representa todas as mensagens enviadas/recebidas por um processo, o qual possui sua própria matriz. Na matriz da Figura 5.1, por exemplo, os números

de bloco das últimas mensagens recebidas pelos processos p_1 e p_2 são 4 e 5, respectivamente.

	p_1	p_2	p_3	p_4	p_5	p_6
1	+			+		
2		+			+	+
3	+		+	+		
4	+				+	
5		+				

Figura 5.1. Matriz de blocos de um grupo de processos com seis membros

Graças ao uso de canais FIFO confiáveis, uma vez que a matriz de blocos em um processo p_i indica a recepção de uma mensagem m com número de bloco $m.b$ originada de um processo p_j, nenhuma outra mensagem de p_j com um número de bloco igual ou inferior ($b' \leq m.b$) jamais será recebida por p_i.

Nesse sentido, a noção de *completude de bloco* (*block completion*) pode ser construída para determinar se um dado bloco contém todas as mensagens relacionadas (i.e. nenhuma mensagem com o mesmo número de bloco é esperada) – definindo, então, um bloco completo. No exemplo da Figura 5.1, apenas os blocos 1 e 2 (i.e. linhas, $BM[1]$ e $BM[2]$) são completos.

A *completude de bloco* pode ser usada para prover entrega em ordem causal e total. Para prover ordenação total, depois do preenchimento do bloco $BM[B]$ é possível entregar as mensagens do mesmo em uma ordem pré-definida (e.g. de acordo com os identificadores únicos dos transmissores). Em ambos os casos, a entrega deve ocorrer em uma ordem crescente em termos dos números dos blocos.

Para garantir *liveness* na *completude do bloco*, e conseqüentemente, na entrega das mensagens, cada processo é provido com um mecanismo simples, chamado de *time-silence*, o qual permite a um processo permanecer em atividade nos momentos nos quais o mesmo não está gerando mensagens da aplicação. Para tanto, o mecanismo de *time-silence* atua conforme a seguir. Quando um bloco $BM[B]$ é criado por um processo p_i, esse processo inicia um *timeout* ts (no instante t_i). Quando ts expira (no instante $t_i + ts$), se p_i não enviou ainda uma mensagem m para contribuir para a *completude do bloco* ($m.b \geq B$), uma mensagem *null*, marcada com o maior número de bloco conhecido, é enviada por p_i.

A Figura 5.2 mostra a evolução das matrizes de blocos de cada processo, a cada envio/recebimento de mensagem e na atuação do mecanismo de *time-silence* para um grupo de processos com 3 membros. Verifique que o processo p_2, ao receber a primeira mensagem (vinda do processo p_1), inicia o mecanismo de *time-silence* para o bloco causal $BM[1]$. Na medida em que novos envios e recebimentos de mensagens acontecem, a matriz de blocos em cada processo é atualizada. Depois de expirado o *time-silence* para $BM[1]$, em p_2, e ,uma vez que, o mesmo não possui qualquer mensagem de aplicação a ser enviada, o mecanismo de *time-silence* provoca a difusão de uma mensagem *null* para os demais processos do grupo. O envio dessa mensagem permite que o bloco $BM[1]$ alcance a completude no processo p_2, enquanto que, os blocos

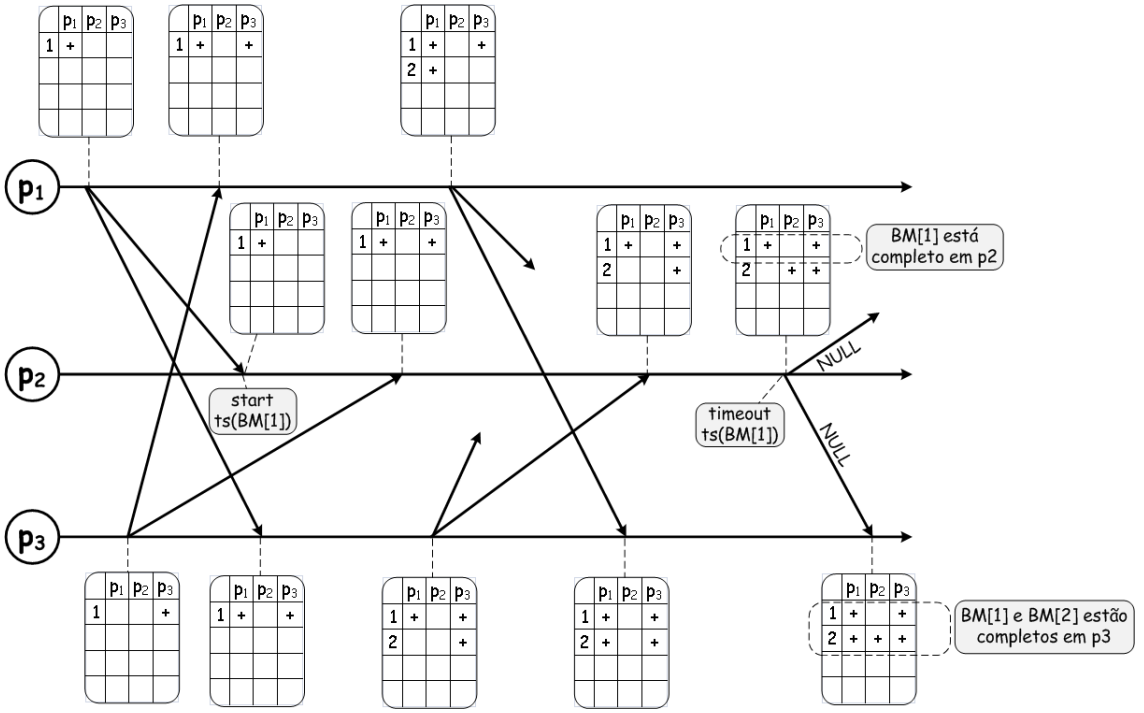


Figura 5.2. Evolução dos blocos causais em um grupo de processos com três membros

$BM[1]$ e $BM[3]$ alcançam a completude no processo p_3 (ver Figura 5.2).

Embora o sistema seja livre de tempo (*time-free*), usa-se os limites estimados (d_{min} e d_{max}) para a transmissão das mensagens para estipular um limite superior para a *completude de bloco*, conforme a seguir. Assuma que $BM[m.b]$ foi criado no instante t_i. Os limites de tempo para a completude de $BM[m.b]$ em um processo p_i, medido a partir do relógio local do mesmo, é:

- $TC1$: $(t_i + ts(m.b) + 2d_{max})(1+\rho)$, se m foi enviada por p_i;
- $TC2$: $(t_i + ts(m.b) + 2d_{max} - d_{min})(1+\rho)$, se m foi recebida por p_i;

Uma discussão sobre a estimativa dos limites d_{min} e d_{max} são apresentados, mais adiante, na Seção 5.4.1.

5.3.2 Visão geral do protocolo básico de comunicação em grupo

Uma mensagem m enviada para um grupo de processos, chega a todos os membros, se o processo transmissor não falha durante a transmissão de m – no caso de falha, alguns membros do grupo podem não receber m. Por isso, quando uma mensagem é recebida por um processo, o mesmo não pode descartá-la imediatamente e a retransmissão da mesma pode ser necessária para satisfazer a propriedade de acordo. Por conta disso, a mensagem recebida deve ser armazenada até que o processo receptor (ou transmissor) saiba que todos os demais processos a tenham recebido.

As mensagens não confirmadas por todos os processos membros do grupo são chamadas

instáveis (as mesmas são ditas estáveis, caso contrário)[3]. No protocolo de comunicação em grupo apresentado em Macêdo (2008a) e Macêdo e Freitas (2009), na medida em que as mensagens se tornam estáveis, as mesmas são descartadas da memória local – o que é o bastante para assegurar o *acordo* (*agreement*) na entrega das mensagens. No protocolo básico apresentado nesta Tese, por outro lado, para assegurar o *acordo uniforme* (*uniform agreement*) na entrega da mensagem, mesmo os processos falhos devem concordar com a ordem de entrega. Por isso, é necessário que uma mensagem seja super-estável antes de ser descartada, isto é: um bloco ou mensagem é dito super-estável quando é sabido que a mesma é estável para todos os processos (inclusive para os processos falhos) – seguindo, deste modo, a mesma estratégia proposta em Macêdo e Freitas (2010).

5.3.2.1 Acordo uniforme na entrega das mensagens

Para determinar quando um bloco é estável, os membros do grupo informam, em toda mensagem transmitida, o último bloco completo (*LCB*). Através da coleta de todas as informações de *LCB* disponíveis, é possível derivar o último bloco estável usando $LSB = min\{LCB_i\}$, $\forall p_i \in g$. Assim, todos os blocos de um processo p_i com número igual ou menor que *LSB* são estáveis para o mesmo (Macêdo, 1994).

Para assegurar que os membros do grupo entregam o mesmo conjunto de mensagens, na mesma ordem e atendendo ao *acordo uniforme* na entrega, as seguintes condições devem ser satisfeitas, em que $m.b$ é o número do bloco da mensagem m:

- *stable-safe1*: uma recebida mensagem (i.e. m) é *deliverable* (i.e. pode ser entregue) se $BM[m.b]$ é estável;

- *stable-safe2*: todas as mensagens que se tornam *deliverable* são entregues em ordem não decrescente de seus números de blocos – uma ordem de entrega pré-determinada é imposta para todas as mensagens com o mesmo número de bloco.

Para garantir *liveness* na entrega das mensagens, o mecanismo de *time-silence* deve atuar mesmo na ausência de blocos incompletos, isto é: depois da completude do bloco, quando existe um período inativo, de ts unidades de tempo, sem blocos incompletos, uma mensagem *null* é enviada para disseminar a informação de *LCB* para todos os processos. Entretanto, mensagens de *time-silence* não criam novos blocos.

5.3.2.2 Timeouts para a estabilidade dos blocos

Assuma que $BM[m.b]$ é criado no instante t_i por p_i (medido usando seu relógio local). Os *timeouts* para a estabilidade do bloco podem ser estimados da seguinte forma:

[3]Maiores detalhes sobre a detecção de mensagens estáveis, no contexto do conceito dos blocos causais, podem ser encontrados em Macêdo (1994).

- $ST1$: $(t_i + 2ts(m.b) + 3d_{max})(1+\rho)$, se m foi enviada por p_i;

- $ST2$: $(t_i + 2ts(m.b) + 3d_{max} - d_{min})(1+\rho)$, se m foi recebida por p_i;

A noção de blocos estáveis também pode ser explorada para implementar as operações de junção (*join*) e dissociação (*leave*) – isto é: quando um processo deseja se juntar (*join*) ou deixar (*leave*) o grupo, essa informação é propagada para os membros do grupo e, uma vez que, tal informação (*join* ou *leave*) se torna estável uma nova visão pode ser instalada.

5.3.2.3 Algoritmos

Os principais algoritmos, contendo os procedimentos implementados pelo protocolo básico de comunicação em grupo, são descritos a seguir.

Procedimentos para a gestão dos blocos causais e dos timeouts. O Algoritmo 5.1 é disparado no envio/recebimento de uma mensagem m, então o mesmo atribui os respectivos *timeouts* (no caso de criação de blocos, linhas 3–7), armazena a mensagem m em seu *buffer* local (linha 10) e dispara (linha 11) a tarefa de entrega (linha 11). Então, a tarefa de entrega (Algoritmo 5.2) realizará a entrega das mensagens estáveis, seguindo as condições de entrega definidas (i.e. *stable-safe1* e *stable-safe2*, ver Seção 5.3.2.1).

Algoritmo 5.1: *Tarefa de manutenção de blocos causais*

```
1  on event ((m sending ) or (m receiving )) at p_i do
2      if BM[m.b] does not exist then
3          create BM[m.b];
4          if p_i = m.sender then
5              set completion timeout TC1 and stability timeout ST1 for BM[m.b];
6          else
7              set completion timeout TC2 and stability timeout ST2 for BM[m.b];
8          end if
9      end if
10     store m at a local buffer;
11     signal delivery task (Algorithm 5.2);
12 end event
```

Algoritmo 5.2: *Tarefa de entrega de mensagens*

```
1  if any causal block gets stable then
2      deliver stable messages according to stable-safe1 and stable-safe2;
3  end if
4  update LCB and LSB;
5  cancel related timeouts for complete or stable causal blocks;
```

Procedimentos para a manutenção das visões de grupo. Na falha de um p_k, um *timeout* expirará em um processo p_i para um bloco $BM[m.b]$. Para proceder com a entrega da mensagem, um novo *membership* para g deve ser estabelecido, de modo a excluir p_k (ou qualquer outro processo que tenha falhado). Assim, para assegurar que todos os membros do grupo executam o mesmo procedimento de instalação de visão, uma primitiva de difusão confiável (*reliable multicast*), dita $rmcast(ChangeViewRequest, B)$, com $B = m.b$, é usada para iniciar o procedimento de mudança de visão (Algoritmo 5.3).

Algoritmo 5.3: *Executado por p_i na expiração de um timeout para um bloco B*

```
1 on event (timeout expiration for B) at p_i do: rmcast(ChangeViewRequest,B);
```

Algoritmo 5.4: *Tarefa de mudança de visão*

```
1  on event ((ChangeViewRequest, B) receiving) at p_i do
2      if (unstable,B,LSB) was already been sent by p_i then exit;
3      block ordinary delivery at delivery task ;
4      rmcast(unstable, B,LSB);
5      wait until
6          (∀p_j ∈ v_i^k: received (unstable,B,LSB) from the majority of processes in view v_i^k) and (∀p_j ∈ v_i^k:
           either received (unstable,B,LSB) from p_j or FD(p_j) = true);
7      end wait
8      let allunstable_i be the union of the unstable sets received from all p_j and LSB_max the maximum value
       of LSB collected ;
9      let v_i^{k+1} be set of all p_j from which (unstable,B) was received ;
10     consensus(B,(v_i^{k+1}, allunstable_i, LSB_max)) ;
11     store messages from allunstable not yet received by p_i, sets LSB = LSB_max and apply stable-safe1 and
       stable-safe2 to blocks that get stable ;
12     if p_i ∉ v_i^{k+1} then
13         terminate p_i; (* p_i was removed due to a false suspicion from a p_j, i ≠ j *)
14     elseif v_i^k ≠ v_i^{k+1} then
15         install the decided view v_i^{k+1} at p_i;
16     end if
17     signal delivery task (Algorithm 5.2) for resuming ordinary message delivery ;
18 end event
```

A difusão confiável realizada, usando a primitiva *rmcast*, será recebida por todos os processos em funcionamento, os quais devem executar a tarefa de mudança de visão (ver Algoritmo 5.4). Com isso, a tarefa de mudança de visão usa a primitiva de difusão confiável para disseminar o valor de *LSB* e o conjunto de mensagens instáveis entre todos o processos em funcionamento – i.e. $rmcast(unstable, B, LSB)$, ver linha 4 do Algoritmo 5.4. Cada processo forma sua própria visão do conjunto de mensagens instáveis, dos processos em operação e do último bloco estável (LSB_{max}) – essa informação será usada junto com o detector $\Diamond S$, baseado no protocolo de consenso publicado Chandra e Toueg (1996), para concordar com uma visão idêntica

a ser instalada em todos os membros do grupo (graças ao *acordo uniforme* de tal consenso), ver linhas 5–10 do Algoritmo 5.4.

Ao final do consenso, o conjunto de mensagens instáveis são armazenadas em um buffer local, o *LSB* é atualizado e as mensagens são entregues seguindo as condições de entrega de mensagens, ver linha 11 do Algoritmo 5.4. Se a visão decidida não incluir um dado processo p_i (que falhou ao enviar seu conjunto de mensagens instáveis), o mesmo é finalizado (ver Algoritmo 5.4, linhas 12–13). Caso contrário, uma nova visão é instalada apenas se algum processo foi removido da visão atual (Algoritmo 5.4, linhas 14–15).

Ao final do procedimento de mudança de visão, as mensagens que faltavam para completar $BM[B]$ foram recuperadas e o procedimento de entrega é disparado (mesmo que nenhuma nova mudança de visão tenha sido instalada).

5.3.2.4 Correção do protocolo

Para ser correto, o protocolo deve satisfazer as propriedades descritas previamente. A seguir, é a apresentada a prova da propriedade de acordo uniforme (*uniform agreement*) para a entrega de mensagens. As provas para as demais propriedades são omitidas por questões de simplicidade. Entretanto, as mesmas podem ser facilmente derivadas das hipóteses do sistema, das propriedades do framework de blocos causais de Macêdo (1994) e do consenso Chandra e Toueg (1996).

Teorema 5.1 ***Acordo uniforme*** *Se um processo p_i entrega uma mensagem m na visão v_i^r, então todo processo correto p_j entrega m em v_j^r .*

Prova Assuma que uma mensagem m é enviada por um processo p_i na visão v_i^r. A mensagem m será entregue, em um processo p_j, assim que $BM[m.b]$ se torne estável em p_j. Em um cenário livre de falhas, se mensagens da aplicação não são geradas para completar $BM[m.b]$, o mecanismo de *time-silence* enviará mensagens *null* para $BM[m.b]$ e, em algum momento, $BM[m.b]$ estará completo. Mais ainda, se não existem blocos incompletos depois de um período de tempo ts, mensagens *null* são transmitidas para todos os processos com a informação do *LCB*. Por conta disso, assumindo que as mensagens não são perdidas, em algum momento, todos os processos receberão a informação de *LCB* relacionada à $BM[m.b]$ e, conseqüentemente, $BM[m.b]$ se tornará estável e suas mensagens serão entregues. Em um cenário com falhas, suponha que p_i entregue m e falhe em seguida. Como conseqüência, m não é entregue em processo correto p_j. Se p_i entregou m, então $BM[m.b]$ é estável em p_i e, conseqüentemente, completo em todos os processos (incluindo p_j). Graças aos *timeouts* para a estabilidade do bloco, a falha de p_i iniciará uma difusão confiável a fim de executar a tarefa de mudança de visão. Por conta da propriedade de entrega da primitiva de difusão confiável, essa mensagem é recebida por todos os processos e, em algum momento, o consenso será executado. Uma vez que, $BM[m.b]$ é completo em todos os processos, m estará presente na visão de todos os processos proposta no consenso e, conseqüentemente, será entregue a todos os processos em

funcionamento (incluindo p_j) antes que a nova visão seja instalada – graças à propriedade de acordo uniforme do consenso de Chandra e Toueg (1996). ∎

5.4 O MECANISMO AUTONÔMICO

A abordagem autonômica estende o protocolo básico de comunicação em grupo para lidar com a questão do ajuste do *time-silence* em ambientes computacionais dinâmicos. Em tais cenários, escolher um valor apropriado para o *time-silence* é um desafio por que: a) *time-silences* muito longos diminuem a sobrecarga de mensagens e o consumo de recursos, mas podem implicar em longos tempos de bloqueio na entrega das mensagens quando os processos não estão enviando mensagens da aplicação para completar os blocos; b) em contrapartida, *time-silences* muito pequenos reduzem o tempo de bloqueio, mas incrementam a sobrecarga de mensagens e o consumo de recursos do protocolo, o que pode ser um problema quando o ambiente computacional está sujeito a altas condições de carga, levando a atrasos mais longos na recepção e entrega das mensagens.

Para atender tal relação de compromisso, é usada a teoria de controle (Hellerstein et al., 2004) para ajustar dinamicamente o *time-silence*. De modo similar ao apresentado no Capítulo 4, a idéia básica é a implementação de um laço de controle em que um gestor autonômico monitora o comportamento do protocolo de comunicação em grupo (ou planta) e executa um algoritmo de controle (ou lei de controle) para definir parâmetros operacionais que assegurem que o comportamento do protocolo atenda as expectativas do usuário (i.e. *set-point* ou comportamento esperado). Para tanto, o laço de controle implementado monitora e controla a estrutura de blocos causais embutidos em cada processo distribuído. Esse laço de controle é composto por dois componentes básicos, denominados *sensor* e *controlador*, ver Figura 5.3.

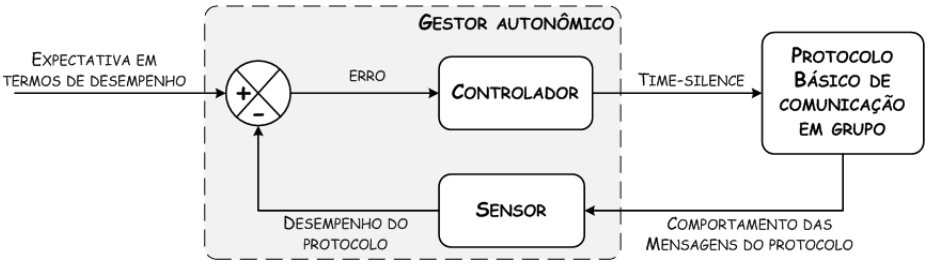


Figura 5.3. Laço de controle implementado pela abordagem autonômica

O componente *sensor* é responsável pelo sensoriamento e estimativa das informações a respeito do ambiente computacional e do protocolo de comunicação em grupo (e.g. atrasos fim-a-fim, sobrecarga do protocolo, carga da aplicação etc.). O componente *controlador*, por sua vez, usa as informações sensoriadas e estimadas para regular dinamicamente o *time-silence* considerando o comportamento desejado (i.e. *ponto de operação* ou *set-point*). Esses *set-points* são expressos em termos da sobrecarga de mensagens, i.e. percentual de mensagens de controle, transmitidas para completar os blocos causais e usadas nos gerenciamento das visões

de grupo. A definição dos valores de *set-point* depende do número atual de processos no grupo e da carga do sistema. Uma vez que essas condições podem variar em tempo de execução, os *set-points* devem ser recalculados dinamicamente de acordo com o percentual de consumo de recursos desejado. Assim, o *set-point* dinâmico é definido a partir de: i) requisitos do usuário em termos de consumo de recursos; ii) consumo de recursos atual no ambiente computacional; e iii) número de membros no grupo de processos.

Em sistemas distribuídos abertos (e.g. Internet), a quantidade real de recursos disponíveis no sistema é extremamente difícil de ser determinada. Por conta disso, neste trabalho, o consumo de recursos é expresso como uma metáfora para representar uma medida relativa que depende da disponibilidade dos recursos em certo instante. A idéia por traz de tal metáfora é que mais recursos (como largura de banda na rede, número de nós, memória disponível etc.) produz atrasos computacionais mais curtos, e vice-versa. Por exemplo, se um nó possui pouca memória ou ocorrem falhas, os atrasos computacionais serão mais longos, isso pode ser indicativo de que um percentual menor de recursos está disponível. Na perspectiva dos processos de uma computação distribuída, o ambiente distribuído é uma caixa preta, assim, a percepção dos mesmos a cerca dos recursos disponíveis se manifesta a partir dos atrasos de comunicação fim-a-fim – os quais englobam atrasos computacionais tanto relacionados ao processamento quanto a transmissão das mensagens. Mais precisamente, o consumo de recursos é tido como uma função dos atrasos de comunicação fim-a-fim, como segue: a) se o atraso médio fim-a-fim é próximo do atraso mínimo observado, então o atual consumo de recursos é próximo de zero; b) se o atraso médio fim-a-fim é próximo do máximo observado, então o atual consumo de recursos é próximo de um; c) caso contrário, o consumo de recursos é um valor entre zero e um.

