Vinícius Angiolucci Reis

Eleição de Líder com Qualidade de Serviço para o Modelo Falha-e-Recuperação

Sorocaba, SP

05 de Maio de 2017

Vinícius Angiolucci Reis

Eleição de Líder com Qualidade de Serviço para o Modelo Falha-e-Recuperação

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação (PPGCC-So) da Universidade Federal de São Carlos como parte dos requisitos exigidos para a obtenção do título de Mestre em Ciência da Computação. Linha de pesquisa: Redes de Computadores e Engenharia de Software.

Universidade Federal de São Carlos – UFSCar

Centro de Ciências em Gestão e Tecnologia – CCGT

Programa de Pós-Graduação em Ciência da Computação – PPGCC-So

Orientador: Prof. Dr. Gustavo Maciel Dias Vieira

Sorocaba, SP

05 de Maio de 2017

Angiolucci Reis, Vinícius

Eleição de Líder com Qualidade de Serviço para o Modelo Falha-e-Recuperação / Vinícius Angiolucci Reis. -- 2017.
68 f. : 30 cm.

Dissertação (mestrado)-Universidade Federal de São Carlos, campus Sorocaba, Sorocaba
Orientador: Prof. Dr. Gustavo Maciel Dias Vieira
Banca examinadora: Prof. Dr. Irineu Sotoma, Profa. Dra. Islene Calciolari Garcia
Bibliografia

1. Algoritmos distribuídos. 2. Escalabilidade e confiabilidade. 3. Tolerância a falhas. I. Orientador. II. Universidade Federal de São Carlos. III. Título.

Ficha catalográfica elaborada pelo Programa de Geração Automática da Secretaria Geral de Informática (SIn).
DADOS FORNECIDOS PELO(A) AUTOR(A)

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências em Gestão e Tecnologia

Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a defesa de dissertação de mestrado do candidato Vinícius Angiolucci Reis, realizada em 05/05/2017:

Prof. Dr. Gustavo Maciel Dias Vieira
UFSCar

Prof. Dr. Irineu Sotoma
UFMS

Profa. Dra. Islene Calciolari Garcia
UNICAMP

Certifico que a sessão de defesa foi realizada com a participação à distância do membro Prof. Dr. Irineu Sotoma e, depois das arguições e deliberações realizadas, o participante à distância está de acordo com o conteúdo do parecer da comissão examinadora redigido no relatório de defesa do aluno Vinícius Angiolucci Reis.

Prof. Dr. Gustavo Maciel Dias Vieira
Presidente da Comissão Examinadora
UFSCar

Dedicado à minha família.

Agradecimentos

Agradeço aos meus pais, Plínio e Rosângela, pelo carinho e apoio incondicional durante todos estes anos que dediquei aos estudos. Obrigado pelas caronas de madrugada, pela comida sempre quentinha ao final do dia, pelas palavras de conforto e motivação que nunca me faltaram e por muito, muito mais coisas que não caberiam aqui.

Ao meu irmão, Rafael, que sempre esteve presente e assumiu as responsabilidades familiares de modo primoroso. A verdade é que como irmão mais velho sempre pensei que eu deveria dar o exemplo, mas eu percebo que tenho muito mais a aprender com você do que ensinar.

À minha namorada Priscila, pelo companheirismo inabalável em todos os momentos, desde os primeiros anos de graduação. Sua ajuda, carinho e otimismo tornam qualquer jornada possível e divertida.

Ao meu orientador e amigo Gustavo, que de modo incansável e sempre entusiasmado se dedicou ao projeto e ao meu desenvolvimento como pessoa, muito além do domínio acadêmico. Seu brilhantismo e profissionalismo são algumas de suas características em que vou sempre me espelhar.

Não posso expressar aqui o carinho e a imensa dívida de gratidão que carrego por vocês, pois nada do que realizei nos últimos anos seria possível sem todo o apoio que me deram.

Aos amigos e colegas que fiz durante estes anos todos, tive a oportunidade de conhecer e trocar experiências com pessoas realmente formidáveis.

Aos professores e à Universidade pela estrutura oferecida durante toda a pesquisa.

E à Capes, pelo suporte financeiro oferecido.