Detalhes sobre a implementação dos componentes de sensoriamento (i.e. *sensor*) e de regulação de protocolo (i.e. *controlador*) são descritos no restante desta seção. Uma lista com a descrição de todo o conjunto de variáveis usadas pelo gestor autonômico é apresentada no Apêndice B.

5.4.1 Sensoriamento do ambiente e do protocolo de comunicação em grupo

Na implementação da facilidade de gestão autonômica proposta, o componente sensor colabora, fundamentalmente, na coleta, tratamento e distribuição das informações a cerca do protocolo básico de comunicação em grupo e do ambiente computacional. Para tanto, o mesmo realiza a execução de duas tarefas básicas, denominadas sensoriamento e transdução, respectivamente, ver Figura 5.4.

A tarefa de sensoriamento tem como responsabilidade:

i) observar o comportamento do protocolo, através da contabilidade do total de mensagens recebidas (n_{rv}) e do total de mensagens de controle (n_{ct}) – definidas pelas mensagens *null* do mecanismo de *time-silence* e pelas mensagens relacionadas aos subprotocolos de controle de *membership* e mudança de visão;

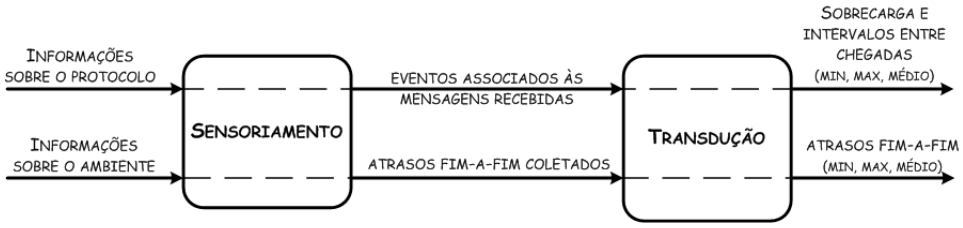


Figura 5.4. Diagrama do componente sensor

ii) observar o comportamento do ambiente, através da interação com os mecanismos do sistema de comunicação subjacente para extrair informações a respeitos dos atrasos de ida-e-volta observados (i.e. *rtt*, *round-trip-time*)[4];

iii) registrar os instantes de chegada (A, *arrival time*) das mensagens de aplicação.

O Algoritmo 5.5 apresenta o procedimento usado pela tarefa de sensoriamento. A cada mensagem recebida, o procedimento contabiliza o recebimento da mesma realizando um incremento em n_{rv} (linha 2). Caso a mensagem recebida seja uma mensagem de controle, o incremento em n_{ct} também é realizado (linhas 1–4). Caso contrário, o mesmo armazena o instante de chegada da mensagem, considerando o processo remetente (linha 6). Por fim, o procedimento coleta o último atraso de ida-e-volta estimado (linha 8).

Algoritmo 5.5: *Componente sensor: Tarefa de sensoriamento*

```
1 on event (m_k receiving) at p_i from p_j do
2     n_rv = n_rv + 1;
3     if m is a control message then
4         n_ct = n_ct + 1;
5     else
6         A_k[j] = clock(p_i);
7     end if
8     obtain rtt from FIFO channels ;
9 end event
```

A tarefa de transdução, por sua vez, é responsável por receber os dados da tarefa de sensoriamento e realizar as seguintes atividades: a) estimar o desempenho do protocolo em termos da sobrecargas máxima (ovh_{max}) e atual (ovh); b) estimar os atrasos fim-a-fim máximo (d_{max}), mínimo (d_{min}) e médio (d_{mean}); c) estimar a carga média de trabalho (*workload*) das aplicações do grupo, usando, para tanto, o intervalo entre chegada das mensagens (τ); d) realizar estimativas a respeito do limite máximo do *time-silence*, usando estimativas do intervalo máximo entre chegadas; e e) usar o atrasos estimados para obter estimativas a respeito do percentual de consumo de recursos no ambiente computacional (*RC*).

[4] observe que o protocolo básico de comunicação em grupo é construído considerando canais FIFO confiáveis (ver Seção 5.2), os quais, em geral, são implementados usando técnicas de retransmissão baseadas em estimativas de atraso de ida-e-volta, de modo a tolerar perda de mensagens e evitar retransmissões desnecessárias, ver Tanenbaum (2003).

Essas variáveis são usadas, mais adiante, nas estimativas dos *set-points* dinâmicos e na regulação do *time-silence*. Por conta disso, uma descrição mais detalhada de cada uma das atividades realizadas pela tarefa de transdução é discutida a seguir.

Estimativa da sobrecarga de mensagens do protocolo. Para estimar a sobrecarga média introduzida pelo protocolo, a tarefa de transdução considera a relação entre o total de mensagens recebidas/enviadas pelo protocolo básico de comunicação em grupo e o total de mensagens de controle recebidas/enviadas pelo mesmo, isto é:

$$ovh = \frac{n_{ct}}{n_{rv}} \tag{5.1}$$

A sobrecarga máxima é estimada considerando o comportamento do protocolo em um cenário de pior caso, em relação às mensagens *null*. Este cenário, ocorre quando apenas um membro do grupo de processos colabora para a *completude* de um bloco, necessitando, deste modo, que o mecanismo de *time-silence* seja disparado nos demais membros do grupo (ver Figura 5.5).

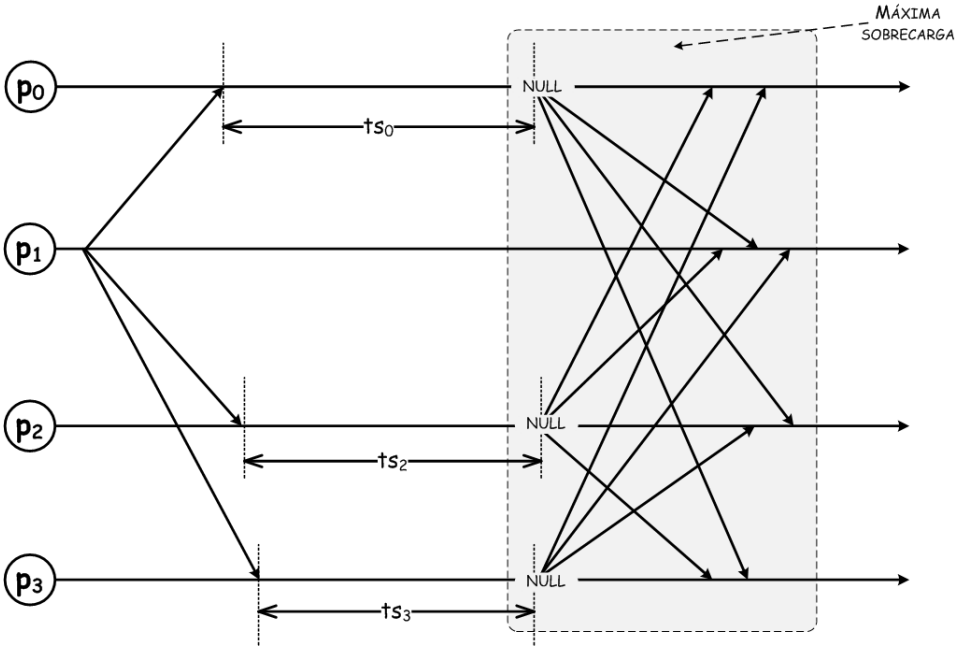


Figura 5.5. Sobrecarga máxima de mensagens de controle em um grupo com quatro membros

Assim, em um cenário de pior caso e considerando um grupo de processos com n_{am} membros ativos, a sobrecarga máxima pode ser estimada usando:

$$ovh_{max} = \frac{n_{am} - 1}{n_{am}} \tag{5.2}$$

Estimativa dos atrasos fim-a-fim. O comportamento do ambiente computacional é expresso a partir dos atrasos fim-a-fim, isto é, a partir dos atrasos atual (d), médio (d_{mean}), máximo (d_{max}) e mínimo (d_{min}). Esses atrasos são importantes tanto para que o gestor autonômico possa realizar

uma estimativa do percentual do consumo de recursos, quanto na definição dos timeouts de completude ($TC1$ e $TC2$) e estabilidade dos blocos causais ($ST1$ e $ST2$). Assim, a tarefa de transdução estima o k^{th} tempo de viagem de mensagem usando o k^{th} intervalo de ida-e-volta sensoriado, isto é:

$$d_k = \frac{rtt_k}{2} \tag{5.3}$$

Observe que o fato dos tempos de viagem das mensagens, de um processo para outro (i.e. de um processo p_i para um processo p_j e vice-versa) poderem ser diferentes, tem pouca relevância para a estimativa do consumo de recursos usada na atividade de regulação, ver discussão na Seção 4.3.1.1 do Capítulo 4.

A tarefa de transdução estima o atraso médio (d_{mean}) usando a seguinte média móvel:

$$d_{mean_k} = \alpha * d_{mean_{k-1}} + (1-\alpha) * d_k \tag{5.4}$$

em que α é o parâmetro de ponderação usado para determinar o tamanho do histórico a ser usado no cálculo da média – para o histórico dos w últimos atrasos fim-a-fim, use-se:

$$\alpha = \frac{w-1}{w}.$$

O atraso mínimo (d_{min}) é estimado como o menor atraso fim-a-fim observado durante a execução do protocolo, isto é:

$$d_{min_k} = min(d, d_{min_{k-1}}) \tag{5.5}$$

em que $min(x,y) = x$, se $x < y$; e y, caso contrário.

O atraso máximo (d_{min}), por sua vez, é o maior atraso fim-a-fim observado, considerando, ainda, uma margem de segurança $\beta > 0$, isto é:

$$d_{max_k} = (1+\beta) * max(d, d_{max_{k-1}}) \tag{5.6}$$

em que $max(x,y) = x$, se $x > y$; e y, caso contrário.

Note que o atraso máximo contribui, de forma significativa, na definição dos *timeouts* de completude e estabilidade dos blocos causais usados pelo protocolo básico de comunicação em grupo (ver Seção 5.3). Assim, subestimar o atraso máximo pode provocar um maior número de falsas suspeitas de falhas ou levar a execuções desnecessários do protocolo de *membership* – estes aspectos justificam o uso da margem de segurança (β) na estimativa de tal variável.

Os limites estimados d_{max} e d_{min} dependem das características do ambiente computacional. Desse modo, se as características do ambiente computacional podem mudar com o tempo, então os valores de d_{max} e d_{min} também podem mudar. Com isso, os valores observados pela tarefa de transdução podem não ser válidos de uma execução para outra, isto é: mudanças nos recursos

disponíveis, por conta de falhas ou reconfiguração de canais, por exemplo, pode implicar em um aumento no valor do atraso mínimo ou em uma redução no valor do atraso máximo. Com isso, para acomodar tais variações dinâmicas nas estimativas de d_{max} e d_{min}, um fator de esquecimento ($0 < \phi < 1$) é usado, de tal modo que as amostras mais recentes dos atrasos influenciem nos valores de d_{max} e d_{min}. Sendo assim:

$$d_{max_k} = \phi * d_{max_{k-1}} + (1 - \phi) * d \tag{5.7}$$

e

$$d_{min_k} = \phi * d_{min_{k-1}} + (1 - \phi) * d \tag{5.8}$$

Estimativa dos limites para o *time-silence*. A determinação dos limites do *time-silence* é um aspecto importante para prover garantias de desempenho do gestor autonômico durante as mudanças das característica do ambiente. Note que, o menor *time-silence* possível é $ts = 0$, o qual determina que os membros do grupo enviem mensagens *null* imediatamente após receber uma mensagem de aplicação de um dos demais membros. Esse modo de funcionamento (imposto por $ts = 0$), leva o protocolo básico de comunicação em grupo a um tempo de bloqueio mínimo e à sobrecarga máxima – o que não representa necessariamente um problema, pois a sobrecarga máxima (com $ts = 0$) somente será induzida pelo gestor autonômico nos momentos nos quais as condições de carga do ambiente e os requisitos do usuário em termo do consumo de recursos permitirem.

Por outro lado, deixar que o limite superior do *time-silence*, seja definido pelas condições do ambiente ou requisitos do usuário pode ser um problema. Isto por que certas condições com pouca disponibilidade de recursos no ambiente, severas restrições em termos dos requisitos de expectativas de custo determinada pelo usuário ou baixas condições de carga, podem levar o gestor autonômico a sugerir *time-silences* extremamente elevados, podendo, em alguns casos, implicar em $ts \rightarrow \infty$ – representando, portanto, um problema para o atendimento das propriedades de *liveness* (i.e. terminação) do protocolo básico de comunicação em grupo.

Contudo, é suficiente que o limite superior do *time-silence* seja um valor finito maior que o maior intervalo entre chegadas de mensagens de aplicação observado – isto garante que existirá uma menor (ou nenhuma) ativação do mecanismo do *time-silence* e, portanto, a sobrecarga de mensagens de controle será mínima. Assim, o limite superior do *time-silence* é obtido a partir do maior intervalo entre chegadas observado das mensagens de aplicação vindas de membros ativos do grupo e considerando uma margem de segurança ($\beta > 0$), isto é:

$$ts_{max} = (1 + \beta) * \tau_{max} \tag{5.9}$$

em que τ_{max} é o maior intervalo entre chegadas observado durante a execução do protocolo de comunicação em grupo.

O intervalo entre chegadas de mensagens da aplicação (τ) pode ser calculado usando:

$$\tau_k[i] = A_k[i] - A_{k-1}[i] \tag{5.10}$$

em que, $A_k[i]$ representa o instante de chegada da k^{th} mensagem de aplicação enviada por um processo p_i e recebida por algum processo p_j. Além disso, assume-se $\tau_0 = ts_0$, em que ts_0 representa o valor inicial do *time-silence* definido pelo usuário.

Observe que as características de carga da aplicação pode mudar com o tempo. Assim, intervalo máximo entre chegadas pode mudar também. Conseqüentemente, o τ_{max}, verificado durante a execução, pode não ser válido de uma execução para outra – podendo implicar em valores desnecessariamente longos de *time-silence*. Por conta disso, também é considerado um fator de esquecimento no cálculo do de τ_{max}, isto é:

$$\tau_{max_k} = \phi * \tau_{max_{k-1}} + (1-\phi) * \tau_k \tag{5.11}$$

Estimativa do percentual de consumo de recursos. A metáfora usada para determinar o percentual de consumo de recursos (*RC*), a partir dos atraso de comunicação fim-a-fim (d), considera que a variação do percentual de consumo de recursos é proporcional à variação observada para o atraso de comunicação fim-a-fim, ou seja: quanto maior o atraso de comunicação fim-a-fim, maior o percentual de consumo de recursos; e vice-versa – ver Figura 5.6.

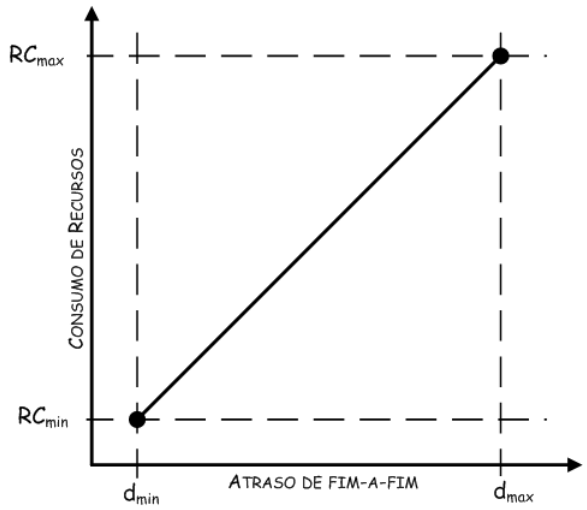


Figura 5.6. Relação entre o atraso fim-a-fim e o consumo de recursos

Por conta disso, representa-se a relação entre *RC* e d, através de uma função linear, descrita da seguinte forma:

$$\frac{RC_k - RC_{min}}{RC_{max} - RC_{min}} = \frac{d_k - d_{min}}{d_{max} - d_{min}} \tag{5.12}$$

O percentual de consumo de recursos é observado a partir do *jitter*, isto é da variação do atraso atual entre os atrasos mínimo e máximo, significando que: se o atraso atual é igual ao

mínimo ($d = d_{min}$), então *RC* é mínimo; por outro lado, se o atraso atual é igual ao máximo ($d = d_{max}$), então *RC* é máximo.

Assim, substituindo $RC_{min} = 0$ e $RC_{max} = 1$ na expressão da Equação 5.12, é possível determinar:

$$RC_k = \frac{d_k - d_{min}}{d_{max} - d_{min}} \tag{5.13}$$

Algoritmo da tarefa de transdução. O Algoritmo 5.6 apresenta todo o procedimento usado pela tarefa de transdução para realizar a estimativa das variáveis consideradas (i.e. d, τ, ts_{max}, *RC* etc.). Esse procedimento é ativado a cada evento de entrega de mensagens, então calcula os valores atual e máximo para a sobrecarga do protocolo (linhas 2–3). Em seguida, estima os valores atual, médio, máximo e mínimo para o atraso fim-a-fim (linhas 4–12). Depois disso, o intervalo máximo entre chegadas é obtido (linhas 13–14) e usado para determinar o *time-silence* máximo (linha 15). Por fim, o percentual de consumo de recursos é calculado, respectivamente (linha 16).

Algoritmo 5.6: *Componente sensor: Tarefa de transdução*

1 ***on event*** $(m$ *delivery* $)$ ***at*** p_i ***from*** p_j ***do***
2 ***estimate*** *the actual overhead* $ovh_i = \frac{n_{ct}}{n_{rv}}$;
3 ***estimate*** *the maximum overhead* $ovh_{max} = \frac{(n_{am}-1)}{n_{am}}$;
4 ***estimate*** *the end-to-end delay* $d = \frac{rtt}{2}$;
5 ***if*** d_{mean} *is unset* ***then***
6 ***set*** $d_{mean} = d$; ***set*** $d_{max} = d$; ***set*** $d_{min} = d$;
7 ***end if***
8 ***estimate*** $d_{mean} = \alpha * d_{mean} + (1-\alpha) * d$;
9 ***if*** $d_{max} < d$ ***then*** *compute* $d_{max} = (1+\beta) * d$;
10 ***if*** $d_{min} > d$ ***then*** *compute* $d_{min} = d$;
11 ***compute*** $d_{min} = \phi * d_{min} + (1-\phi) * d$;
12 ***compute*** $d_{max} = \phi * d_{max} + (1-\phi) * d$;
13 ***if*** $\tau_{max} < \tau[j]$ ***then*** $\tau_{max} = \tau[j]$;
14 ***compute*** $\tau_{max} = \phi * \tau_{max} + (1-\phi) * \tau[j]$;
15 ***compute*** $ts_{max} = (1+\beta) * \tau_{max}$;
16 ***estimate*** $rc = \frac{d_{mean} - d_{min}}{d_{max} - d_{min}}$;
17 ***end event***

5.4.2 Regulação do protocolo de comunicação em grupo

O gestor autonômico usa um componente, denominado *controlador*, o qual dispõe das informações obtidas pelo sensor e dos requisitos definidos pelo usuário (em termos de custos, baseados na metáfora de consumo de recursos) para regular o desempenho do protocolo. Para

tanto, o componente controlador é composto por duas tarefas básicas (ver Figura 5.7): o estimador de *set-points* e o regulador de *time-silence*.

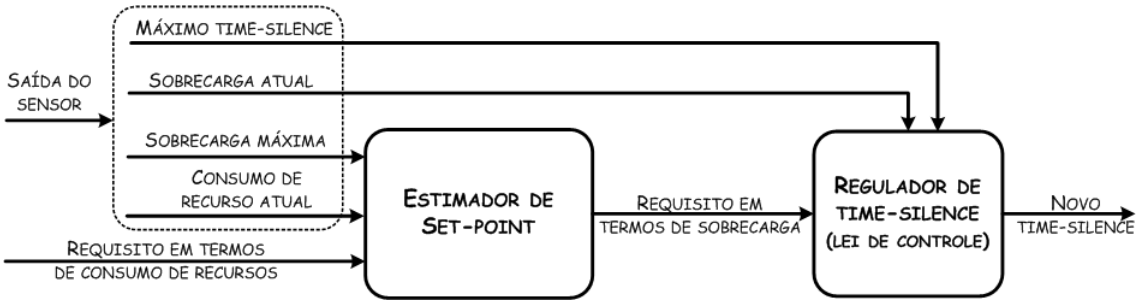


Figura 5.7. Diagrama do componente controlador

O estimador de *set-points* usa os requisitos definidos e o desempenho do protocolo sensoriado, ambos em termos do custos e baseados na metáfora de consumo de recursos, para definir *set-points* dinâmicos em termos de sobrecarga de mensagens. O regulador usa uma lei de controle para ajustar o *time-silence* de modo a alinhar o desempenho do protocolo com as expectativas (*set-points*) em termos de sobrecarga de mensagens. Uma maior discussão dos detalhes de implementação associados a cada uma dessas tarefas é apresentada a seguir.

5.4.2.1 Estimativa de set-points dinâmicos

As estimativas dos *set-points* dinâmicos (i.e. pontos de referência) para a operação do regulador de *time-silence* se baseiam em uma premissa simples: se o consumo de recursos estimado (RC) é maior que o desejado pelo usuário (RC_D), então o *set-point* deve ser modificado para que o protocolo trabalhe em modo de baixo consumo (reduzindo a sobrecarga de mensagens de controle); por outro lado, se o consumo de recursos está abaixo do especificado, o *set-point* deve ser modificado para permitir que o protocolo trabalhe em um modo de operação que beneficie o latência de bloqueio, adequadamente aumentando a sobrecarga e, conseqüentemente, o consumo de recursos – ver Figura 5.8.

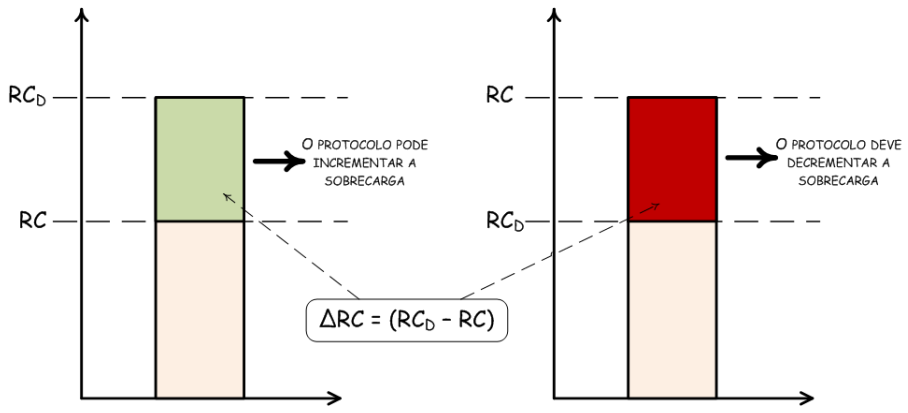


Figura 5.8. Operação do gestor autonômico face ao desvio entre os valores atual e desejado para o consumo de recursos

O consumo de recursos atual é resultante da carga de trabalho (*workload*) das aplicações associada à carga (em termos de mensagens de controle) proveniente do protocolo básico de comunicação em grupo. Uma vez que, as mensagens de controle representam um custo adicional em termos de consumo de recursos, o estimador de *set-points* deve acomodar o consumo de recursos (ou custo) do protocolo de modo a atender as expectativas do usuário.

Nesse sentido, se o consumo atual é RC e o consumo desejado é RC_D, então a sobrecarga proveniente do protocolo deve ser acomodada em $\Delta RC_D = RC_D - RC$. Portanto, o ponto de operação (i.e. *set-point*) do protocolo deve ser aquele que permita que a sobrecarga do protocolo determine um consumo de recursos residual dito $RC_P = \Delta RC_D$, em que RC_P é o consumo de recursos associado a sobrecarga de controle do protocolo. Para que isso seja possível, entretanto, é necessário determinar a relação entre sobrecarga e consumo de recursos.

Para tanto, a estimativa dos pontos de referência (*set-points*) explora a metáfora do consumo de recursos na perspectiva do protocolo de comunicação em grupo (ver Figura 5.9), isto é: quando a sobrecarga imposta pelo protocolo de comunicação em grupo é máxima (i.e. $ovh = ovh_{max}$), então o consumo de recursos do protocolo em termos de mensagens de controle também é máximo (i.e. $RC_P = 1$ ou $RC_P = RC_{max}$); por outro lado, quando a sobrecarga imposta pelo protocolo é mínima (i.e. $ovh = 0$), da mesma forma, o consumo de recursos relacionado é mínimo (i.e. $RC_P = 0$ ou $RC_P = RC_{min}$); caso contrário, o consumo de recurso é algo entre zero e um (i.e. $0 < RC_P < 1$ ou $RC_{min} < RC_P < RC_{max}$)[5].

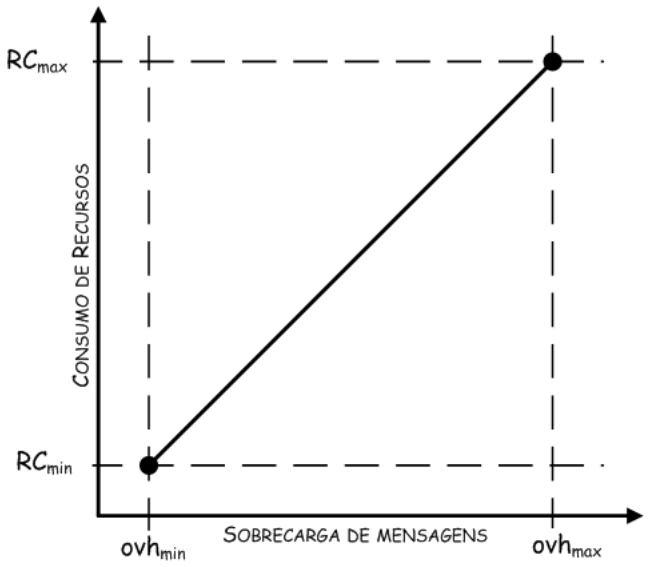


Figura 5.9. Relação entre sobrecarga de mensagens e o consumo de recursos

Assim, uma variação na sobrecarga implicará em uma variação proporcional em termos do consumo de recursos do protocolo básico de comunicação em grupo. Com isso, a partir desse relacionamento entre a sobrecarga e o consumo de recursos, é possível estabelecer a relação:

[5]Observe que, apesar de ΔRC_D poder assumir valores negativos, isto apenas indica para o regulador de *time-silence* a direção na qual o ajuste do consumo de recursos do protocolo deve ser realizado – isto é, realizar consumo de recursos negativo equivale a liberar recursos ou reduzir a sobrecarga.

$$\frac{RC_{min} + RC_P}{RC_{max} - RC_{min}} = \frac{ovh_{min} + ovh_D}{ovh_{max} - ovh_{min}} \tag{5.14}$$

Substituindo os limites para a sobrecarga (i.e. $ovh_{min} = 0$) e para o consumo de recursos (i.e. $RC_{max} = 1$ e $RC_{min} = 0$) na expressão da Equação 5.14, é possível obter a relação entre a variação da sobrecarga e a variação do consumo de recursos por:

$$ovh_D = RC_P * ovh_{max} \tag{5.15}$$

em que ovh_D representa a sobrecarga desejada e é usada, mais adiante, como parâmetro de entrada para o mecanismo de regulação do *time-silence*.

O Algoritmo 5.7 apresenta o procedimento usado pelo controlador na estimativa dos *set-points* dinâmicos. A cada entrega de mensagem, o gestor autonômico obtém os valores desejado e atual em termos do consumo de recursos (linhas 2–3). Assim, a partir do desvio entre os consumos de recurso atual e observado (linha 4), o *set-point* em termos de sobrecarga é, então, determinado (linha 5).

Algoritmo 5.7: *Componente controlador: tarefa de estimativa de set-points dinâmicos*

```
1 on event (m delivery ) at p_i do
2     obtain the desired resource consumption RC_D;
3     estimate the current resource consumption RC;
4     obtain the residual resource RC_P = RC_D − RC;
5     compute the dynamic set-point ovh_D = RC_P * ovh_max;
6 end event
```

5.4.2.2 Regulação de time-silence

A tarefa de regulação realiza ajustes continuados no *time-silence* (ts) para permitir que o protocolo atenda ao limiar de sobrecarga desejado (ovh_D), face às mudanças dinâmicas no ambiente ou nos requisitos definidos pelo usuário. Esse procedimento de ajuste requer a definição de uma função (F) que determine um valor de *time-silence* (ou variação do mesmo – Δts) a partir da sobrecarga desejada (ou desvio entre a mesma e a sobrecarga atual, $\Delta ovh_D = ovh_D - ovh$), isto é:

$$\Delta ts_D = F(\Delta ovh_D) \tag{5.16}$$

Além disso, uma vez conhecida a função F, deve-se optar por uma lei de controle, que determine a forma como os ajustes necessários serão realizados no *time-silence*. Para tanto, o mecanismo autonômico proposto é implementado considerando uma lei de controle proporcional, definida da seguinte forma:

$$\Delta ts_D = K_P * F(\Delta ovh_D) \tag{5.17}$$

em que K_P é um parâmetro que representa o ganho proporcional usado pelo controlador para ponderar a variação do *time-silence* com relação à função $F(\Delta ovh)$.

Observe que, a lei de controle usada equivale a incrementos (ou decrementos) sucessivos proporcionais aos valores determinados pela função F. Isto é, uma vez que $\Delta ts_D = ts_D - ts_k$, a Equação 5.17 pode ser rescrita da seguinte forma:

$$ts_D = ts_k + K_P * F(\Delta ovh_k) \tag{5.18}$$

em que ts_D é o *time-silence* desejado, o qual representa o próximo *time-silence* a ser usado pelo protocolo (i.e. $ts_{k+1} = ts_D$).

Na implementação da tarefa de regulação, recorre-se a uma alternativa mais simples, baseada em uma relação linear, isto é: se o *time-silence* é máximo (i.e. $ts = ts_{max}$), então a sobrecarga é mínima (i.e. $ovh = 0$ ou $ovh = ovh_{min}$); por outro lado, se o *time-silence* é mínimo ($ts = 0$), então a sobrecarga é máxima (i.e. $ovh = 1$ ou $ovh = ovh_{max}$); caso contrário, a sobrecarga é algo entre as sobrecargas e mínima e máxima (i.e. $ovh_{min} < ovh < ovh_{max}$) – ver Figura 5.10.

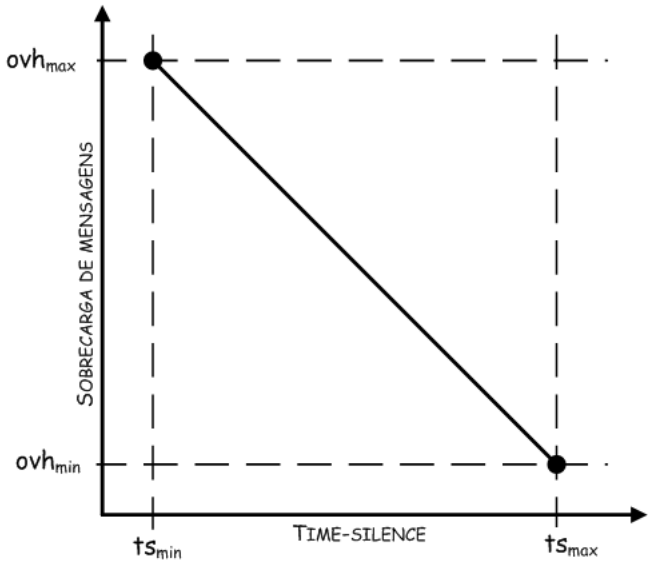


Figura 5.10. Relação entre time-silence e sobrecarga de mensagens

Com isso, é possível estabelecer a relação:

$$\frac{ovh_{min} + \Delta ovh_D}{ovh_{max} - ovh_{min}} = -\left(\frac{ts_{min} + \Delta ts_D}{ts_{max} - ts_{min}}\right) \tag{5.19}$$

Usando $ts_{min} = 0$ e $ovh_{min} = 0$, é possível reescrever a Equação 5.19 da seguinte forma:

$$\Delta ts_D = -\left(\frac{\Delta ovh_D}{ovh_{max}}\right) * ts_{max} \tag{5.20}$$

Uma vez que, $F(\Delta ovh_D) = \Delta ts_D$, então, a partir das equações 5.16 e 5.20, tem-se:

$$F(\Delta ovh_D) = -\left(\frac{\Delta ovh_D}{ovh_{max}}\right) * ts_{max} \tag{5.21}$$

Note que, a definição de F, na Equação 5.21, usa informações sobre os principais fatores que impactam no desempenho do protocolo, isto é: número de membros do grupo (através de ovh_{max}, ver Equação 5.2); taxa de envio da aplicação (através de ts_{max}, ver Equação 5.9); e requisitos do usuário e atrasos fim-a-fim (através de Δovh_D, ver equações 5.15, 5.14 e 5.13).

O Algoritmo 5.8 apresenta o procedimento usado pelo controlador. A cada entrega de mensagem o controlador obtém as estimativas de sobrecarga atual (ovh) e desejada (ovh_D) para o protocolo (linhas 3–2), calculando, em seguida, o desvio entre as mesmas (linha 4). Então, o desvio é utilizado para estimar a variação desejada para o *time-silence* (linha 5). Por fim, o valor do time-silence é calculado e limitado entre os valores máximo (ts_{max}) e mínimo ($ts_{min} = 0$) – ver linhas 6–8.

Algoritmo 5.8: *Componente controlador: Tarefa de regulação do time-silence*

```
1 on event (m delivery ) at p_i do
2     obtain the dynamic set-point ovh_D;
3     obtain the current overhead ovh;
4     compute Δovh_D = ovh_D − ovh;
5     compute F = −(Δovh_D / ovh_max) * ts_max;
6     compute ts = ts + K_P * F;
7     if ts < 0 then ts = 0;
8     if ts > ts_max then ts = ts_max;
9 end event
```

5.5 AVALIAÇÃO DE DESEMPENHO

5.5.1 Descrição do ambiente de simulação

Os experimentos consideram um sistema distribuído assíncrono implementado no simulador HDDSS (Freitas; Macêdo, 2009; Macêdo; Freitas, 2009). Os processos e canais de comunicação formam uma rede completamente conectada e os atrasos variam de acordo com uma função de distribuição de probabilidade. Cada canal apresenta um comportamento *log-normal* (com média 10*ms* e desvio padrão de 5*ms*) e também é afetado pela quantidade de tráfego na rede (i.e. os atrasos fim-a-fim aumentam devido ao efeito do enfileiramento de mensagens).

Nas avaliações, os fatores de simulação são o número de nós da rede ($n = 5$, 15 e 25 nós) e o período de *time-silence*. O mecanismo de blocos causais original (*CB*) com *time-silence* fixo

($ts = 20ms$ e $ts = 200ms$) é comparado com a versão auto-gerenciável (*ACB*) em que o *time-silence* é ajustado dinamicamente para atender a um *set-point* desejado em termos de consumo de recursos ($RC_D = 25\%$) – os parâmetros do gestor autonômico são configurados usando $K_P = 4$, $\alpha = 0,1$ e $\beta = 0,1$ e $\phi = 0,99999$ (ver Apêndice B). Além disso, são considerados, nas simulações, cenários com e sem falhas em membros dos grupos e cenários nos quais os *set-points* definidos variam durante a execução.

Para simular a geração de mensagens de aplicação, cada processo usa a função de distribuição de probabilidade de *Bernoulli* para decidir quando enviar uma mensagem, seguindo três diferentes taxas de transmissão: 100, 150 e 80 mensagens por segundo – essas taxas de transmissão são denominadas perfis de carga moderada, alta e baixa, respectivamente. Como métricas de desempenho são considerados o tempo de bloqueio (i.e. o atraso entre a recepção e a respectiva entrega de uma mensagem) e a sobrecarga de mensagens do protocolo.

5.5.2 Resultados obtidos

Para obtenção dos resultados, cada série de experimentos foi repetida três vezes e as métricas são observadas em termos de seus valores médios e respectivos intervalos de confiança (com nível de confiança de 95%). Nos experimentos, o tempo é medido em *ms* enquanto que a sobrecarga é medida em valores percentuais. A Tabela 5.1 apresenta um resumo dos resultados obtidos em termos das métricas de desempenho consideradas. Uma análise mais detalhada dos dados apresentados na Tabela 5.1 é realizada a seguir.

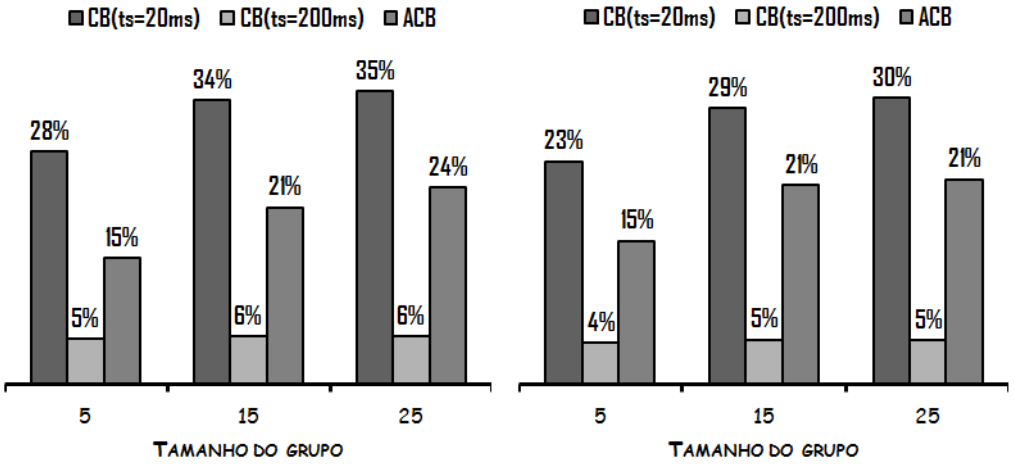
Tabela 5.1. Resultados dos experimentos para diferentes perfis de carga e tamanhos de grupo

Fatores		tempo de bloqueio (ms)			sobrecarga (%)		
n	Perfil de carga	$CB(ts = 20ms)$	$CB(ts = 200ms)$	*ACB*	$CB(ts = 20ms)$	$CB(ts = 200ms)$	*ACB*
5	Baixa	$40,44 \pm 11,92$	$79,81 \pm 36,47$	$54,17 \pm 20,08$	$27,9\% \pm 0,28\%$	$5,44\% \pm 0,04\%$	$15,18\% \pm 0,19\%$
	Moderada	$38,44 \pm 11,51$	$70,31 \pm 32,84$	$46,28 \pm 16,05$	$23,30\% \pm 0,11\%$	$4,37\% \pm 0,02\%$	$15,00\% \pm 0,04\%$
	Alta	$35,75 \pm 10,93$	$58,03 \pm 27,36$	$37,77 \pm 12,36$	$16,01\% \pm 0,18\%$	$2,94\% \pm 0,04\%$	$12,98\% \pm 0,19\%$
15	Baixa	$48,18 \pm 10,07$	$108,76 \pm 35,97$	$65,22 \pm 16,55$	$34,00\% \pm 0,06\%$	$5,80\% \pm 0,01\%$	$21,18\% \pm 0,30\%$
	Moderada	$46,12 \pm 9,80$	$97,52 \pm 33,30$	$55,57 \pm 13,74$	$28,80\% \pm 0,04\%$	$4,70\% \pm 0,01\%$	$20,80\% \pm 0,01\%$
	Alta	$42,84 \pm 9,31$	$79,94 \pm 27,48$	$44,15 \pm 10,07$	$20,43\% \pm 0,07\%$	$3,17\% \pm 0,01\%$	$19,27\% \pm 0,11\%$
25	Baixa	$51,23 \pm 9,38$	$102,39 \pm 35,79$	$63,91 \pm 16,06$	$35,10\% \pm 0,04\%$	$5,77\% \pm 0,01\%$	$23,61\% \pm 0,84\%$
	Moderada	$49,44 \pm 9,18$	$107,90 \pm 32,79$	$56,50 \pm 13,84$	$29,90\% \pm 0,01\%$	$4,67\% \pm 0,01\%$	$21,40\% \pm 3,03\%$
	Alta	$45,52 \pm 8,68$	$87,90 \pm 27,37$	$46,23 \pm 9,26$	$21,39\% \pm 0,01\%$	$3,16\% \pm 0,01\%$	$21,49\% \pm 0,06\%$

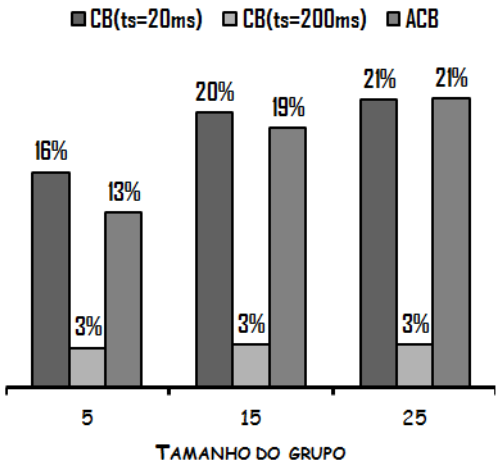
5.5.2.1 Análise do desempenho em termos da sobrecarga de mensagens

Os desempenhos médios em termos da sobrecarga de mensagens de controle, tanto para o protocolo de comunicação em grupo baseado em blocos causais (dito *CB*) quanto para a abordagem autonômica (*ACB*), são apresentados na Figura 5.11. Todos os resultados obtidos em termos dessa métrica possuem valores centrados na média, com intervalos de confiança

muito pequenos (ver Tabela 5.1). Portanto, as análises a seguir são conduzidas observando apenas os valores médios encontrados.



(a) Carga baixa ($80 msgs/s$) (b) Carga moderada ($100 msgs/s$)



(c) Carga alta ($150 msgs/s$)

Figura 5.11. Sobrecargas médias para diferentes tamanhos de grupos e perfis de carga

As diferentes configurações de CB com *time-silences* fixados em $ts = 20ms$ e $ts = 200ms$, ditas $CB(ts = 20ms)$ e $CB(ts = 200ms)$, obtiveram, respectivamente, o pior e o melhor desempenho em termos da sobrecarga média de mensagens (para todos os perfis de carga e composições de grupo considerados) – como pode ser verificado na Figura 5.11.

Para $CB(ts = 20ms)$, na medida em que a carga da aplicação aumenta, a sobrecarga com mensagens de controle diminui – uma vez que menos mensagens *null* são necessárias para que a completude dos blocos seja alcançada (ver Figura 5.11). Entretanto, ainda para $CB(ts = 20ms)$, na medida em que o número de membros do grupo aumenta a sobrecarga aumenta em baixas condições de carga, mas os incrementos na sobrecarga tende a diminuir na medida em que a carga da aplicação aumenta.

Para $CB(ts = 200ms)$, por outro lado, o aumento da carga da aplicação ou do número de processos no grupo (nos cenários considerados) tem pouco efeito na sobrecarga, uma vez que

essa configuração possui um *time-silence muito* largo – note que no caso do perfil de carga mais baixo (80 mensagens por segundo), a aplicação impõe um intervalo médio entre emissões de mensagens de $12,5ms$ (i.e. $1/80$), o que é muito inferior ao *time-silence* ($ts = 200ms$) adotado e justifica o bom desempenho em termos de sobrecarga.

A abordagem autonômica (*ACB*) apresenta desempenho intermediário em termos de sobrecarga e tende, por sua vez, a manter a sobrecarga em torno do *set-point*, independentemente do perfil de carga. No caso de *ACB*, as variações no desempenho em termos de sobrecarga estão associados ao aumento no limiar de sobrecarga máxima imposto pela mudança no número de membros no grupo – observe que, no pior caso, o *set-point* dinâmico (ovh_D) equivale a $ovh_{max} * RC_D$, implicando em ovh_D menor ou igual a $0,20$ (i.e. $4/5 * 0,25$), $0,23$ (i.e. $14/15 * 0,25$) e $0,24$ (i.e. $24/25 * 0,25$) para grupos de processos com 5, 15 e 25 membros, respectivamente.

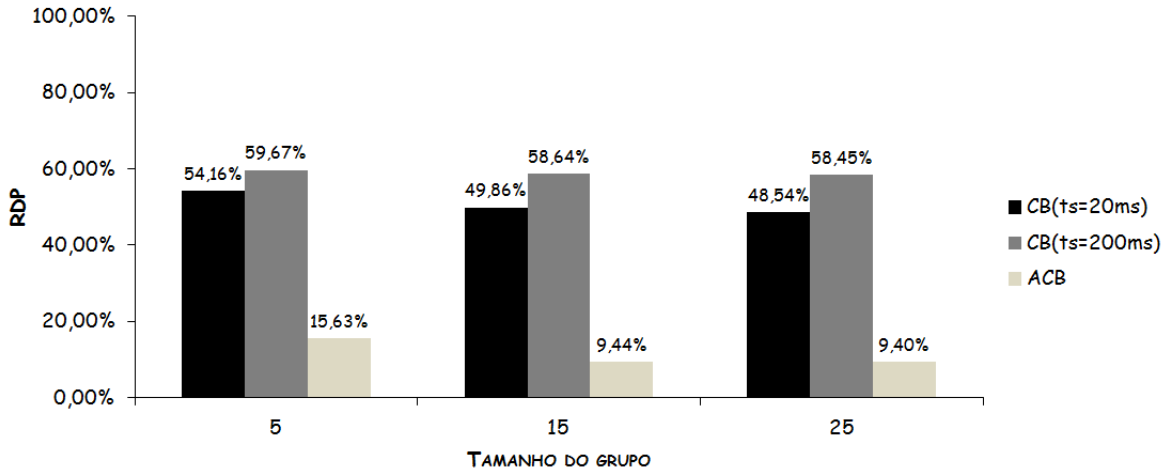


Figura 5.12. Diferença percentual relativa em termos da sobrecarga

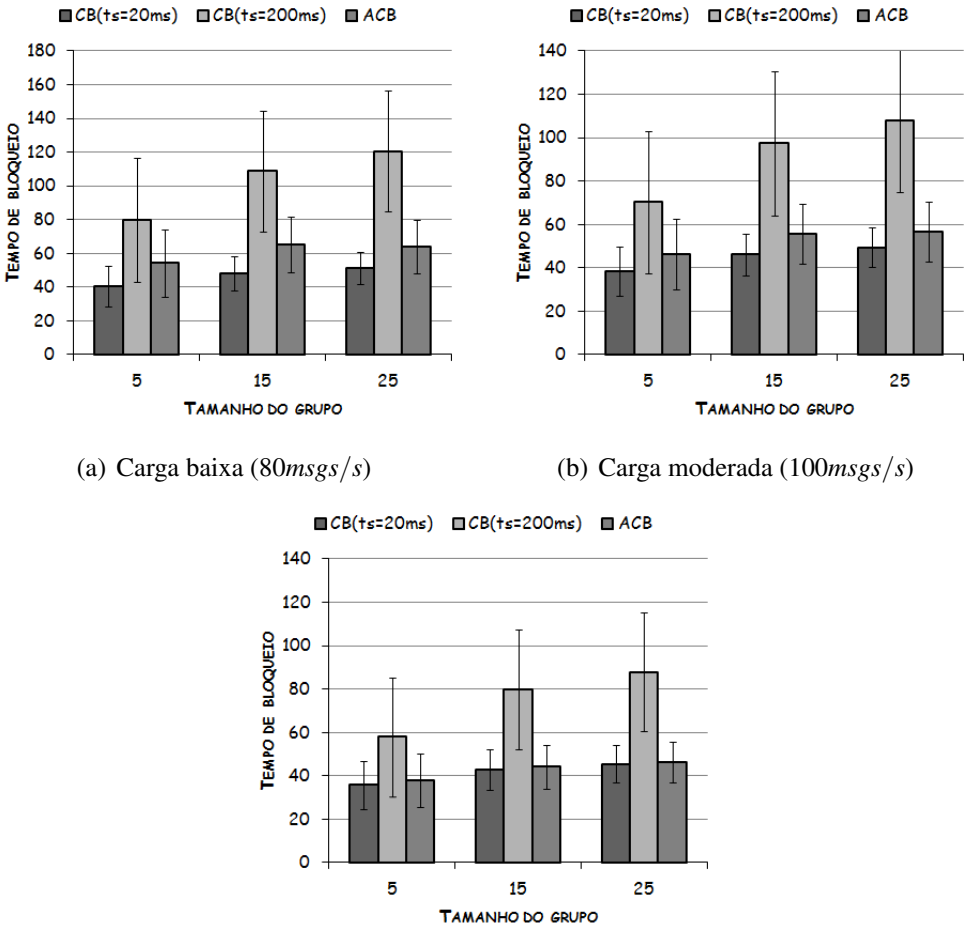
A eficiência da abordagem autonômica é mais expressiva quando se realiza uma análise comparativa das variações de sobrecargas, nas diferentes condições de carga e tamanhos de grupo. Essas variações são calculadas usando a Diferença Percentual Relativa (*RDP*)[6] – ver Figura 5.12. Para tanto, calcula-se a RDP entre as sobrecargas máxima (medida em baixo perfil de carga) e mínima (medida em alto perfil de carga).

Para o protocolo básico com *time-silence* fixo, $ts = 20ms$ e $ts = 200ms$, observa-se altas variações de sobrecarga (i.e. *RDP* de $48,54\%$ a $59,67\%$, de acordo com o ts e o tamanho do grupo). Enquanto que, a abordagem autonômica apresenta pequena variação, possuindo *RDP* de $9,44\%$ a $15,63\%$, de acordo com o tamanho do grupo (ver Figura 5.12) – o que demonstra a eficiência da abordagem autonômica em manter a sobrecarga próximo ao ponto de operação definido.

[6] A Diferença Percentual Relativa (RDP, *Relative Percent Diferent*) pode ser calculada dividindo a diferença absoluta de dois valores pela média aritmética simples dos mesmos.

5.5.2.2 Análise do desempenho em termos do tempo médio de bloqueio

Os desempenhos médios em termos do tempo de bloqueio são apresentados na Figura 5.13. Os resultados obtidos em termos dessa métrica apresentam valores com alta variabilidade, possuindo intervalos de confiança muito largos (ver Tabela 5.1). Portanto, as análises a seguir são conduzidas observando não apenas os valores médios, mas também seus respectivos intervalos de confiança.



(a) Carga baixa ($80 msgs/s$)

(b) Carga moderada ($100 msgs/s$)

(c) Carga alta ($150 msgs/s$)

Figura 5.13. Tempos de bloqueio para diferentes tamanhos de grupos e perfis de carga

Em todos os experimentos os tempos médios de bloqueio diminuem na medida em que a carga da aplicação aumenta e aumentam na medida em que o número de membros no grupo aumenta – o que é um resultado esperado, condizente com os apresentados em Macêdo (1994).

Para as configurações de grupo com 5 membros, a variabilidade dos tempos de bloqueio impossibilitam determinar quais versões ou configurações do protocolo de comunicação em grupo obteve melhor desempenho. Verifique que, nesses casos, os intervalos de confiança, quando pares de barras dos gráficos são comparadas, possuem valores de intersecção que incluem a média dos pares observados. Considerando, por exemplo, a Figura 5.13(b), note que, para um grupo com 5 membros, $CB(ts = 20ms)$, $CB(ts = 200ms)$ e ACB possuem desempenhos em termos de

tempo de bloqueio equivalentes a $(38,44 \pm 10,93)$, $(70,31 \pm 32,84)$ e $(46,28 \pm 16,05)$, respectivamente – ver Tabela 5.1. Comparando $CB(ts = 20ms)$ com $CB(ts = 200ms)$, por exemplo, observe que o intervalo $[70,31+32,84; 70,31-32,84]$ de $CB(ts = 200ms)$ inclui a média $38,44$ de $CB(ts = 20ms)$ – o mesmo pode ser verificado nos demais, em que o resultado é avaliado considerando grupos de processos com 5 membros.

Para as configurações de grupo com mais de 5 membros, os resultados em termos dos valores médios são mais expressivos, podendo-se determinar com uma maior segurança a diferença de desempenho entre as diferentes configurações do protocolos e a diferença de desempenho destas com a abordagem autonômica. Para grupos com 15 e 25 membros, as configurações $CB(ts = 20ms)$ e $CB(ts = 200ms)$ obtiveram, respectivamente, o melhor e o pior desempenho médio em termos do tempo de bloqueio – o que é condizente com os resultados observados em termos de sobrecarga, uma vez que quanto menor a sobrecarga, maior o tempo de bloqueio. Nesses experimentos, a abordagem autonômica apresenta desempenho intermediário em termos do tempo médio de bloqueio, em conformidade com os *set-points* dinâmicos usados pela abordagem. Entretanto, a distância entre os desempenhos de ACB e de $CB(ts = 20ms)$ decai na medida em que a carga da aplicação e o tamanho do grupo aumentam.

Tomando como base o desempenho do melhor caso, i.e. $CB(ts = 20ms)$, é realizar um desempenho comparativo entre $CB(ts = 200ms)$ e ACB a partir da diferença percentual relativa – a qual determina, em termos percentuais, quanto o desempenho se distância do desempenho de $CB(ts = 20ms)$., ver Figura 5.14.

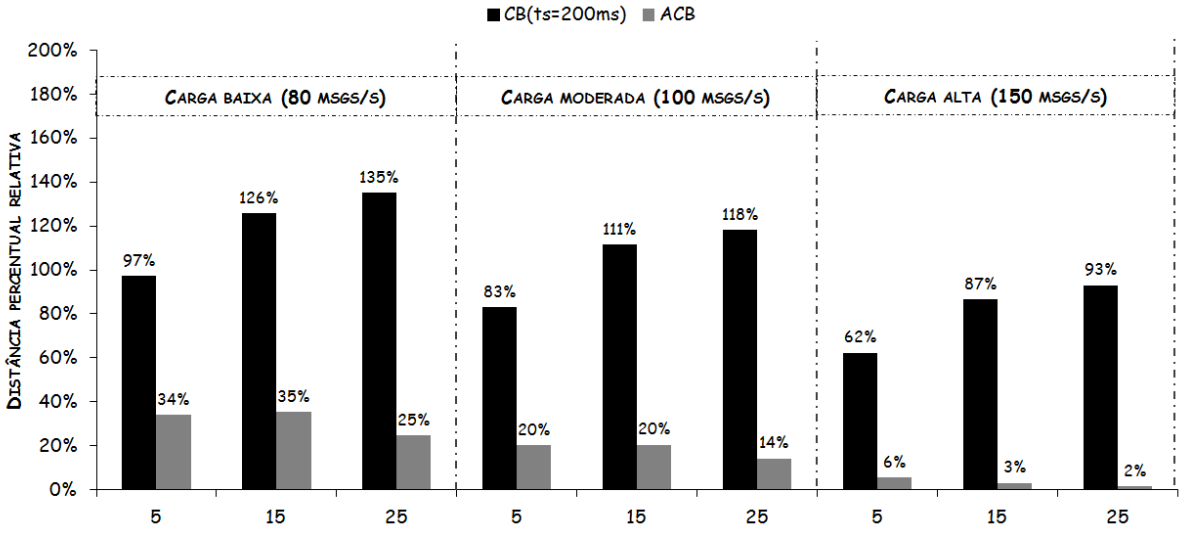


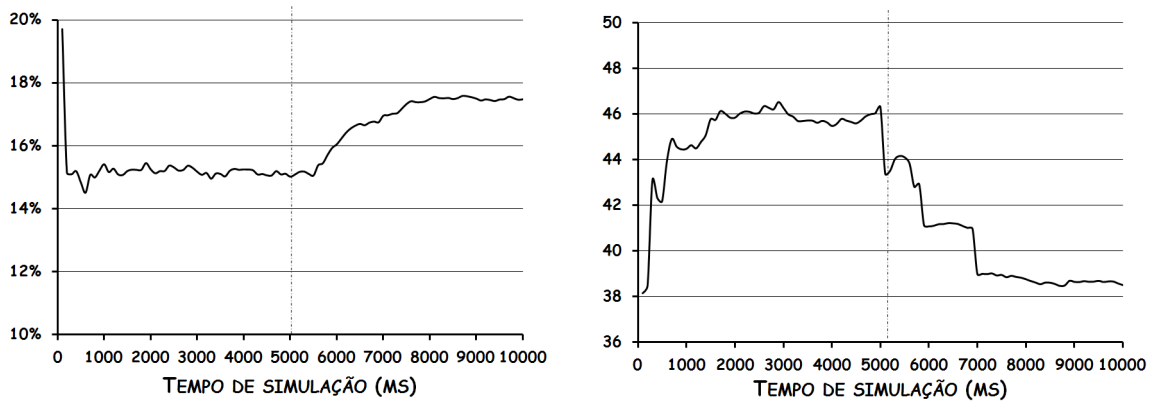
Figura 5.14. Comparativo percentual em termos do tempo de bloqueio da distância entre $CB(ts = 20ms)$ e ACB com a distância entre $CB(ts = 20ms)$ e $CB(ts = 200ms)$

Note que, conforme apresentado na Figura 5.14, os desempenhos de ACB e de $CB(ts = 200ms)$ distam 35% e 97%, respectivamente, do desempenho de $CB(ts = 20ms)$, em um cenário de melhor caso (grupo com 5 membros e perfil de carga baixa) – o que mostra que ACB possui um desempenho muito melhor que $CB(ts = 200ms)$, neste caso. Observe que, em um cenário

de pior caso (grupo com 25 membros e perfil de carga alta), os desempenhos de *ACB* e de $CB(ts = 200ms)$ distam 2% e 93%, respectivamente, do desempenho de $CB(ts = 20ms)$ – o que não apenas demonstra uma proximidade expressiva do desempenho de *ACB* em relação ao melhor desempenho, mas também mostra que *ACB* possui um desempenho expressivamente superior que $CB(ts = 200ms)$, neste caso.

5.5.2.3 Análise da variação dinâmica dos requisitos em termos de consumo de recursos

A Figura 5.15 mostra o comportamento da abordagem autonômica, com relação à sobrecarga de mensagens (Figura 5.15(a)) e do tempo de bloqueio (Figura 5.15(b)), em uma simulação envolvendo uma aplicação distribuída com perfil de carga moderada e um grupo de processos com 5 membros, em que o percentual de consumo de recursos desejado (RC_D) é dinamicamente modificado.



(a) Sobrecarga (%) (b) Tempo de bloqueio (*ms*)

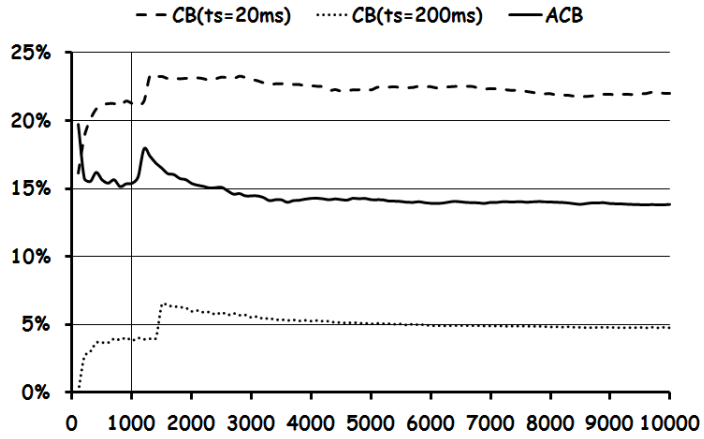
Figura 5.15. Resposta para uma mudança no set-point no instante $t = 5000ms$.

No início da simulação, a abordagem autonômica adequadamente ajusta a sobrecarga de um valor inicial para o valor desejado em termos de *set-point*, i.e. percentual de consumo de recursos igual a 25% ($RC_D = 0,25$) – o que é equivale, no melhor caso, a um valor igual ou menor a 20% em termos de sobrecarga (i.e. $ovh_{max} * RC_D = 4/5 * 0,25 = 0,2$).

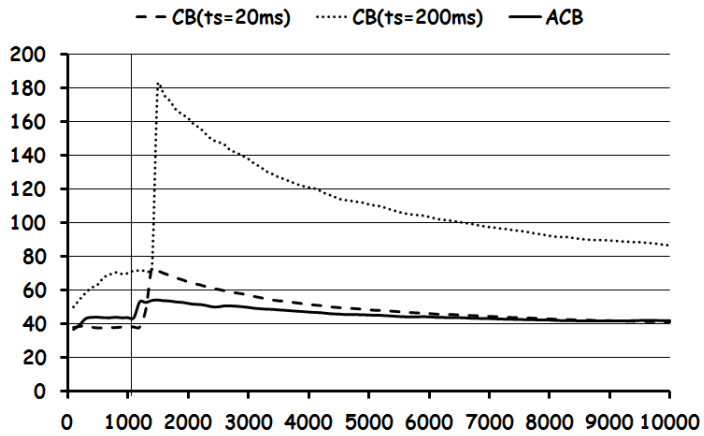
No instante $t = 5000ms$, o percentual de consumo de recurso desejado é incrementado de 25% para 35%, o que causa uma mudança no *set-point* em termos de sobrecarga de mensagens – o mesmo passa de uma valor menor ou igual a 20% para um valor menor ou igual a 28% (i.e. $ovh_{max} * RC_D = 4/5 * 0,35 = 0,28$), dependendo do comportamento dos atrasos fim-a-fim (ver Seção 5.4.2.1). Como conseqüência, observa-se um aumento na sobrecarga média de mensagens (ver Figura 5.15(a)) e uma diminuição no tempo médio de bloqueio (ver Figura 5.15(b)) – indicando que a abordagem autonômica, de forma efetiva, atende ao objetivo de projeto ajustando o modo de operação do protocolo, de acordo com mudanças dinâmicas nos requisitos desejados.

5.5.2.4 Análise em cenários com falhas de membros do grupo

Neste experimento é observado como a abordagem autonômica reage a falhas de processos durante a execução – quando comparada as diferentes configurações do protocolo básico, $CB(ts = 20ms)$ e $CB(ts = 200ms)$. Para tanto, considera-se o comportamento dos protocolos em um cenário de simulação, envolvendo uma aplicação distribuída com perfil de carga de trabalho moderada e um grupo de processos com 5 membros. Nesse cenário, é injetada uma falha em dois processos no instante $t = 1000ms$ – o que causa a execução do procedimento de mudança de visão e exclusão desses processos da visão do grupo (ver A Figura 5.16).



(a) Sobrecarga de mensagens (%)



(b) Tempo de bloqueio (*ms*)

Figura 5.16. Resposta para falha de processos no instante $t = 1000ms$

Como pode ser observado na Figura 5.16, existe um incremento no tempo de bloqueio (Figura 5.16(b)) e na sobrecarga do protocolo (Figura 5.16(a)), um pouco após a ocorrência das falhas (em $t = 1000ms$) – isto por conta da troca de mensagem necessária para que uma nova visão seja instalada. Nesse ponto, a abordagem autonômica, graças ao ajuste dinâmico do *time-silence*, produz uma convergência rápida para um novo ponto de operação em termos de sobrecarga, conforme explicado a seguir.

Quando os processo falham, mensagens *null*, necessárias para a completude dos blocos cau-

sais, não são geradas por tais processos, o que explica os picos após $t = 1000ms$, nas curvas do gráfico da Figura 5.16(b). Por outro lado, quando as falhas são detectadas, as mensagens geradas, pelo procedimento de mudança de visão, causam um aumento na sobrecarga do protocolo, o que explica os picos nas curvas do gráfico da Figura 5.16(a). Após a nova visão de grupo ser instalada, a abordagem autonômica percebe a mudança no número de membros do grupo e o controlador executa duas ações simultâneas: i) Uma vez que o número de processos diminui de 5 para 3, a sobrecarga máxima estimada pelo controlador decresce de 80% para $66,7\%$ (i.e. de $4/5$ para $2/3$) – esse fato força um decréscimo no ponto de operação em termos de sobrecarga; ii) o decréscimo no número de membros do grupo causa um decréscimo no percentual de consumo médio de recursos no ambiente – esse segundo fato força um incremento no ponto de operação em termos de sobrecarga do protocolo, por conta da existência de uma maior disponibilidade de recursos no ambiente computacional.

O algoritmo implementado pelo controlador combina esses dois efeitos e promove um ajuste moderado no ponto de operação em termos da sobrecarga de 15% para 14%, o que explica o comportamento em termos da sobrecarga média apresentada pela abordagem autonômica na Figura 5.16(a). Além disso, a redução do consumo de recursos no ambiente, por conta da diminuição do número de membros no grupo, leva o controlador também a decrementar o *time-silence*, o que conseqüentemente causa um decréscimo no tempo de bloqueio para entrega das mensagens – ver Figura 5.16(b).

Com isso, esse experimento demonstra que, quando comparada com as diferentes configurações do protocolo básico de comunicação em grupo, a abordagem autonômica consegue se adequar rapidamente às mudanças no ambiente de execução, não apenas perseguindo com efetividade os requisitos definidos pelas aplicações, mas também atendendo de forma adequada à relação de compromisso entre tempo de bloqueio e sobrecarga.

5.6 CONSIDERAÇÕES FINAIS

As abordagens de comunicação em grupo tradicionais para sistemas distribuídos não suportam a auto-configuração de seus parâmetros operacionais a partir de requisitos definidos pelo usuário. Entretanto, quando o comportamento do ambiente computacional é desconhecido e pode mudar com o tempo, ou quando os requisitos das aplicações podem mudar dinamicamente, a auto-configuração é uma questão básica que é requerida para lidar com a relação de compromisso entre os requisitos de desempenho tais como velocidade (e.g. latência de entrega das mensagens) e custo (i.e. sobrecarga do protocolo).

A habilidade de auto-configuração requer a modelagem do comportamento dinâmico do sistema distribuído, o que é um grande desafio quando o ambiente computacional pode mudar em tempo de execução. Para tratar esses desafios, este Capítulo introduziu um novo protocolo autonômico de comunicação em grupo, baseado em teoria de controle, que é capaz de, em tempo de execução, auto-configurar seus parâmetros de operação a partir dos requisitos de desempe-

nho dinamicamente especificados. A abordagem proposta integra mecanismos de detecção de defeitos, ordenação de mensagens e reconfiguração de grupo, o que facilita a construção do mecanismo de auto-configuração, o que seria mais difícil se, por exemplo, a detecção de defeitos fosse um mecanismo separado – o que é a razão da escolha do projeto com blocos causais.

Foram realizadas uma série de simulações para verificar o desempenho da proposta auto-gerenciável de comunicação em grupo, em termos da velocidade (tempo de bloqueio) e do custo (sobrecarga). Esses experimentos mostraram o desempenho do protocolo em diferentes cenários, demonstrando também como o protocolo pode ajustar dinamicamente sua operação de acordo com os *set-points* desejados (i.e. requisitos definidos em termos de consumo de recursos). Quando comparado com as versões não auto-gerenciáveis, a abordagem proposta produziu um desempenho aceitável em termos de latência de entrega e sobrecarga de mensagens em condições normais de operação e um melhor desempenho em condições de falhas ou mudanças dinâmicas dos requisitos.

Apesar de a abordagem autonômica ter sido apresentada para um protocolo de comunicação em grupo específico, seus princípios básicos e mecanismos podem ser aplicados a qualquer protocolo de comunicação em grupo existente, uma vez que as ações de controle são realizadas na freqüência de transmissão de mensagens de controle (ou monitoramento) – o que é um requisito básico para qualquer protocolo de comunicação em grupo que precise atualizar as visão do grupo (*membership*).

Como trabalhos futuros, planeja-se a avaliação da abordagem auto-gerenciável de comunicação em grupo em cenários reais, além da integração do mesmo com outros mecanismos relacionados à gestão de aplicações autonômicas em desenvolvimento no LaSiD[7], ver por exemplo Andrade e Macêdo (2009) e Sá e Macêdo (2010b).

[7]Laboratório de Sistemas Distribuídos, Departamento de Ciência da Computação, Universidade Federal da Bahia.

Capítulo 6

Este capítulo apresenta algumas considerações gerais sobre as contribuições desta Tese, fazendo um apanhado das principais contribuições e comentando as pesquisas relacionadas em andamento e futuras.

CONSIDERAÇÕES FINAIS

6.1 PRINCIPAIS CONTRIBUIÇÕES

As facilidades computacionais oriundas das novas tecnologias têm promovido o surgimento de novas modalidades de ambientes distribuídos. Estes ambientes são caracterizados pela dinamicidade em suas composições, no provisionamento de seus recursos e nas características e requisitos de suas aplicações.

Esta dinamicidade traz novos desafios à confiabilidade, um atributo essencial à grande maioria dos sistemas distribuídos modernos. Um destes desafios está na incapacidade dos mecanismos tradicionais de tolerância a falhas em atender aos requisitos de desempenho, ao mesmo tempo em que suportam a confiabilidade do sistema. Isto porque, em geral, o projeto destes mecanismos requer um conhecimento prévio das características dos ambientes e de suas aplicações, para que possam oferecer configurações adequadas ao atendimento dos requisitos especificados – isto representa um problema, uma vez que, nos ambientes distribuídos modernos, estas informações mudam dinamicamente. Neste contexto, nem mesmo os mecanismos adaptativos de tolerância falhas obtêm sucesso, pois realizam a sua configuração dinamicamente, mas confiam em comportamentos e requisitos definidos em tempo de projeto.

Para enfrentar este desafio, esta Tese introduz os mecanismos autonômicos de tolerância a falhas, baseados em teoria de controle e capazes de se auto-configurar face às mudanças dinâmicas nas características do ambiente ou nos requisitos de suas aplicações. Com o intuito de demonstrar a viabilidade destes mecanismos, foram implementados e avaliados, como estudo de caso, detectores autonômicos de defeitos e protocolos autonômicos de comunicação em grupo, dois mecanismos básicos à construção de sistemas distribuídos confiáveis.

Os detectores autonômicos de defeitos são os primeiros capazes de configurar dinamicamente seus períodos de monitoramento e timeouts de detecção, a partir de expectativas de consumo de recursos e requisitos de qualidade de serviço de detecção, i.e. tempo máximo

de detecção, intervalos mínimo entre falsas suspeitas e duração máxima das falsas suspeitas. Estes detectores foram avaliados, através de simulações, em cenários com condições de carga variadas. Além disto, os detectores autonômicos foram comparados com detectores adaptativos com períodos de monitoramento manualmente fixados. As avaliações demonstraram que os detectores autonômicos propostos possuem um desempenho superior em todos os cenários, quando comparados aos detectores adaptativos considerados. Outra contribuição importante na concepção deste mecanismo autonômico é a introdução das métricas operacionais: *atraso de interação*, *intervalo de incerteza* e *disponibilidade de detecção*. O *atraso de interação* e o *intervalo de incerteza* permitem mensurar a natureza das interações entre os processos do sistema distribuído, considerando nesta medida o impacto de perda de mensagens e o desempenho dos processos e canais, sem assumir distribuições de probabilidade ou outra estratégia definida em tempo de projeto. A *disponibilidade de detecção* permite avaliar a confiabilidade do detector sem a necessidade de um conhecimento prévio das naturezas dos atrasos dos canais de comunicação.

O protocolo autonômico de comunicação em grupo é o primeiro capaz de configurar dinamicamente seu *timeout* de bloqueio (i.e. *time-silence*) a partir requisitos de desempenho especificados pelo usuário, e.g. percentual de consumo de recursos. Este protocolo autonômico foi implementado a partir da especialização de um protocolo de comunicação em grupo baseado em blocos causais (Macêdo; Ezhilchelvan; Shrivastava, 1993). A escolha deste protocolo se deve ao fato do mesmo disponibilizar, em um *framework* integrado, mecanismos de detecção de defeitos, ordenação de mensagens e reconfiguração de grupo – o que facilitou a construção da abordagem autonômica. O protocolo autonômico foi avaliado, através de simulações, em cenários com condições de carga variadas e falhas de membros do grupo. Além disto, esta abordagem autonômica foi comparada com sua versão não autonômica, configurada com timeouts de bloqueios manualmente fixados. As avaliações demonstraram um desempenho compatível em condições normais de operação e um desempenho superior quando existem reconfigurações no grupo ou mudanças nos requisitos.

As facilidades de auto-configuração, implementadas nos mecanismos autonômicos propostos, usam estratégias que permitem que tais facilidades possam ser re-utilizadas em abordagens tradicionais que disponibilizem serviços da mesma natureza (i.e. serviços de detecção de defeitos e de comunicação em grupo). Além disto, uma vez que a modelagem do sistema distribuído considera um abordagem caixa-preta, é possível usar os mecanismos em diferentes ambientes, sem a necessidade de modificações nos algoritmos apresentados.

6.2 PESQUISAS EM ANDAMENTO

6.2.1 Prototipagem dos mecanismos propostos em ambientes reais

Atualmente, em uma de suas diversas iniciativas, o LaSiD integra uma rede nacional de cooperação acadêmica, constituída por diferentes grupos de pesquisas que colaboram no desen-

volvimento de uma plataforma federada de computação em nuvens, nomeado *JiT Clouds*[1]. O objetivo da plataforma JiT Clouds é o provisionamento dinâmico e elástico de recursos virtualizados a partir dos serviços oferecidos, de forma integrada, por diferentes ambientes de computação em nuvens, federados nas diversas instituições de pesquisa. Este tipo provisionamento requer, entre outras coisas, a negociação de acordos de níveis de serviço em tempo de execução, conforme as demandas dinâmicas das diferentes aplicações suportadas pela a plataforma.

Para tanto, dentre os diversos eixos de pesquisa necessários para o desenvolvimento da plataforma *JiT Clouds*, o LaSiD é responsável pelo suporte à confiabilidade. Uma das responsabilidades deste eixo, é o desenvolvimento de mecanismos autonômicos de tolerância a falhas capazes suportar a confiabilidade a partir de requisitos de qualidade de serviço definidos dinamicamente. Assim, atualmente estão sendo desenvolvidos protótipos dos mecanismos propostos nesta Tese para atender aos objetivos expostos nesta cooperação acadêmica. Isto representa uma oportunidade para aplicar os mecanismos autonômicos propostos nesta Tese, considerando cenários com ambientes distribuídos e aplicações reais.

6.2.2 Desenvolvimento de novos mecanismos autonômicos de tolerância a falhas

Em paralelo ao esforço de pesquisa realizado na prototipagem dos mecanismos propostos, estão sendo desenvolvidos mecanismos autonômicos de replicação (i.e. replicação ativa e failover), seguindo objetivos de projeto similares aos utilizados pelos mecanismos autonômicos de detecção de defeitos e de comunicação em grupo.

O mecanismo autonômico de replicação ativa encontra-se em fase de teste e experimentação, enquanto que o mecanismo autonômico de *failover* ainda se encontra em fase de análise. A motivação para o desenvolvimento destes mecanismos é permitir a criação de um *framework* com mecanismos autonômicos de tolerância a falhas, agregando os principais mecanismos usados para a construção de sistemas distribuídos confiáveis.

6.3 PESQUISAS FUTURAS

6.3.1 Mapeamento vertical e horizontal das demandas de qualidade de serviço

Os mecanismos autonômicos propostos têm o desempenho orientado pela qualidade de serviço demandada pelas aplicações. O mapeamento vertical das demandas de qualidade de serviço determina como os requisitos comuns às aplicações (e.g. tempo de resposta, vazão de transações por segundo, confiabilidade etc.) podem ser traduzidos em métricas de qualidade de serviço que orientam a auto-configuração dos mecanismos de tolerância a falhas propostos (e.g. tempo de detecção, intervalo entre falsas suspeitas, latência de entrega de mensagens etc.).

Outra questão é o mapeamento horizontal das demandas de qualidade de serviço das apli-

[1] *Jit in Time Cloud* (2011–2013) financiado pelo *Centro de Pesquisa e Desenvolvimento de Tecnologias Digitais para Informação e Comunicação* (CTIC) incubado na *Rede Nacional de Ensino e Pesquisa* (RNP) suportada pelo *Ministério de Ciência e Tecnologia* (MCT).

cações, isto é, um mesmo requisito pode possuir diferentes interpretações a depender do mecanismo. A confiabilidade pode, por exemplo, significar para o detector de defeitos a precisão com relação às informações reportadas sobre os estados dos processos, enquanto que para um mecanismo de replicação pode significar o número de réplicas defeituosas que são toleradas.

Estas duas questões são desafios importantes a serem tratados para garantir uma aplicação mais efetiva dos mecanismos propostos nos contextos de sistemas distribuídos considerados.

6.3.2 Formalização de um modelo de referência

A partir da consolidação do *framework* de mecanismos autonômicos de tolerância a falhas que está sendo desenvolvido e do enfrentamento das questões relacionadas ao mapeamento das demandas de qualidade de serviço, outro ponto de pesquisa futura é o estabelecimento de um modelo de referência que possa ser usado na construção e validação dos novos mecanismos de tolerância a falhas com características autonômicas.

A idéia deste modelo de referência é prover uma definição formal dos mecanismos e estabelecer o conjunto de relações que devem ser perseguidas para que tais mecanismos possam ser projetados e implementados com sucesso.

REFERÊNCIAS BIBLIOGRÁFICAS

Abdelwahed, S.; Kandasamy, N. A control-based approach to autonomic performance management in computing systems. In: Parashar, M.; Hariri, S. (Ed.). *Autonomic computing: Concepts, infrastructure and applications*. Abingdon, UK: CRC Press, 2007. p. 149–167. ISBN 0-8493-9367-1.

Afanasyev, A.; Tilley, N.; Reiher, P.; Kleinrock, L. Host-to-host congestion control for TCP. *IEEE Communications Surveys and Tutorials*, IEEE Communications Society, Washington, DC, USA, v. 12, p. 304–342, May 2010. ISSN 1553-877X. Disponível em: <http://dx.doi.org/10.1109/SURV.2010.042710.00114>. Acesso em: October, 4, 2011.

Aguirre, L. A. *Introdução à identificação de sistemas: técnicas lineares e não lineares aplicadas aos sistemas reais*. 3. ed. Belo Horizonte, MG, Brasil: Editora UFMG, 2007. 735 p. ISBN 978-85-7041-584-4.

Alima, L. O.; Haridi, S.; Roy, P. V.; Brand, P. *A survey of concepts, techniques, and systems for high availability computing*. Sweden, February 2002. 163 p. Disponível em: <https://www.sics.se/ seif/Publications/HA.pdf>. Acesso em: October, 4, 2011.

Andrade, S. S.; Macêdo, R. J. A. A non-intrusive component-based approach for deploying unanticipated self-management behaviour. In: *Proceedings of IEEE ICSE 2009 Workshop Software Engineering for Adaptive and Self-Managing Systems*. Vancouver, CANADA: [s.n.], 2009. (SEMS'2009), p. 152–161. ISBN 978-1-4244-3724-5. Disponível em: <http://dx.doi.org/10.1109/SEAMS.2009.5069084>. Acesso em: October, 4, 2011.

Ang, K. H.; Chong, G.; Li, Y. Pid control system analysis, design, and technology. *IEEE Transactions on Control Systems Technology (TCST'2005)*, IEEE Control Systems Society, v. 13, n. 4, p. 559–576, July 2005. ISSN 1063-6536. Disponível em: <http://dx.doi.org/10.1109/TCST.2005.847331>. Acesso em: October, 4, 2011.

Armstrong, D.; Carter, S.; Frazier, G.; Frazier, T. Autonomic defense: Thwarting automated attacks via real-time feedback control. *Complexity Journal – Special issue: Resilient and adaptive defense of computing networks*, John Wiley & Sons, Inc., New York, NY, USA, v. 9, p. 41–48, November 2003. ISSN 1076-2787. Acesso em: October, 4, 2011.

Astrom, K. J.; Wittenmark, B. *Adaptive Control*. 2nd. ed. Mineola, New York, USA: Dover Publications, 1995. 574 p. ISBN 0-486-46278-1.

Avizienis, A.; Laprie, J.-C.; Randell, B.; Landwehr, C. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing (TDSC2004)*,

IEEE Computer Society, v. 1, n. 1, p. 11–33, January–March 2004. ISSN 1545-5971. Disponível em: <http://dx.doi.org/10.1109/TDSC.2004.2>. Acesso em: October, 4, 2011.

Baker, M.; Buyya, R.; Laforenza, D. Grids and grid technologies for wide-area distributed computing. *Software – Practice and Experience*, John Wiley and Sons, Inc., New York, NY, USA, v. 32, p. 1437–1466, December 2002. ISSN 0038-0644. Disponível em: <http://dx.doi.org/10.1002/spe.488>. Acesso em: October, 4, 2011.

Barroso, L. A.; Dean, J.; Hölzle, U. Web search for a planet: The google cluster architecture. *IEEE Micro*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 23, p. 22–28, March 2003. ISSN 0272-1732. Disponível em: <http://dx.doi.org/10.1109/MM.2003.1196112>. Acesso em: October, 4, 2011.

Bertier, M.; Marin, O.; Sens, P. Implementation and performance evaluation of an adaptable failure detector. In: *Proceedings of the International Conference on Dependable Systems and Networks*. Washington, DC, USA: IEEE Computer Society, 2002. (DSN'2002), p. 354–363. ISBN 0-7695-1597-5. Disponível em: <http://dx.doi.org/10.1109/DSN.2002.1028920>. Acesso em: October, 4, 2011.

Bertier, M.; Marin, O.; Sens, P. Performance analysis of a hierarchical failure detector. In: *Proceedings of the 2003 International Conference on Dependable Systems and Networks*. San-Francisco,USA: IEEE Computer Society, 2003. (DSN'2003), p. 635–644. ISBN 0-7695-1952-0. Disponível em: <http://dx.doi.org/10.1109/DSN.2003.1209973>. Acesso em: October, 4, 2011.

Bezerra, R. M. S.; Martins, J. S. B. A policy-based autonomic model suitable for quality of service management. *Journal of Networks*, Academy Publisher, v. 4, n. 6, p. 495–504, August 2009. ISSN 1796-2056. Disponível em: <http://dx.doi.org/10.4304/jnw.4.6.495-504>. Acesso em: October, 4, 2011.

Birman, K. P. The process group approach to reliable distributed computing. *Communications of the ACM*, ACM, New York, NY, USA, v. 36, p. 37–53, December 1993. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/163298.163303>. Acesso em: October, 4, 2011.

Campbell, R. H.; Randell, B. Error recovery in asynchronous systems. *IEEE Transactions on Software Engineering*, IEEE Press, Piscataway, NJ, USA, v. 12, p. 811–826, August 1986. ISSN 0098-5589. Disponível em: <http://dl.acm.org/citation.cfm?id=6223.6276>. Acesso em: October, 4, 2011.

Carneiro, I. S. C.; Sá, A. S. de; Barreto, L. P. Aplicação e análise de teoria de controle realimentado no tratamento de faltas de páginas em sistemas de gerenciamento de memória. In: *VI Workshop de Sistemas Operacionais (WSO2009*. Bento Gonçalves, RS, Brasil: SBC, 2009. (WSO'2009), p. 2367–2379.

Chandra, T. D.; Hadzilacos, V.; Toueg, S.; Charron-bost, B. On the impossibility of group membership. In: *Proceedings of the Fifteenth Annual ACM Symposium on Principles of distributed computing*. New York, NY, USA: ACM, 1996. (PODC'1996), p. 322–330. ISBN 0-89791-800-2. Disponível em: <http://doi.acm.org/10.1145/248052.248120>. Acesso em: October, 4, 2011.

Chandra, T. D.; Toueg, S. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, ACM, New York, NY, USA, v. 43, n. 2, p. 225–267, March 1996. Disponível em: <http://doi.acm.org/10.1145/226643.226647>. Acesso em: October, 4, 2011.

Chen, W.; Toueg, S.; Aguilera, M. K. On the quality of service of failure detectores. *IEEE Transactions on Computers*, IEEE Computer Society, Los Alamitos, CA, USA, v. 51, n. 2, p. 561–580, May 2002. ISSN 0018-9340. Disponível em: <http://dx.doi.org/10.1109/TC.2002.1004595>. Acesso em: October, 4, 2011.

Chockler, G. V.; Huleihel, N.; Dolev, D. An adaptive totally ordered multicast protocol that tolerates partitions. In: *Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*. New York, NY, USA: ACM, 1998. (PODC'1998), p. 237–246. ISBN 0-89791-977-7. Disponível em: <http://doi.acm.org/10.1145/277697.277741>. Acesso em: October, 4, 2011.

Chockler, G. V.; Keidar, I.; Vitenberg, R. Group communication specifications: a comprehensive study. *ACM Computing Surveys*, ACM, New York, NY, USA, v. 33, p. 427–469, December 2001. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/503112.503113>. Acesso em: October, 4, 2011.

Claudel, B.; Palma, N. D.; Lachaize, R.; Hagimont, D. Self-protection for distributed component-based applications. In: Datta, A.; Gradinariu, M. (Ed.). *Stabilization, Safety, and Security of Distributed Systems (SSS'2006)*. Springer Berlin / Heidelberg, 2006, (Lecture Notes in Computer Science, v. 4280). p. 184–198. ISBN 978-3-540-49018-0. Disponível em: <http://dx.doi.org/10.1007/978-3-540-49823-0_13>. Acesso em: October, 4, 2011.

Colom, P. M. *Analysis and design of real-time control systems with varying control timing constraints*. 219 p. Tese (PHD Thesis) — Département D'Enginyeria De Sistemes, Automàtica I Informàtica Industrial, Barcelona, June 2002. Disponível em: <http://www.tdx.cat/handle/10803/6182>. Acesso em: October, 4, 2011.

Coulouris, G.; Dollimore, J.; Kindberg, T. *Distributed systems: Concepts and design*. 4th. ed. USA: Addison-Wesley, 2005. 944 p. ISBN 0-321-26354-5.

Cristian, F. Understanding fault-tolerant distributed systems. *Communications of the ACM*, ACM, New York, NY, USA, v. 34, n. 2, p. 56–78, February 1991. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/102792.102801>. Acesso em: October, 4, 2011.

Cristian, F. Synchronous and asynchronous group communication. *Communications of the ACM*, ACM, New York, NY, USA, v. 39, p. 88–97, April 1996. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/227210.227231>. Acesso em: October, 4, 2011.

Cristian, F.; Fetzer, C. The timed asynchronous distributed system model. *IEEE Transactions on Parallel Distributed Systems*, IEEE Computer Society, Piscataway, NJ, USA, v. 10, p. 642–657, June 1999. ISSN 1045-9219. Disponível em: <http://dx.doi.org/10.1109/71.774912>. Acesso em: October, 4, 2011.

Dai, S.; Wang, M. *Reliability analysis in engineering applications*. New York, NY, USA: Van Nostrand Reinhold, 1992. 448 p. ISBN 978-04-4200-842-0.

Défago, X.; Schiper, A.; Urbán, P. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys*, ACM, New York, NY, USA, v. 36, p. 372–421, December 2004. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/1041680.1041682>. Acesso em: October, 4, 2011.

Deswarte, Y.; Kanoun, K.; Laprie, J. Diversity against accidental and deliberate faults. In: *Proceedings of the Conference on Computer Security, Dependability, and Assurance: From Needs to Solutions*. Washington, DC, USA: IEEE Computer Society, 1998. (CSDA'1998), p. 171–181. ISBN 0-7695-0337-3. Disponível em: <http://dx.doi.org/10.1109/CSDA.1998.798364>. Acesso em: October, 4, 2011.

Diao, Y.; Hellerstein, J.; Parekh, S. Optimizing quality of service using fuzzy control. In: Feridun, M.; Kropf, P.; Babin, G. (Ed.). *Management Technologies for E-Commerce and E-Business Applications*. Springer Berlin / Heidelberg, 2002, (Lecture Notes in Computer Science, v. 2506). p. 42–53. ISBN 978-3-540-00080-8. Disponível em: <http://dx.doi.org/10.1007/3-540-36110-3_7>. Acesso em: October, 4, 2011.

Diao, Y. et al. A control theory foundation for self-managing computing systems. *Journal on Selected Areas in Communications*, IEEE, p. 2213–2222, December 2005. ISSN 0733-8716. Disponível em: <http://dx.doi.org/10.1109/JSAC.2005.857206>. Acesso em: October, 4, 2011.

Dixit, M.; Casimiro, A. Adaptare-fd: A dependability-oriented adaptive failure detector. In: *Proceedings of the 29th IEEE Symposium on Reliable Distributed Systems*. New Delhi, India: IEEE Computer Society, 2010. (SRDS'2010), p. 141–147. ISBN 978-0-7695-4250-8. ISSN 1060-9857. Disponível em: <http://dx.doi.org/10.1109/SRDS.2010.24>. Acesso em: October, 4, 2011.

Dolev, D.; Dwork, C.; Stockmeyer, L. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, ACM, New York, NY, USA, v. 34, p. 77–97, January 1987. ISSN 0004-5411. Disponível em: <http://doi.acm.org/10.1145/7531.7533>. Acesso em: October, 4, 2011.

Doudou, A.; Garbinato, B.; Guerraoui, R. Encapsulating failure detection: From crash to byzantine failures. In: *Proceedings of the 7th Ada-Europe International Conference on Reliable Software Technologies*. London, UK, UK: Springer-Verlag, 2002. (Ada-Europe'2002), p. 24–50. ISBN 3-540-43784-3. Disponível em: <http://dx.doi.org/10.1007/3-540-48046-3_3>. Acesso em: October, 4, 2011.

Dwork, C.; Lynch, N.; Stockmeyer, L. Consensus in the presence of partial synchrony. *Journal of ACM*, ACM, New York, NY, USA, v. 35, p. 288–323, April 1988. ISSN 0004-5411. Disponível em: <http://doi.acm.org/10.1145/42282.42283>. Acesso em: October, 4, 2011.

Ezhilchelvan, P. D.; Macêdo, R. J. A.; Shrivastava, K. S. Newtop: a fault-tolerant group communication protocol. In: *Proceedings of the 15th International Conference on Distributed Computing Systems*. Washington, DC, USA: IEEE Computer Society, 1995. (ICDCS'1995), p. 296–306. ISBN 0-8186-7025-8. Disponível em: <http://dx.doi.org/10.1109/ICDCS.1995.500032>. Acesso em: October, 4, 2011.

Falai, L.; Bondavalli, A. Experimental evaluation of the qos failure detectors on wide area network. In: *Proceedings of the 2005 International Conference On Dependable Systems And Networks*. Yokohama, Japan: IEEE Computer Society, 2005. (DSN'2005), p. 624–633. ISBN 0-7695-2282-3. Disponível em: <http://dx.doi.org/10.1109/DSN.2005.47>. Acesso em: October, 4, 2011.

Felber, P. *The corba object group service : A service approach to object groups in CORBA*. 179 p. Tese (Doutorado) — Département D'Informatique, École Polytechnique Fédérale De Lausanne, 1998. Disponível em: <http://biblion.epfl.ch/EPFL/theses/1998/1867/EPFL_TH1867.pdf>. Acesso em: October, 4, 2011.

Fischer, M. J.; Lynch, N. A.; Paterson, M. S. Impossibility of distributed consensus with one faulty process. *Journal of ACM*, ACM, New York, NY, USA, v. 32, p. 374–382, April 1985. ISSN 0004-5411. Disponível em: <http://doi.acm.org/10.1145/3149.214121>. Acesso em: October, 4, 2011.

Foletto, T.; Moraes, V.; Moreno, U.; Sá, A. S. de; Macêdo, R. J. A. Avaliação sobre a inclusão de estimadores baseados em filtragem de kalman em sistemas de controle via rede. In: *XVIII Congresso Brasileiro de Automática*. Bonito, MS, Brasil: SBA, 2010. (CBA'2010), p. 1–7. ISBN 978-8-5615-0702-2.

Freitas, A. E. S.; Macêdo, R. J. A. A simulation tool for dynamic and hybrid distributed systems (in portuguese: *Um Ambiente para Testes e Simulações de Protocolos Confiáveis em Sistemas Distribuídos Híbridos e Dinâmicos*). In: *Proceedings of the XXVII Brazilian Symposium on Computer Networks and Distributed Systems*. Recife, PE, Brazil: SBC, 2009. (SBRC'2009), p. 826–839. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/sbrc/2009/058.pdf>. Acesso em: October, 4, 2011.

Ganek, A. Overview of autonomic computing: Origins, evolution, direction. In: Parashar, M.; Hariri, S. (Ed.). *Autonomic computing: Concepts, infrastructure and applications*. Abingdon, UK: CRC Press, 2007. p. 3–18. ISBN 0-8493-9367-1.

Garlan, D.; Shaw, M. An introduction to software architecture. *Advances in software engineering and knowledge engineering*, Singapore, v. 1, p. 1–40, 1993.

Gärtner, F. C. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Computing Surveys*, ACM, New York, NY, USA, v. 31, p. 1–26, March 1999. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/311531.311532>. Acesso em: October, 4, 2011.

Goodwin, G. C.; G., S. F.; Salgado, M. E. *Control system design*. Upper Saddle River, NJ, USA: Prentice-Hall, 2001. 908 p. ISBN 0-13-958653-9.

Gorender, S.; Macêdo, R. J. A. Um modelo para tolerância a falhas em sistemas distribuídos com qos. In: *Anais do 20o Simpósio Brasileiro de Redes de Computadores*. Búzios, RJ, Brazil: SBRC / SBC, 2002. (SBRC'2002), p. 277–292. ISBN 85-88442-21-3. Disponível em: <http://www.sbrc2007.ufpa.br/anais/2002/18.pdf>. Acesso em: 4 de outubro de 2011.

Gorender, S.; Macêdo, R. J. A. Consenso com recuperação no modelo partitioned synchronous. In: *Anáis do XI Workshop de Testes e Tolerância a Falhas*. Gramado, Brazil: SBC, 2010. (WTF'2010), p. 3–16. Disponível em: <http://sbrc2010.inf.ufrgs.br/anais/data/pdf/wtf/st01_01_wtf.pdf>. Acesso em: 4 de outubro de 2011.

Gorender, S.; Macêdo, R. J. A. Um algoritmo eficiente de consenso distribuído para o modelo síncrono particionado. In: *Anais do XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Campo Grande, MS, Brazil: SBRC / SBC, 2011. (SBRC'2011), p. 587–600. Disponível em: <http://sbrc2011.facom.ufms.br/files/main/ST12_4.pdf>. Acesso em: 4 de outubro de 2011.

Gorender, S.; Macêdo, R. J. A.; Raynal, M. An adaptive programming model for fault-tolerant distributed computing (with an application to consensus). *IEEE Transactions on Dependable Secure Computing*,

IEEE Computer Society Press, Los Alamitos, CA, USA, v. 4, p. 18–31, January 2007. ISSN 1545-5971. Disponível em: <http://dx.doi.org/10.1109/TDSC.2007.3>. Acesso em: October, 4, 2011.

Gorender, S.; Mâcedo, R. J.; Raynal, M. A hybrid and adaptative model for fault-tolerant distributed computing. In: *Proceedings of the International Conference on Dependable Systems and Networks*. Yokohama, Japan: IEEE Computer Society, 2005. (DSN'2005), p. 412–421. ISBN 0-7695-2282-3. Disponível em: <http://dx.doi.org/10.1109/DSN.2005.8>. Acesso em: October, 4, 2011.

Guerraoui, R. et al. Consensus in asynchronous distributed systems: A concise guided tour. In: Krakowiak, S.; Shrivastava, S. (Ed.). *Advances in Distributed Systems*. Springer Berlin / Heidelberg, 2000, (Lecture Notes in Computer Science, v. 1752). p. 33–47. ISBN 978-3-540-67196-1. Disponível em: <http://dx.doi.org/10.1007/3-540-46475-1_2>. Acesso em: October, 4, 2011.

Hadzilacos, V.; Toueg, S. Fault-tolerant broadcasts and related problems. In: ____. *Distributed systems*. 2nd. ed. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1993. cap. 5, p. 97–145. ISBN 0-201-62427-3.

Hegering, H. G.; Abeck, S.; Neumair, B. *Integrated management of networked systems: concepts, architectures, and their operational application*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998. 669 p. ISBN 1-55860-571-1.

Hellerstein, J. L.; Diao, Y.; Parekh, S.; Tilbury, D. M. *Feedback control of computing systems*. Canada: Wiley-Interscience, 2004. 429 p. ISBN 0-471-26637-X.

Henriksson, D.; Cervin, A. *Truetime 1.5 - Reference manual*. Department of Automatic Control, Lund University, Sweden, January 2007. Disponível em: <http://www3.control.lth.se/truetime/>.

Hermant, J. F.; Lann, G. L. Fast asynchronous uniform consensus in real-time distributed systems. *IEEE Transactions on Computers*, IEEE Computer Society, Washington, DC, USA, v. 51, p. 931–944, August 2002. ISSN 0018-9340. Disponível em: <http://dx.doi.org/10.1109/TC.2002.1024740>. Acesso em: October, 4, 2011.

Horn, P. Autonomic computing: Ibm's perspective on the state of information technology. *Computing Systems*, IBM, v. 15, n. January, p. 1–40, 2001.

Hsu, H. P. *Teoria e problemas de sinais e sistemas*. Porto Alegre, RS, Brasil: Bookman, 2004. 432 p. (Coleção Schaum). ISBN 85-363-0360-3.

Huebscher, M. C.; Mccann, J. A. A survey of autonomic computing: Degrees, models, and applications. *ACM Computing Surveys*, ACM, New York, NY, USA, v. 40, p. 7:1–7:28, August 2008. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/1380584.1380585>. Acesso em: October, 4, 2011.

Jacob, B.; Lanyon-hang, R.; Nadgir, D. K.; Yassin, A. F. *A pratical guide to the IBM autonomic computing toolkit*. 1. ed. USA: IBM, 2004. 274 p. (IBM Redbooks). ISBN 073849805X. Disponível em: <http://www.redbooks.ibm.com/abstracts/sg246635.html?Open>. Acesso em: October, 4, 2011.

Jacobson, V. Congestion avoidance and control. In: *Symposium proceedings on Communications architectures and protocols*. New York, NY, USA: ACM, 1988. (SIGCOMM '88), p. 314–329. ISBN 0-89791-279-9. Disponível em: <http://doi.acm.org/10.1145/52324.52356>. Acesso em: October, 4, 2011.

Jain, R. *The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation, and modeling*. USA: Addison-Wesley, 1991. 685 p. ISBN 0-471-50336-3.

Jain, R.; Routhier, S. Packet trains : Measurements and a new model for computer network traffic. *IEEE Journal on Selected Areas in Communications*, IEEE, v. 4, p. 986–995, September 1986. ISSN 0733-8716. Disponível em: <http://dx.doi.org/10.1109/JSAC.1986.1146410>. Acesso em: October, 4, 2011.

Jalote, P. *Fault tolerance in distributed systems*. New Jersey: Prentice Hall, 1994.

Karmakar, S.; Gupta, A. Adaptive broadcast by distributed protocol switching. In: *Proceedings of the 2007 ACM symposium on Applied computing*. New York, NY, USA: ACM, 2007. (SAC'2007), p. 588–589. ISBN 1-59593-480-4. Disponível em: <http://doi.acm.org/10.1145/1244002.1244136>. Acesso em: October, 4, 2011.

Kephart, J. O. Research challenges of autonomic computing. In: *Proceedings of the 27th international conference on Software engineering*. New York, NY, USA: ACM, 2005. (ICSE'2005), p. 15–22. ISBN 1-58113-963-2. Disponível em: <http://doi.acm.org/10.1145/1062455.1062464>. Acesso em: October, 4, 2011.

Keshav, S. A control-theoretic approach to flow control. In: *Proceedings of the conference on Communications architecture and protocols*. New York, NY, USA: ACM, 1991. (SIGCOMM'1991), p. 3–15. ISBN 0-89791-444-9. Disponível em: <http://doi.acm.org/10.1145/115992.115995>. Acesso em: October, 4, 2011.

Kopetz, H. *Real-time systems: Design principles for distributed embebed applications*. 1st. ed. Norwell, Massachusetts, USA: Kluwer Academic Publishers, 1997. 338 p. ISBN 0-792-39894-7.

Lamport, L. Ti clocks, and the ordering of events in a distributed system. *Communications of the ACM*, ACM, New York, NY, USA, v. 21, p. 558–565, July 1978. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/359545.359563>. Acesso em: October, 4, 2011.

Lamport, L.; Lynch, N. *Chapter on distributed computing: Methods and models*. Cambridge, MA, USA, February 1989. 71 p. Disponível em: <http://research.microsoft.com/users/lamport/pubs/lamport-chapter.pdf>. Acesso em: October, 4, 2011.

Lamport, L.; Shostak, R.; Pease, M. The byzantine generals problem. *ACM Transaction on Programming Languages Systems*, ACM, New York, NY, USA, v. 4, p. 382–401, July 1982. ISSN 0164-0925. Disponível em: <http://doi.acm.org/10.1145/357172.357176>. Acesso em: October, 4, 2011.

Lampson, B. W. Protection. *ACM SIGOPS Operating Systems Review*, ACM, New York, NY, USA, v. 8, p. 18–24, January 1974. ISSN 0163-5980. Disponível em: <http://doi.acm.org/10.1145/775265.775268>. Acesso em: October, 4, 2011.

Lann, G. L.; Schmid, U. *How to maximize computing systems coverage*. Vienna, Austria, April 2003. 26 p. Disponível em: <http://ti.tuwien.ac.at/ecs/research/projects/theta/papers>.

Leroy, X. Java bytecode verification: Algorithms and formalizations. *Journal of Automated Reasoning*, Springer Netherlands, v. 30, p. 235–269, 2003. ISSN 0168-7433. Disponível em: <http://dx.doi.org/10.1023/A:1025055424017>. Acesso em: October, 4, 2011.

Litiu, R.; Prakash, A. Adaptive group communication services for groupware systems. In: *Proceeding of Second International Enterprise Distributed Object Computing Workshop (EDOC'1998)*. La Jolla, CA , USA: IEEE Computer Society, 1998. p. 218–229. ISBN 0-7803-5158-4. Disponível em: <http://dx.doi.org/10.1109/EDOC.1998.723257>. Acesso em: October, 4, 2011.

Little, J. D. C. A proof for the queuing formula: $l = \lambda w$. *Operations Research Journal*, v. 9, n. 3, p. 383–387, May–June 1961. ISSN 0030364X. Disponível em: <http://www.jstor.org/stable/167570>. Acesso em: October, 4, 2011.

Liu, X.; Renesse, R. van; Bickford, M.; Kreitz, C.; Constable, R. Protocol switching: Exploiting meta-properties. In: *Proceedings of the International Conference on Distributed Computing Systems Workshops*. Los Alamitos, CA, USA: IEEE Computer Society, 2001. (CDCS'2001), p. 37–42. ISBN 0-7695-1080-9. Disponível em: <http://dx.doi.org/10.1109/CDCS.2001.918684>. Acesso em: October, 4, 2011.

Lynch, N. A. *Distributed algorithms*. San Francisco, California: Morgan Kaufmann, 1996. 904 p. ISBN 1-55-860348-4.

Macêdo, R. J. A. *Fault-tolerant group communication protocols for asynchronous systems*. 166 p. Tese (PHD thesis) — University of Newcastle upon Tyne, 1994.

Macêdo, R. J. A. Failure detection in asynchronous distributed systems. In: *Proceedings of the 2nd Workshop on Tests and Fault-Tolerance*. Curitiba, PR, Brazil: SBC, 2000. (WTF '2000), p. 76–81. Disponível em: <http://www.lasid.ufba.br/publicacoes/artigos/2wtf.ps>. Acesso em: October, 4, 2011.

Macêdo, R. J. A. An integrated group communication infrastructure for hybrid real-time distributed systems. In: *9th Workshop on Real-Time Systems*. Belém, PA, Brazil: SBC, 2007. (WTR'2007), p. 81–88.

Macêdo, R. J. A. *Adaptive and Dependable Group Communication*. Salvador, Ba, Brazil, January 2008. 9 p. Disponível em: <http://www.lasid.ufba.br/publicacoes/artigos/TimedCBResumidav5.pdf>.

Macêdo, R. J. A. *Approaches for adaptive and dependable distributed systems*. IFIP Workshop on Dependability of Large-Scale and Dynamic Systems (IFIP'2008). Natal, RN, Brazil: [s.n.], February 2008. eletronic. Disponível em: <http://www2.laas.fr/IFIPWG/Workshops&Meetings/53/workshop/11.Macedo.pdf>. Acesso em: October, 4, 2011.

Macêdo, R. J. A.; Ezhilchelvan, P. D.; Shrivastava, K. S. Newtop: a total order multicast protocol using causal blocks. In: *First Open Broadcast Workshop. Also in First Year Report - Fundamental Concepts, vol. I, chap. II, Broadcast Espirit Basic Research Project 6360.*

Newcastle upon Tyne, U. K.: University of Newcastle, 1993. (OBW'1993). Disponível em: <http://www.lasid.ufba.br/publicacoes/artigos/mes93b.ps>. Acesso em: October, 4, 2011.

Macêdo, R. J. A.; Freitas, A. E. S. A generic group communication approach for hybrid distributed systems. In: *Proceedings of the 9th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems*. Berlin, Heidelberg: Springer-Verlag, 2009. (DAIS'2009), p. 102–115. ISBN 978-3-642-02163-3. Disponível em: <http://dx.doi.org/10.1007/978-3-642-02164-0_8>. Acesso em: October, 4, 2011.

Macêdo, R. J. A.; Freitas, A. E. S. Group communication for self-aware distributed systems. In: *Proceedings of the XXVIII Brazilian Symposium on Computer Networks and Distributed Systems*. Gramado, RS, Brazil: SBC, 2010. (SBRC'2010), p. 915–928. Disponível em: <http://sbrc2010.inf.ufrgs.br/anais/data/pdf/trilha/st19_02_artigo.pdf>. Acesso em: October, 4, 2011.

Macêdo, R. J. A.; Freitas, A. E. S.; Sá, A. S. de. A self-manageable group communication protocol for partially synchronous distributed systems. In: *Proceeding of Fifth Latin-Amarican Symposium on Dependable Computing*. São José dos Campos, SP, Brazil: IEEE Computer Society Press, 2011. (LADC'2011), p. 146–155. ISBN 978-0-7695-4320-8.

Macêdo, R. J. A.; Gorender, S. Detectores perfeitos em sistemas distribuídos não síncronos. In: *Anáis do IX Workshop de Testes e Tolerância a Falhas*. Rio de Janeiro, RJ, Brazil: SBC, 2008. (WTF'2008), p. 127–140. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/wtf/2008/009.pdf>. Acesso em: 4 de outubro de 2011.

Macêdo, R. J. A.; Gorender, S. Perfect failure detection in the partitioned synchronous distributed system model. In: *Proceedings of the 2009 International Conference on Availability, Reliability and Security*. Washington, DC, USA: IEEE Computer Society, 2009. (ARES'2009), p. 273–280. ISBN 978-1-4244-3572-2. Disponível em: <http://dx.doi.org/10.1109/ARES.2009.165>. Acesso em: October, 4, 2011.

Macêdo, R. J. A.; Lima, F. Improving the quality of service of failure detectors with SNMP and artificial neural networks. In: *Anais do 22o Simpósio Brasileiro de Redes de Computadores (SBRC'2004)*. Gramado, RS, Brazil: SBRC / SBC, 2004. (SBRC'2004), p. 583–586. ISBN 85-88442-81-7. Disponível em: <http://www.sbrc2007.ufpa.br/anais/2004/2164.pdf>. Acesso em: 4 de outubro de 2011.

Manoel, E.; Nielsen, M. J.; Salahshour, A.; Sampath, S.; Sudarshanan, S. *Problem determination using self-mananging autonomic technology*. 1. ed. USA: IBM, 2005. 406 p. (IBM Redbooks). ISBN 073849111X. Disponível em: <http://www.redbooks.ibm.com/abstracts/sg246665.html?Open>. Acesso em: October, 4, 2011.

Menascé, D. A.; Bennani, M. N. Autonomic virtualized environments. In: *Proceedings of the International Conference on Autonomic and Autonomous Systems*. Washington, DC, USA: IEEE Computer Society, 2006. (ICAS'2006), p. 28–37. ISBN 0-7695-2653-5. Disponível em: <http://dx.doi.org/10.1109/ICAS.2006.13>. Acesso em: October, 4, 2011.

Menascé, D. A.; Barbará, D.; Dodge, R. Preserving qos of e-commerce sites through self-tuning: a performance model approach. In: *Proceedings of the 3rd ACM conference on Electronic Commerce*.

New York, NY, USA: ACM, 2001. (EC'2001), p. 224–234. ISBN 1-58113-387-1. Disponível em: <http://doi.acm.org/10.1145/501158.501186>. Acesso em: October, 4, 2011.

Microsoft. *Microsoft dynamic systems initiative overview*. [S.l.], March 2005. 26 p. Disponível em: <http://www.microsoft.com/dsi>. Acesso em: January, 28, 2007.

Mills, K.; Rose, S.; Quirolgico, S.; Britton, M.; Tan, C. An autonomic failure-detection algorithm. In: *Proceedings of the 4th International Workshop on Software and Performance*. New York, NY, USA: ACM, 2004. (WOSP'2004), p. 79–83. ISBN 1-58113-673-0. Disponível em: <http://doi.acm.org/10.1145/974044.974056>. Acesso em: October, 4, 2011.

Milojicic, D. S. et al. *Peer-to-Peer computing*. Hewlett-Packard, HP Labs, July 2002. 51 p. Disponível em: <http://www.hpl.hp.com/techreports/2002/HPL-2002-57R1.html>.

Müller, H. A.; O'brien, L.; Klen, M.; Wood, B. *Autonomic Computing*. Carnegie, USA, April 2006. 61 p.

Mocito, J.; Rodrigues, L. Run-time switching between total order algorithms. In: Nagel, W.; Walter, W.; Lehner, W. (Ed.). *Proceedings of the Euro-Par 2006 Parallel Processing*. Springer Berlin / Heidelberg, 2006, (Lecture Notes in Computer Science, v. 4128). p. 582–591. ISBN 978-3-540-37783-2. Disponível em: <http://dx.doi.org/10.1007/11823285_60>. Acesso em: October, 4, 2011.

Mont, M. C.; Bramhall, P.; Pato, J. *On adaptive identity management: The Next generation of identity management technologies*. Bristol, UK, July 2003. 19 p.

Morrison, F. *The art of modeling dynamic systems: Forecasting for chaos, randomness and determinism*. Mineola, New York, USA: Dover Publications, 2008. 412 p. ISBN 0-486-46295-1.

Nunes, R. C.; Jansch-pôrto, I. Qos of timeout-based self-tuned failure detectors: The effects of the communication delay predictor and the safety margin. In: *Proceedings of the 2004 International Conference On Dependable Systems And Networks*. Florence, Italy: IEEE Computer Society, 2004. (DSN'2004), p. 753–761. ISBN 0-7695-2052-9. Disponível em: <http://dx.doi.org/10.1109/DSN.2004.1311946>. Acesso em: October, 4, 2011.

Ogata, K. *Discrete-time control systems*. 2nd. ed. Upper Saddle River, NJ 07458, USA: Prentice-Hall, 1995. 745 p. ISBN 0-13-034281-5.

Ogata, K. *Modern Control Engineering*. 5th. ed. Upper Saddle River, NJ 07458, USA: Prentice-Hall, 2009. 912 p. ISBN 0-13-615673-8.

Parashar, M.; Hariri, S. Autonomic computing: An overview. In: Banâtre, J. P.; Fradet, P.; Giavitto, J. L.; Michel, O. (Ed.). *Unconventional Programming Paradigms*. Springer Berlin / Heidelberg, 2005, (Lecture Notes in Computer Science, v. 3566). p. 97–97. ISBN 978-3-540-27884-9. Disponível em: <http://dx.doi.org/10.1007/11527800_20>. Acesso em: October, 4, 2011.

Pasin, M.; Fontaine, S.; Bouchenak, S. Failure detection in large scale systems: a survey. In: *Proceedings of the 2008 IEEE Network Operations and Management Symposium Workshops*. Salvador, BA, Brazil: [s.n.], 2008. (NOMS'2008), p. 165–168. ISBN 978-1-4244-2067-4. Disponível em: <http://dx.doi.org/10.1109/NOMSW.2007.28>. Acesso em: October, 4, 2011.

Randell, B.; Xu, J. The evolution of the recovery block concept. In: Lyu, M. R. (Ed.). *In Software Fault Tolerance*. [S.l.]: John Wiley & Sons Ltd, 1995. (Software Fault Tolerance), p. 1–22.

Raynal, M. A short introduction to failure detectors for asynchronous distributed systems. *ACM Special Interest Group on Algorithms and Computation Theory (SIGACT) News*, ACM, New York, NY, USA, v. 36, p. 53–70, March 2005. ISSN 0163-5700. Disponível em: <http://doi.acm.org/10.1145/1052796.1052806>. Acesso em: October, 4, 2011.

Ren, F.; Lin, C.; Wei, B. A nonlinear control theoretic analysis to TCP-RED system. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, Elsevier North-Holland, Inc., New York, NY, USA, v. 49, p. 580–592, November 2005. ISSN 1389-1286. Disponível em: <http://dx.doi.org/10.1016/j.comnet.2005.01.016>. Acesso em: October, 4, 2011.

Renesse, R. van; Birman, K.; Hayden, M.; Vaysburd, A.; Karr, D. Building adaptive systems using ensemble. *Software: Practice and Experience*, John Wiley & Sons, Ltd., v. 28, n. 9, p. 963–979, 1998. ISSN 1097-024X. Disponível em: <http://dx.doi.org/10.1002/(SICI)1097-024X(19980725)28:9<963::AID-SPE179>3.0.CO;2-9>. Acesso em: October, 4, 2011.

Rimal, B. P.; Choi, E.; Lumb, I. A taxonomy and survey of cloud computing systems. In: *Proceedings of the 2009 Fifth International Joint Conference on INC, IMS and IDC*. Washington, DC, USA: IEEE Computer Society, 2009. (NCM'2009), p. 44–51. ISBN 978-0-7695-3769-6. Disponível em: <http://dx.doi.org/10.1109/NCM.2009.218>. Acesso em: October, 4, 2011.

Rochwerger, B. et al. The reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, IBM Corporation, Riverton, NJ, USA, v. 53, p. 535–545, July 2009. ISSN 0018-8646. Disponível em: <http://dx.doi.org/10.1147/JRD.2009.5429058>. Acesso em: October, 4, 2011.

Rütti, O.; Wojciechowski, P. T.; Schiper, A. Structural and algorithmic issues of dynamic protocol update. In: *Proceedings of the 20th international conference on Parallel and distributed processing*. Washington, DC, USA: IEEE Computer Society, 2006. (IPDPS'2006), p. 133–133. ISBN 1-4244-0054-6. Disponível em: <http://dx.doi.org/10.1109/IPDPS.2006.1639369>. Acesso em: October, 4, 2011.

Sá, A. S. de; Macêdo, R. J. A. An adaptive failure detection approach for real-time distributed control systems over shared ethernet. *ABCM Symposium Series in Mechatronics*, Assiciação Brasileira de Ciências Mecânicas (ACBM), v. 2, n. 1, p. 43–50, 2006.

Sá, A. S. de; Macêdo, R. J. A. Um framework para prototipagem e simulação de detectores de defeitos para sistemas de tempo real distribuídos de controle e supervisão. In: *VIII Workshop de Testes e Tolerância a Falhas*. Belém, PA, Brasil: SBRC/SBC, 2007. (WTF'2007), p. 43–56.

Sá, A. S. de; Macêdo, R. J. A. Uma proposta de detector autonômico de defeitos usando engenharia de controle. In: *Anáis X do Workshop de Testes e Tolerância a Falhas*. João Pessoa, PB, Brazil: SBC, 2009. (WTF'2009).

Sá, A. S. de; Macêdo, R. J. A. Detectores de defeitos autonômicos para sistemas distribuídos. In: *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Gramado, Brasil: SBRC / SBC, 2010. (SBRC'2010), p. 785–798. ISSN 2177-4978.

Sá, A. S. de; Macêdo, R. J. A. QoS self-configuring failure detectors for distributed systems. In: *Proceedings of the 10th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS' 2010)*. Berlin, Heidelberg: Springer-Verlag, 2010. (Lecture Notes in Computer Science (LNCS), v. 6115), p. 126–140. Disponível em: <http://dx.doi.org/10.1007/978-3-642-13645-0>. Acesso em: October, 4, 2011.

Sá, A. S. de; Macêdo, R. J. A.; Moreno, U.; Santos, T. Um procedimento para avaliação de redes ethernet comutada baseada em uma métrica de qualidade de controle. In: *XVII Congresso Brasileiro de Automática*. Juiz de Fora, MG, Brasil: SBA, 2008. (CBA'2008), p. 1–6.

Satzger, B. *Self-healing distributed systems*. 238 p. Tese (PHD thesis) — Universität Augsburg, Universitätsstr. 22, 86159 Augsburg, 2008.

Satzger, B.; Pietzowski, A.; Trumler, W.; Ungerer, T. A new adaptive accrual failure detector for dependable distributed systems. In: *Proceedings of the 2007 ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2007. (SAC'2007), p. 551–555. ISBN 1-59593-480-4. Disponível em: <http://doi.acm.org/10.1145/1244002.1244129>. Acesso em: October, 4, 2011.

Schneider, F. B. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Computing Surveys*, ACM, New York, NY, USA, v. 22, p. 299–319, December 1990. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/98163.98167>. Acesso em: October, 4, 2011.

Silberschatz, A.; Korth, H. F.; Sudarshan, S. *Sistemas de banco de dados*. 5nd. ed. Rio de Janeiro, RJ, Brasil: Editora Campus, 2006. 808 p. ISBN 10-85-352-1107-1.

Simon, D. Analyzing control system robustness. *IEEE Potentials*, v. 21, n. 1, p. 16–19, February 2002. ISSN 0278-6648. Disponível em: <http://dx.doi.org/10.1109/45.985322>. Acesso em: October, 4, 2011.

Sivasubramanian, S.; Alonso, G.; Pierre, G.; Steen, M. van. Globedb: autonomic data replication for web applications. In: *Proceedings of the 14th international conference on World Wide Web*. New York, NY, USA: ACM, 2005. (WWW'2005), p. 33–42. ISBN 1-59593-046-9. Disponível em: <http://doi.acm.org/10.1145/1060745.1060756>. Acesso em: October, 4, 2011.

So, K. C. W.; Sirer, E. G. Latency and bandwidth-minimizing failure detectors. In: *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*. New York, NY, USA: ACM, 2007. (EuroSys'2007), p. 89–99. ISBN 978-1-59593-636-3. Disponível em: <http://doi.acm.org/10.1145/1272996.1273008>. Acesso em: October, 4, 2011.

Tanenbaum, A. S. *Computer networks*. 4th. ed. Upper Saddle River, NJ 07458, USA: Prentice-Hall, 2003. ISBN 0-13-066102-3.

Tanenbaum, A. S. *Modern Operating Systems*. 3rd. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2007. ISBN 9780136006633.

Tanenbaum, A. S.; Steen, M. V. *Distributed systems: Principles and paradigms*. 2nd. ed. Upper Saddle, NJ 07458, USA: Prentice-Hall, 2007. 704 p. ISBN 0-132-39227-5.

Veríssimo, P.; Casimiro, A. The timely computing base model and architecture. *IEEE Transactions on Computers*, IEEE Computer Society, Washington, DC, USA, v. 51, p. 916–930, August 2002. ISSN 0018-9340. Disponível em: <http://dx.doi.org/10.1109/TC.2002.1024739>. Acesso em: October, 4, 2011.

Veríssimo, P.; Rodrigues, L. *Distributed systems for systems architects*. USA: Kluwer Academic Publishers, 2000. 623 p. ISBN 0-79-237266-2.

Veríssimo, P. E. Travelling through wormholes: a new look at distributed systems models. *SIGACT News*, ACM, New York, NY, USA, v. 37, p. 66–81, March 2006. ISSN 0163-5700. Disponível em: <http://doi.acm.org/10.1145/1122480.1122497>. Acesso em: October, 4, 2011.

Veríssimo, P.; Casimiro, A.; Fetzer, C. The timely computing base: Timely actions in the presence of uncertain timeliness. In: *Proceedings of the 2000 International Conference on Dependable Systems and Networks (formerly FTCS-30 and DCCA-8)*. Washington, DC, USA: IEEE Computer Society, 2000. (DSN'2000), p. 533–542. ISBN 0-7695-0707-7. Disponível em: <http://dx.doi.org/10.1109/ICDSN.2000.857587>. Acesso em: October, 4, 2011.

Wiesmann, M.; Urban, P.; Defago, X. An snmp based failure detection service. In: *Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems*. Washington, DC, USA: IEEE Computer Society, 2006. (SRDS'2006), p. 365–376. ISBN 0-7695-2677-2. Disponível em: <http://dx.doi.org/10.1109/SRDS.2006.9>. Acesso em: October, 4, 2011.

Wilfredo, T. *Software Fault Tolerance: A Tutorial*. Langley Research Center, Hampton, Virginia, October 2000. 66 p. Disponível em: <http://dl.acm.org/citation.cfm?id=886626>. Acesso em: October, 4, 2011.

Xiong, N.; Yang, Y.; Chen, J.; He, Y. On the quality of service of failure detectors based on control theory. In: *Proceedings of the 20th International Conference on Advanced Information Networking and Applications*. Vienna, Austria: IEEE Computer Society, 2006. (AINA'2006), p. 75–80. ISBN 0-7695-2466-4. ISSN 1550-445X. Disponível em: <http://dx.doi.org/10.1109/AINA.2006.248>. Acesso em: October, 4, 2011.

Yang, T. C. Networked control system: a brief survey. In: *IEEE Proceedings of Control Theory and Applications*. The Institute of Engineering and Technology (IET), 2006. (CTA'2006, 4), p. 403–412. ISSN 1350-2379. Disponível em: <http://dx.doi.org/10.1049/ip-cta:20050178>. Acesso em: October, 4, 2011.

Yellin, F. Low level security in java. In: O'reilly; Associates (Ed.). *Proceedings of the Fourth International World Wide Web Conference*. Boston, Massachusetts, USA: O'Reilly & Associates, Inc., 1995. (W3C'1995), p. 369–380. ISBN 1-56592-169-0. ISSN 1085-2301.

APÊNDICES

Apêndice

ASPECTOS GERAIS SOBRE AS PROPOSTAS DE DETECÇÃO AUTONÔMICA DE DEFEITOS

A.1 DEMONSTRAÇÃO DAS PRINCIPAIS EQUAÇÕES

A.1.1 Demonstração do modelo da planta $P(z)$ para a abordagem *RBL*

O objetivo do modelo da planta na abordagem *RBL* é descrever como o percentual de consumo de recursos (*RC*) varia quando varia-se a freqüência de monitoramento $\lambda = \frac{1}{\delta}$. Para obter tal modelo, deriva-se *RC* em relação a λ, assim (usando a 4.16):

$$\nu = \frac{\Delta RC_k}{\Delta \lambda_{k-1}} = (rtt^E - rtt^L) \tag{1.1}$$

Então, suponha que ν é constante e assuma que $u = \Delta\lambda$ e $y = \Delta RC$ são, respectivamente, a entrada e a saída do modelo da planta. Em seguida, aplique a transformada z para obter uma solução simplificada para o problema da modelagem da planta. A definição da transformada z para um sinal discreto $x = \{x_0, x_1, x_2, ...\}$ é:

$$\mathcal{Z}\{x\} = X(z) = \sum_{k=0}^{\infty} x_k * z^{-k}$$

Desde que supõe-se ν constante, então:

$$\mathcal{Z}\{\nu\} = \nu * \sum_{k=0}^{\infty} z^{-k} = \nu * \frac{z}{z-1} \tag{1.2}$$

A partir da definição de ν, pode-se afirmar que ν relaciona a entrada u a saída y e que a $(k-1)^{th}$ (i.e. u_{k-1}) afeta a k^{th} saída (i.e. y_k). Assim, é possível escrever:

$$\mathcal{Z}\{\nu\} = \frac{Y(z)}{z^{-1} * U(z)} \tag{1.3}$$

Conseqüentemente,

$$P(z) = \frac{Y(z)}{U(z)} = \frac{rtt^E - rtt^L}{z-1} \tag{1.4}$$

em que $P(z)$ é o modelo da planta que relaciona $u = \Delta\lambda$ a $y = \Delta RC$.

A.1.2 Demonstração do máximo sobre-sinal Mp e da lei de adaptação para o parâmetro Kp na abordagem *RBL*

A lei de adaptação Kp é obtida a partir da definição do máximo sobre-sinal. Sendo assim, primeiramente é apresentado como o máximo sobre-sinal para a abordagem *RBL* é obtido, então a lei de adaptação de Kp é apresentada.

Uma vez que o máximo sobre-sinal (*overshoot*) é a maior diferença percentual entre a saída do sinal e seu valor desejado (o setpoint), é possível defini-lo por (Hellerstein et al., 2004):

$$M_p^* = \frac{\left| y_{max} - y_{set\,point} \right|}{y_{set\,point}} \tag{1.5}$$

No caso da abordagem *RBL*, dado que $RC \in [0,1]$, então o valor máximo de y acontece quando *RC* varia de zero para aproximadamente 1. Assim, é possível escrever:

$$M_p^* = \frac{\left| 1 - RC^D \right|}{RC^D} \tag{1.6}$$

Observe que, a partir da Equação 1.6, $M_p^* \in (0,\infty)$, uma vez que $RC^D \in (0,1)$.

A abordagem *RBL* assume um sobre-sinal mais restritivo, definido por:

$$M_p = RC^D * M_p^* \tag{1.7}$$

isto é,

$$M_p = 1 - RC^D \tag{1.8}$$

Uma vez que $RC^D \in (0,1)$, o máximo sobre-sinal (M_p) para a abordagem *RBL* se restringe ao intervalo aberto entre 0 e 1.

A sensibilidade do controlador proporcional está diretamente ligada à magnitude do ganho Kp. Se Kp é largo, o controlador faz com que a saída da planta convirja mais rapidamente – entretanto, a saída da planta apresentará um sobre-sinal mais largo. Por outro lado, se Kp é muito pequeno, a saída da planta apresentará um menor sobre-sinal, mas levará um tempo maior até atingir o valor desejado. Para atender a essa relação de compromisso, utiliza-se o sobre-sinal, definido de acordo com a Equação 1.8, para determinar o valor de Kp da forma a seguir. Quando RC^D tende a 1 (M_p tende a zero), significa que o detector de defeitos pode consumir até 100% do percentual dos recursos disponível, então é possível assumir um ganho pequeno, permitindo que a freqüência de monitoramento convirja lentamente para um valor que

permita o detector usar apenas o percentual de recursos desejado. Contudo, se RC^D tende a zero (M_p tende a 1), significa que o ganho deve ser o mais largo possível para que se tenha uma convergência rápida da freqüência de monitoramento para a freqüência que minimiza o percentual de consumo de recursos do detector de defeitos.

Assim, seja a função de transferência em malha fechada $F_r(z) = \frac{C(z)*P(z)}{1+C(z)*P(z)}$, definida para a abordagem *RBL*, a lei de adaptação de Kp é obtida igualando o pólo de F_r ao máximo sobre-sinal definido, isto é:

$$1 - Kp * (rtt^E - rtt^L) = 1 - RC^D$$

Deste modo, obtêm-se:

$$Kp = \frac{RC^D}{rtt^E - rtt^L} \tag{1.9}$$

Entretanto, se $rtt^E = rtt^L$, o gestão autonômico assume $Kp = 1$. Observe que como rtt^L representa uma estimativa para o atraso de ida-e-volta mínimo, então rtt^E nunca é menor que rtt^L.

A.1.3 Demonstração do modelo da planta $P(z)$ para a abordagem *RBS*

O objetivo do modelo da planta na abordagem *RBS* é descrever como o atraso de interação (d) varia quando varia-se o período de monitoramento δ. Para obter tal modelo, deriva-se d^E em relação a δ, assim (usando a Equação 4.22):

$$\nu = -\frac{\Delta d_k^E}{\Delta \delta_{k-1}} = \frac{d^U - d^L}{\delta^U - \delta^L}$$

ν não é constante porque d^U, d^L, δ^U e δ^L podem variar. Entretanto, inicialmente supõe-se que ν é constante e assume-se que $u = \Delta\delta$ e $y = \Delta d^E$ são, respectivamente, a entrada e a saída do modelo da planta. Então, aplica-se transformada z para obter uma solução simplificada para o problema da modelagem da planta. A definição da transformada z para um sinal discreto $x = \{x_0, x_1, x_2, ...\}$ é:

$$\mathcal{Z}\{x\} = X(z) = \sum_{k=0}^{\infty} x_k * z^{-k}$$

Desde que supõe-se ν constante, então:

$$\mathcal{Z}\{\nu\} = \nu * \sum_{k=0}^{\infty} z^{-k} = \nu * \frac{z}{z-1} \tag{1.10}$$

A partir da definição de ν, pode-se afirmar que ν relaciona a entrada u a saída y e que a $(k-1)^{th}$ entrada (i.e. u_{k-1}) afeta a k^{th} saída (i.e. y_k). Assim, é possível escrever:

$$\mathcal{Z}\{\nu\} = \frac{Y(z)}{z^{-1} * U(z)} \tag{1.11}$$

Conseqüentemente,

$$P(z) = \frac{Y(z)}{U(z)} = \frac{\nu}{z-1} \tag{1.12}$$

em que $P(z)$ é o modelo da planta que relaciona $u = \Delta d^E$ a $y = \Delta\delta$.

A.1.4 Demonstração da lei de adaptação para os parâmetros K_P e K_I

Na abordagem *RBS*, a lei de adaptação para os parâmetros do laço de controle foram obtidos a partir da função de transferência em malha fechada:

$$F_r(z) = \frac{C(z) * P(z)}{1 + C(z) * P(z)}$$

Em que os pólos de F_r são a solução da equação característica $1 + C(z) * P(z) = 0$. Quando F_r é desenvolvido com relação a $C(z)$ e $P(z)$, obtem-se:

$$F_r(z) = \frac{[K_P * (z-1) + K_I * (z-1) + \Delta t * z] * \nu}{z^2 + [(K_P + K_I + \Delta t) * \nu - 2] * z + [1 - (K_P - K_I) * \nu]}$$

Em seguida, especifica-se a localização dos pólos de F_r a partir da definição do desempenho do controle e das métricas M_p e K_s. Então, assume-se dois pólos conjugados definidos por $cp = g * exp(\pm j\theta)$, em que $g = exp(\frac{-4}{K_s})$ e $\theta = \pi * \frac{\log(g)}{\log(M_p)}$ são a magnitude e o ângulo dos pólos do número complexo cp, respectivamente.

Assim, o problema é solucionar a expressão:

$$[z - g * exp(j\theta)] * [z - g * exp(-j\theta)] = 1 + C(z) * P(z) \tag{1.13}$$

Isto é:

$$z^2 - 2 * g * cos(\theta) * z + g^2 = z^2 + [(K_P + K_I + \Delta t) * \nu - 2] * z + [1 - (K_P - K_I) * \nu] \tag{1.14}$$

Finalmente, resolvendo a Expressão 1.14, encontra-se:

$$K_P = \frac{\phi - g^2}{\psi}$$

e

$$K_I = \frac{g^2 - 2 * g * cos(\theta) + 1}{\psi}$$

em que

$$\phi = \frac{1}{d^U - d^L}$$

e

$$\psi = \frac{1}{\delta^U - \delta^L}$$

A.1.5 Demonstração das equações para os limites do período de monitoramento (δ)

Para evitar que o gestor autonômico sugira períodos de monitoramento que decremente a qualidade do serviço do detector de defeitos ou consuma de forma inapropriada muitos recursos computacionais, deve-se estimar os limites nos quais o período de monitoramento pode ser ajustado. Para tanto, analisa-se o período de monitoramento com relação aos tempo de detecção máximo e mínimo, conforme a seguir.

Propriedade A.1 ***Tempo de detecção no melhor caso****: Se $tt(j,i)$ é o tempo de viagem da mensagem de um processo p_j para um processo p_i, então, no melhor caso, p_i leva $tt(j,i)$ unidades de tempo para detectar, com segurança, a falha de p_j.*

Demonstração. O melhor caso para a detecção de uma falha é aquele em que p_j falha imediatamente após receber um "are you alive?´´. Neste caso, a falha será detectada quando o *timeout* de detecção rto expirar em p_i. Suponha que o tempo de viagem da mensagem de p_i para p_j é $tt(i,j)$. Então, para que p_i detecte a falha de p_j, com segurança, é preciso que $rto \geq tt(i,j) + tt(j,i)$. Se p_i envia um "are you alive?´´ no instante t, em relação ao tempo real, esta mensagem será recebida por p_j em $t_f = t + tt(i,j)$. Se p_j falha exatamente em t_f, então, no melhor caso, p_i detectará a falha de p_j em $t_s = t_f + tt(j,i)$. Assim, o tempo de detecção, neste caso, será $TD = t_s - t_f$, i.e. $TD = tt(j,i)$. ∎

A partir da Propriedade A.1 é possível determinar o tempo mínimo de detecção TD^L, conforme a seguir.

Propriedade A.2 ***Tempo de detecção mínimo*** *(TD^L): o menor intervalo de tempo possível para que um processo monitor p_i detecte a falha de um processo monitorado p_j é sempre equivalente ao menor atraso fim-a-fim d^L.*

Demonstração. O tempo de detecção mínimo acontece no melhor caso de detecção. Assim, a partir da Proprieade A.1, o tempo de detecção no melhor caso é $TD = tt(j,i)$. Se d^L é o menor atraso fim-a-fim, então o menor tempo de detecção acontece quando $tt(j,i) = d^L$. Neste caso, $TD = d^L$ e, consequentemente, $TD^L = d^L$. ∎

Note que, os tempos de detecção determinados pelas propriedades A.1 e A.2 independem do período de monitoramento.

No modelo caixa-preta, considerado para o projeto da regulação de período, o d^L é uma estimativa para o atraso nominal do sistema. Assim, períodos de monitoramento menor que d^L implicariam em congestionamentos no sistema, sem benefícios para o tempo mínimo de detecção. Por conta disto, assume-se que o período de monitoramento deve ser no mínimo d^L (i.e. $\delta^L = d^L$).

Propriedade A.3 ***Tempo de detecção no pior caso****: Se δ é o período de monitoramento e $tt(j,i)$ é o tempo de viagem da mensagem de um processo p_j para um processo p_i, então, no pior caso, p_i leva pelo menos $\delta + tt(j,i)$ unidades de tempo para detectar, com segurança, a falha de p_j.*

Demonstração. O pior caso para a detecção de uma falha é aquele em que p_j falha imediatamente depois de enviar um "heartbeat´´. Para este caso, suponha que o tempo de viagem da mensagem de p_i para p_j é $tt(i,j)$. Para que p_i detecte uma falha de p_j, com segurança, é preciso que $rto \geq tt(i,j) + tt(j,i)$. Se p_i envia um "are you alive?´´ no instante t, em relação ao tempo real, esta mensagem será recebida por p_j em $t_f = t + tt(i,j)$. Se p_j envia o *heartbeat* e falha em t_f, então, neste caso, p_i receberá o heartbeat em p_j em $t_a = t_f + tt(j,i)$. No próximo ciclo de monitoramento, p_i enviará um novo heartbeat em $t_b = t + \delta$. Entretanto, como p_j falhou, p_i suspeitará de p_j em $t_s = t_b + rto$. Assim, o tempo de detecção, neste caso, será $TD = t_s - t_f$, i.e. $TD = (t_b + rto) - [t + tt(i,j)] = [t + \delta + rto] - [t + tt(i,j)] = \delta + rto - tt(i,j)$. Como $rto \geq tt(i,j) + tt(j,i)$, então $TD \geq \delta + tt(j,i)$. ∎

Note que, no pior caso, é desejado que TD seja menor ou igual ao tempo máximo de detecção especificado pelo usuário (i.e. $TD \leq TD^U$). Para que isto seja possível, é necessário que pelo menos o limite inferior de TD, no pior caso, seja menor ou igual a TD^U, i.e. $\delta + tt(j,i) \leq TD^U$. Consequemente, o período de monitoramento é limitado da seguinte forma: $\delta \leq TD^U - tt(j,i)$. Com isso, o período de monitoramento pode ser máximo quando $tt(j,i) = d^L$. Assim, o maior período de monitoramento possível é $\delta^U = TD^U - d^L$.

A.2 IMPACTO DAS MÉTRICAS DE QOS NO DESEMPENHO DO DETECTOR AUTONÔMICO

Esta seção apresenta uma análise experimental para avaliar o impacto das métricas de QoS no desempenho dos detectores autonômicos propostos. Essas análises são realizadas de modo a

ajudar os projetistas no entendimento de como a configuração dos detectores autonômicos afeta seu desempenho em termos dos requisitos de qualidade de serviço.

A.2.1 Configuração dos experimentos

Os experimentos consideram o mesmo ambiente de simulação usado na avaliação de desempenho realizada na Seção 4.4.1 do Capítulo 4. Entretanto, as rajadas de mensagens geradas pelo processo em c_3 são controladas por uma variável denominada $NETLOAD$, a qual determina o percentual de recursos de rede que são consumidos por tais rajadas de mensagens.

A.2.2 Fatores e métricas de desempenho

Para a avaliação experimental, são definidos como fatores que afetam o desempenho do detector autonômico, os seus parâmetros de configuração (i.e. TD^U, AV^L e RC^D, ver Seção 4.3) e as condições de carga impostas pelas aplicações de usuário no ambiente combutacional – estas últimas especificadas experimentalmente através da variável $NETLOAD$.

Como métricas de desempenho são usadas as métricas de qualidade de serviço de detecção (i.e. TD, TM e TMR) e a taxa de falsas suspeitas cometidas pelo detector autonômico (RM).

A.2.3 Detectores autonômicos avaliados

Na avaliação experimental são consideradas ambas as abordagens de detecção autonômica (i.e. *RBL* e *RBS*). Entretanto, diferentemente da avaliação de desempenho apresentada na Seção 4.4.3 do Capítulo 4, são consideradas duas versões dos detectores autonômicos para cada uma das abordagens, isto é:

- *JAFD-RBL* – detector autonômico baseado na abordagem de regulação de período *RBL*, encapsulando o estimador de *timeouts* de Jacobson (1988);
- *BAFD-RBL* – detector autonômico baseado na abordagem de regulação de período *RBL*, encapsulando o estimador de *timeouts* de Bertier, Marin e Sens (2002);
- *JAFD-RBS* – detector autonômico baseado na abordagem de regulação de período *RBS*, encapsulando o estimador de *timeouts* de Jacobson (1988);
- *BAFD-RBS* – detector autonômico baseado na abordagem de regulação de período *RBS*, encapsulando o estimador de *timeouts* de Bertier, Marin e Sens (2002);

A.2.4 Metodologia de avaliação experimental adotada

A avaliação experimental considera a metodologia de projeto experimental 2^q_r fatorial, em que q e r referem-se, respectivamente, ao número de fatores e ao número de repetições dos

experimentos, ver Jain (1991). Essa metodologia não propriamente identifica a relação entre os parâmetros de configuração e o desempenho do detector autonômico, mas pode prover uma expectativa sobre o quão importante é cada um dos parâmetros de configurações na obtenção do desempenho desejado para o detector.

Na análise de experimentos, considera-se os quatro fatores apresentados (i.e. $q = 4$), ou seja: RC^D, TD^U, AV^L e $NETLOAD$. Tais análises consideram a troca de 3000 mensagens de monitoramento entre os processos monitores e monitorados (i.e. $r = 3000$). Desta forma, foram realizados 2^4 experimentos com 3000 amostras cada um – totalizando 48000 amostras.

Na metodologia de análise experimental 2^q_r, os fatores considerados secundários[1] são fixados, enquanto que os fatores básicos (i.e. RC^D, TD^U, AV^L e $NETLOAD$) variam. Para tanto, os fatores básicos são variados considerando dois valores para cada um. Um desses valores determina impactos positivos em uma ou mais métricas de desempenho, enquanto o outro implica em impactos negativos (ver Tabela A.1).

Tabela A.1. Fatores e seus níveis de impactos

Apelido do fator	Nome do fator	Níveis de impacto	
		-	+
		Valores considerados	
x_1	TD^U	$0,6072ms$	$60,72ms$
x_2	AV^L	0	1
x_3	RC^D	0	1
x_4	$NETLOAD$	90%	0%

Por examplo, $0.6072ms$ é o atraso de ida-e-volta mínimo para cada mensagem de monitoramento. Se $TD^U = 0.6072ms$, então o gestor autonômico do detector não ajusta de forma apropriada o período de monitoramento sob altas condições de carga (i.e. $NETLOAD = 90\%$). Assim, espera-se que o detector autonômico possua um baixo desempenho em termos do tempo de detecção – neste caso, associa-se um nível de impacto negativo para esse valor de TD^U. Por outro lado, para $TD^U = 60.72ms$, o qual representa 10 vezes o atraso de ida-e-volta mínimo, é esperado que o detector autonômico ajuste de forma adequada o período de monitoramento sob altas condições de carga, apresentando assim um bom desempenho em termos do tempo de detecção – neste caso, associa-se um nível de impacto positivo para esse valor de TD^U.

Considerações similares são realizadas para os demais fatores básicos apresentados na Tabela A.1. Caso os níveis de impacto para cada um dos valores sejam diferentes daqueles supostos durante o projeto dos experimentos, os valores percentuais médios de impactos observados após a execução das análises terão valores negativos.

De acordo com a metodologia 2^q_r, na execução da avaliação experimental, todas os detectores são configurados considerando todas as combinações de valores dos fatores básicos

[1] e.g. tipo de rede, taxa nominal de transferência da rede, tamanho do buffer de mensagens etc.

definidos na Tabela A.1. Então, os valores médios do desempenho em termos das métricas consideradas são observados. Em seguida, o percentual de variação em cada uma das métricas de desempenho são obtidos usando o método dos mínimos quadrados. Esses percentuais de variação compõem uma tabela que determina o percentual de impacto de cada um dos fatores básicos (e de cada uma de suas interações) no desempenho do detector autonômico considerado. Além disso, erros experimentais são calculados e registrados. Esses erros experimentais dizem respeito aos impactos de fatores secundários que não foram considerados na avaliação experimental. A depender da magnitude dos erros experimentais novos experimentos podem ser conduzidos incorporando outros fatores que foram desconsiderados em uma avaliação preliminar. Para uma discussão mais detalhada sobre a metodologia de avaliação experimental 2_r^q, veja Jain (1991).

A.2.5 Resultado da avaliação experimental

As tabelas A.2, A.3, A.4 e A.5 apresentam o percentual de impacto de cada um dos fatores básicos e de suas interações sobre o desempenho dos detectores autonômicos em termos do tempo de detecção, da duração da falsa suspeita, da taxa de falsas suspeitas e do intervalo entre falsas suspeitas, respectivamente. As colunas de cada uma dessas tabelas somam 100%. Nas tabelas, as interações entre fatores são apresentados no padrão *x*-*y*, significando que o percentual de impacto no desempenho refere-se à combinação do fator *x* com o fator *y*. A última linha de cada tabela apresenta os erros experimentais encontrados. As linhas em cinza representam os fatores e/ou interações com percentual de impacto relevante para o desempenho em termos da métrica considerada.

A.2.5.1 Impacto dos fatores e de suas interações no tempo de detecção.

A Tabela A.2 apresenta o percentual de impacto dos fatores e de suas interações no desempenho em termos do tempo de detecção para as diferentes versões do detector autonômico. Conforme pode ser visto nessa tabela, RC^D e TD^U e sua interação (RC^D-TD^U) possui impacto relevante no desempenho de todas as versões – fatores e interações somados representam entre 53% e 95% do desempenho em termos do tempo de detecção para cada versão do detector autonômico. Conforme esperado, isto indica que é possível explorar os parâmetros RC^D e TD^U para obter boas alternativas de desempenho em termos do tempo de detecção. Observe que os detectores autonômicos na abordagem *RBL*, usando os estimadores de Jacobson (1988) (*JAFD-RBL*) e de Bertier, Marin e Sens (2002) (*BAFD-RBL*), somam 95% do desempenho em termos do tempo de detecção – quando os fatores RC^D e TD^U e suas interações são considerados. Para os detectores autonômicos com a abordagem *RBS*, por outro lado, os fatores RC^D e TD^U e sua interação têm percentual de impacto de 53% e 81% para *JAFD-RBS* e *BAFD-RBS*, respectivamente.

No caso de *JAFD-RBS*, existe uma forte influência dos fatores $NETLOAD$ e AV^L e de

Tabela A.2. Percentual de impacto no tempo de detecção

Fatores e Interações	*JAFD-RBL*	*BAFD-RBL*	*JAFD-RBS*	*BAFD-RBS*
AV^L	2	2	1	5
RC^D	33	35	14	23
TD^U	27	24	24	34
$NETLOAD$	1	1	13	4
AV^L-RC^D	0	0	7	1
AV^L-TD^U	1	1	2	0
AV^L-$NETLOAD$	0	0	10	2
RC^D-TD^U	35	36	13	24
RC^D-$NETLOAD$	0	0	0	1
TD^U-$NETLOAD$	0	0	4	0
AV^L-RC^D-TD^U	0	0	7	1
AV^L-RC^D-$NETLOAD$	0	0	0	1
AV^L-TD^U-$NETLOAD$	0	0	5	0
RC^D-TD^U-$NETLOAD$	0	0	0	1
AV^L-RC^D-TD^U-$NETLOAD$	0	0	0	1
Erro experimental	0	0	1	1

suas interações. Observando apenas os fatores $NETLOAD$ e AV^L, e as interações entre os mesmos e dos mesmos com os demais fatores, obtem-se um impacto percentual de 48% no tempo de detecção de *JAFD-RBS*. Isto porque, o estimador de Jacobson (1988) se ajusta mais rapidamente à variação da carga que o estimador de Bertier, Marin e Sens (2002)[2], o que deixa o mesmo mais susceptível a falsas suspeitas por conta de variações na carga. Dessa forma, o estimador de Jacobson (1988) precisa de uma maior intervenção do regulador de *timeout* da abordagem autonômica – o que explica a influência de AV^L. Por outro lado, a influência da ação integral do controlador PI usado pela abordagem *RBS*, torna os detectores mais lentos à variação de carga, o que implica em uma interferência maior do período de monitoramento na carga da rede e, consequentente, uma maior susceptibilidade dos estimadores de *timeout*, usados por esses detectores, à variação de carga – o que explica um aumento da influência do fator $NETLOAD$ e de suas interações com os demais para o caso de *JAFD-RBS* e de *BAFD-RBS*.

[2]Observe que o estimador de Jacobson (1988) tem um atraso de fase de $(\frac{1}{1-0,1} \cong 1)$ (i.e. estima usando o último valor), enquanto que o estimador de Bertier, Marin e Sens (2002) tem atraso de fase de $\frac{1}{\frac{1}{1000}} \cong 1000$ (i.e. estima usando uma janela de 1000 valores).

A.2.5.2 Impacto dos fatores e de suas interações na duração da falsa suspeita.

A Tabela A.3 apresenta o percentual de impacto dos fatores e de suas interações no desempenho em termos da duração da falsa suspeita para as diferentes versões do detector autonômico. Conforme pode ser visto nessa tabela, AV^L e TD^U e sua interação (AV^L-TD^U) possui impacto relevante no desempenho – esses fatores e interações somados representam entre 64% e 80% do desempenho em termos da duração da falsa suspeita para cada versão do detector autonômico. Conforme esperado, isto indica que é possível explorar os parâmetros AV^L e TD^U para obter boas alternativas de desempenho em termos da duração da falsa suspeita. Observe que os detectores autonômicos na abordagem *RBL*, usando os estimadores de Jacobson (1988) (*JAFD-RBL*) e de Bertier, Marin e Sens (2002) (*BAFD-RBL*), somam 80% do desempenho em termos da duração da falsa suspeita – quando os fatores AV^L e TD^U e suas interações são considerados. Para os detectores autonômicos com a abordagem *RBS*, por outro lado, os fatores AV^L e TD^U e sua interação têm percentual de impacto de 64% e 76% para *JAFD-RBS* e *BAFD-RBS*, respectivamente.

Tabela A.3. Percentual de impacto na duração da falsa suspeita

Fatores e Interações	*JAFD-RBL*	*BAFD-RBL*	*JAFD-RBS*	*BAFD-RBS*
AV^L	24	26	29	25
RC^D	0	2	3	5
TD^U	34	34	19	29
NETLOAD	1	0	0	2
AV^L-RC^D	1	0	0	2
AV^L-TD^U	22	20	16	22
AV^L-*NETLOAD*	0	3	6	0
RC^D-TD^U	3	0	1	0
RC^D-*NETLOAD*	0	2	6	5
TD^U-*NETLOAD*	1	3	7	0
AV^L-RC^D-TD^U	1	0	0	1
AV^L-RC^D-*NETLOAD*	1	0	1	2
AV^L-TD^U-*NETLOAD*	0	5	6	0
RC^D-TD^U-*NETLOAD*	3	0	3	0
AV^L-RC^D-TD^U-*NETLOAD*	1	0	0	1
Erro experimental	7	6	4	4

Dentre as versões do detector autonômico, AV^L possui maior impacto percentual na duração da falsa suspeita quando se observa *JAFD-RBS*, o que reforça a argumentação apresentada na Seção A.2.5.1 sobre a influência de tal métrica no tempo de detecção, isto é: AV^L influência o tempo de detecção por conta da maior susceptibilidade do estimador de Jacobson (1988) a variação da carga, então também afetará a duração da falsa suspeita – uma vez que, a correção realizada pelo regulador de *timeout* pode ser insuficiente em alguns casos.

A.2.5.3 Impacto dos fatores e de suas interações na taxa de falsas suspeitas.

A Tabela A.4 apresenta o percentual de impacto dos fatores e de suas interações no desempenho em termos da taxa de falsas suspeitas para as diferentes versões do detector autonômico. Conforme pode ser visto nessa tabela, AV^L é um fator dominante do desempenho de todas as versões. Este fator apresenta percentuais de impacto na taxa de falsas suspeitas de 44%, 52%, 48% e 46% para *JAFD-RBL*, *BAFD-RBL*, *JAFD-RBS* e *BAFD-RBS*, respectivamente.

Tabela A.4. Percentual de impacto na taxa de falsas suspeitas

Fatores e Interações	*JAFD-RBL*	*BAFD-RBL*	*JAFD-RBS*	*BAFD-RBS*
AV^L	44	52	48	46
RC^D	3	2	2	4
TD^U	4	8	4	4
$NETLOAD$	6	3	6	4
AV^L-RC^D	2	2	2	2
AV^L-TD^U	6	9	8	7
AV^L-$NETLOAD$	5	3	6	3
RC^D-TD^U	3	2	3	4
RC^D-$NETLOAD$	2	1	2	2
TD^U-$NETLOAD$	2	2	2	1
AV^L-RC^D-TD^U	2	2	3	2
AV^L-RC^D-$NETLOAD$	1	1	1	1
AV^L-TD^U-$NETLOAD$	2	1	2	1
RC^D-TD^U-$NETLOAD$	2	1	3	2
AV^L-RC^D-TD^U-$NETLOAD$	1	1	2	1
Erro experimental	16	9	7	17

Vale salientar que, no caso da taxa de falsas suspeitas, os percentuais de impacto de RC^D, de TD^U e de $NETLOAD$ e de suas interações somados significam algo entre 37% e 45%. Isto significa que tanto o fator diretamente relacionado à regulação do *timeout* (i.e. AV^L) quanto os fatores relacionados à regulação de período (i.e. RC^D e TD^U) e as variações de carga ($NETLOAD$) possuem uma importância significativa no desempenho da abordagem autonômica para o caso da taxa de falsas suspeitas.

Observe que o erro experimental observado está entre 9% e 17% para a taxa de falsas suspeitas, o que é um pouco mais representativo que os observados para o tempo de detecção e para a duração da falsa suspeita – significando que os fatores secundários passam a ser um pouco mais relevante quando o desempenho em termos da taxa de falsas suspeitas é considerada.

A.2.5.4 Impacto dos fatores e de suas interações no intervalo entre falsas suspeitas.

A Tabela A.5 apresenta o percentual de impacto dos fatores e de suas interações no desempenho em termos do intervalo entre falsas suspeitas para as diferentes versões do detector autonômico. Conforme pode ser visto nessa tabela, existe um espalhamento dos efeitos dos impactos percentuais dos fatores no desempenho da detecção autonômica. Isto se deve ao fato do intervalo entre falsas suspeitas depender não apenas dos fatores básicos relacionados a configuração (i.e. TD^U, RC^D e AV^L), mas também como esses fatores se relacionam com as diferentes condições de carga (representado pelo fator $NETLOAD$).

Tabela A.5. Percentual de impacto no intervalo entre falsas suspeitas

Fatores e Interações	*JAFD-RBL*	*BAFD-RBL*	*JAFD-RBS*	*BAFD-RBS*
AV^L	5	3	2	3
RC^D	13	13	11	13
TD^U	11	12	10	12
$NETLOAD$	5	8	10	9
AV^L-RC^D	4	2	1	2
AV^L-TD^U	3	2	1	1
AV^L-$NETLOAD$	1	1	2	1
RC^D-TD^U	12	13	11	14
RC^D-$NETLOAD$	5	8	10	8
TD^U-$NETLOAD$	4	8	10	8
AV^L-RC^D-TD^U	4	2	1	2
AV^L-RC^D-$NETLOAD$	1	1	2	1
AV^L-TD^U-$NETLOAD$	1	1	2	1
RC^D-TD^U-$NETLOAD$	4	7	10	9
AV^L-RC^D-TD^U-$NETLOAD$	1	1	2	1
Erro experimental	24	20	18	17

O espalhamento é verificado, quando se observa que nenhum dos fatores ligados à configuração do detector autonômico responde sozindo pelo desempenho percentual em termos do intervalo entre falsas suspeitas – todos esses fatores sozinhos possuem impacto percentual abaixo de 15%. Entretanto, os efeitos de TD^U e RC^D e de sua interação (RC^D-TD^U) são os significativos em termos dessa métrica – apresentando somátorios de impactos percentuais entre 32% e 39%. Isto porque, essas métricas estão diretamente ligadas à regulação do periodo de monitoramento, o qual é um fator determinante para o desempenho em termos do intervalo entre falsas suspeitas – uma vez que, quanto maior o período de monitoramento maior o intervalo entre falsas suspeitas (independentemente dos demais fatores). Quando, além de RC^D, TD^U e RC^D-TD^U, AV^L e suas interações são somados, a faixa percentual de impacto se acomoda entre 35% e 47% – isto é, existe um aumento no impacto de 3 a 15%, a depender da versão do detector autonômico.

Por outro lado, $NETLOAD$ e suas interações com os demais fatores respondem entre 22% e 48% dos impactos percentuais no desempenho das diferentes versões do detector autonômico. Isto porque, dada uma variação na carga, o gestor autonômico deve: (a) encontrar um período de monitoramento que reduza a interferência do detector nas variações de carga; e (b) realizar correções no *timeout* de detecção até que o detector possua a confiabilidade desejada. Esse intervalo transiente no qual o detector encontra a sintonia correta para o período de monitoramento, implica em maiores oscilações de carga, o que justifica a influência de $NETLOAD$ e de suas interações no desempenho do detector autonômico – observe que $NETLOAD$ responde sozinho com impactos percentuais entre 5% e 10% do desempenho, o que é inferior aos impactos percentuais de RC^D e TD^D sozinhos, mas é superior ao impacto de AV^L.

Observe ainda que neste experimento, o erro experimental representa entre 17% e 24%, significando que existem fatores secundários que possuem impacto percentual significativo em termos do intervalo entre falsas suspeitas – por exemplo, os parâmetros de configuração dos estimadores de *timeout* de Jacobson (1988) e de Bertier, Marin e Sens (2002).

A.3 DESEMPENHO DE AFD-RBS COM ESTIMADOR DE BERTIER, MARIN E SENS

Esta seção demonstra como o detector autonômico usando estimador de Bertier, Marin e Sens (2002) expõe melhor desempenho quando comparado com o detector adaptativo com o mesmo estimador. Para tanto, utiliza-se a versão de detector autonômico com a abordagem de regulação de período que obteve maior confiabilidade (i.e. *RBS*), considerando as avaliações de desempenho realizadas na Seção 4.4 do Capítulo 4. Assim, para efeito de comparação, são considerados os seguintes detectores: (a) $Bertier$ – detector adaptativo baseado no estimador de *timeout* de Bertier, Marin e Sens (2002); (b) $BAFD\text{-}RBS$ – detector autonômico baseado na abordagem de regulação de período *RBS*, encapsulando o estimador de *timeouts* de Bertier, Marin e Sens (2002);

Tabela A.6. Comparativo entre as versões adaptativa e autonômica do detector de Bertier

Detector	Métricas de desempenho									
	TD (ms)		TM (ms)		TMR (ms)		RM		AV	
	mean	std	mean	std	mean	std	mean	std	mean	std
Bertier($\delta = 1ms$)	2,69	4,24	2,40	1,31	3,47	0,61	0,3001	0,1269	0,3398	0,3504
Bertier($\delta = 3ms$)	3,35	0,06	0,10	0,03	9,28	3,95	0,3823	0,1984	0,9849	0,0154
Bertier($\delta = 5ms$)	5,34	0,71	0,11	0,04	12,55	4,17	0,4600	0,2107	0,9881	0,0105
$BAFD\text{-}RBS$	2,23	0,71	0,05	0,01	528,10	270,32	0,0011	0,0013	0,9998	0,0002

A Tabela A.6 apresenta um sumário do desempenho desses detectores. Nessa tabela, as colunas nomeadas *mean* e *std* representam, respectivamente os valores médios e os seus desvios padrão. Assim como na avaliação de desemepnho da Seção 4.4, é considerado o detector adaptativo de Bertier, Marin e Sens (2002) configurado, manualmente, com três períodos de monitoramento (i.e. 1, 3 e 5ms).

Observe que a versão autonômica do detector (*BAFD-RBS*) possui desempenho superior, em todas as métricas, que todas as configurações da versão adaptativa.

Apêndice

B

VARIÁVEIS USADAS PELO PROTOCOLO AUTONÔMICO DE COMUNICAÇÃO EM GRUPO

A Tabela B.1 apresenta um resumo das variáveis usadas nos algoritmos da abordagem autonômica de comunicação em grupo (ver Seção 5.4).

Tabela B.1. Resumo das variáveis usadas pelo protocolo autonômico de comunicação em grupo

n_{rv}	número total de mensagens recebidas/enviadas por um membro do grupo.
n_{ct}	número total de mensagens de controle recebidas/enviadas por um membro do grupo.
rtt	atraso fim-a-fim entre um canal (p_i, p_j).
ovh	sobrecarga média de mensagens de controle estimada
ovh_{max}	sobrecarga máxima estimada.
d	estimativa do atraso fim-a-fim.
d_{mean}	atraso fim-a-fim médio estimado a partir de um histórico de $w = 100$ mensagens.
d_{max}	atraso fim-a-fim máximo observado durante a execução do protocolo, assumindo uma margem de segurança $\beta = 0.1$ e um fator de esquecimento $\phi = 0.99999$.
d_{min}	atraso fim-a-fim mínimo observado durante a execução do protocolo, assumindo um fator de esquecimento $\phi = 0.99999$.
RC	estimativa do percentual de consumo de recursos no ambiente computacional.
τ	um vetor que mantém uma estimativa dos intervalos entre chegadas de mensagens recebidas por um processo do grupo.
τ_{max}	representa o maior intervalo entre chegadas observado, durante a execução do protocolo, considerando uma margem de segurança $\beta = 0.1$.
A	um vetor que mantém os instantes de chegada das mensagens recebidas por um processo do grupo.
ts_{max}	*time-silence* máximo, estimado a partir de τ_{max}, considerando uma margem de segurança $\beta = 0.1$.
RC_D	requisito definido pelo usuário (ou aplicação) em termos de consumo de recursos.
ovh_D	set-point dinâmico em termos de sobrecarga de mensagens.
K_P	ganho proporcional do controlador usado no ajuste de *time-silence*. Obtido experimentalmente e definido como $K_p = 4$.