“ Quando penso que cheguei ao meu limite,
descubro que é possível superá-lo.”
(Ayrton Senna)

Resumo

Um dos objetivos de um sistema distribuído é prover poder computacional e persistência de dados mesmo na presença de falhas de um subconjunto de enlaces e processos. Para determinar quais são os processos falhos deste sistema e abstrair o conceito de tempo, estes sistemas utilizam os serviços de um detector de falhas não confiável, encapsulado em um algoritmo de eleição de líder. Embora a literatura sobre a *qualidade de serviço (QoS)* oferecida por detectores de falhas seja abundante, ela é escassa quando se trata da QoS oferecida por algoritmos de eleição de líder. Neste trabalho propomos um algoritmo de eleição de líder para o modelo falha-e-recuperação denominado NFD-L, que segue as especificações de QoS originalmente apresentadas por Chen, Toueg e Aguilera (2002). Utilizamos NFD-L em uma aplicação para replicação, como mecanismo de eleição de coordenador para Paxos e apresentamos uma análise da QoS observada, comparando o seu comportamento com um algoritmo de eleição de líder que não foi projetado explicitamente para prover garantias de QoS.

Palavras-chaves: Qualidade de serviço. Tolerância a falhas. Algoritmos distribuídos.

Abstract

A distributed system is a set of processes and links that is desgined to provide computing power and data persistency, even on the presence of failures. To encapsulate the abstraction of time and to determine which processes have currently failed, these distributed systems are based on unreliable failure detectors, which in its turn are used as a leader election service. Many works are dedicated to analyze the *quality of service (QoS)* of failure detectors, but a few of them has analyzed the QoS of a leader election algorithm. In this work, we present the NFD-L leader election algorithm, designed to work on crash-recovery distributed systems and to follow the QoS specification defined by Chen, Toueg e Aguilera (2002). We used NFD-L to elect Paxos coordinators for a replication framework and compared the observed QoS for NFD-L with the behavior of the framework native leader election algorithm that is not designed to explicitly meet any QoS requirement.

Key-words: Quality of service. Fault tolerance. Distributed algorithms.

Lista de ilustrações

Figura 1 – Organização do algoritmo NFD-S 37
Figura 2 – Organização do algoritmo NFD-U 37
Figura 3 – Organização do algoritmo NFD-E 38
Figura 4 – Organização de componentes em Treplica (VIEIRA; BUZATO, 2008) . 48
Figura 5 – Intervalo de tempo entre enganos 54
Figura 6 – Duração dos enganos 55
Figura 7 – Tempo de detecção de falha 57
Figura 8 – Tempo de detecção da recuperação 58

Lista de algoritmos

Algoritmo 1 Algoritmo NFD-E 39
Algoritmo 2 Algoritmo NFD-L 44
Algoritmo 3 Escalonador 49

Lista de tabelas

Tabela 1 – Percentis repetição entre enganos 53
Tabela 2 – Percentis duração dos enganos 55
Tabela 3 – Percentis tempo de detecção de falha 57
Tabela 4 – Percentis tempo de detecção de recuperação 58
Tabela 5 – Interrupções longas geradas pela *JVM* 59
Tabela 6 – Interrupções longas geradas pela *JVM* após ajuste 60

Lista de abreviaturas e siglas

QoS	*Quality of Service*, Qualidade de Serviço
NFD-S	*New Failure Detector with Synchronized Clocks*
NFD-U	*New Failure Detector with Unsynchronized Clocks*
NFD-E	*New Failure Detector with Estimated Message Arrival*
NFD-L	*New Failure Detector as a Leader Election Algorithm*
rtt	*round-trip time*, tempo necessário para que uma mensagem seja enviada de um processo a outro e retorne ao primeiro processo emissor
NTP	*Network Time Protocol* Protocolo para sincronização de relógios em uma arquitetura cliente-servidor
MTTF	*Mean time to fail*, tempo médio até a falha
MTBF	*Mean time between failures*, tempo médio entre falhas
MTTR	*Mean time to recovery*, tempo médio até a recuperação

Lista de símbolos

Π	Conjunto de processos
Λ	Conjunto de enlaces
Δ	Notação para diferença entre tempos
Ω	Algoritmo de eleição de líder de precisão e acordo futuros
σ	Instante de tempo local para envio de mensagem
η	Intervalo de tempo entre o envio de mensagens
τ	*freshness point*, o instante de tempo máximo para a chegada de uma mensagem
δ	Margem de seguraça para chegada de mensagens com processos com relógios sincronizados
α	Margem de segurança para chegada de mensagens com processos sem relógios sincronizados
ℓ	O maior rótulo de alguma mensagem já enviada
T_D	Tempo de detecção de falha
T_{MR}	Intervalo entre enganos
T_M	Duração de engano
P_A	Probabilidade de precisão de consulta, probabilidade do detector de falhas estar correto
T_{DR}	Tempo de detecção de recuperação
R_{DF}	Proporção de detecção de falhas
X_L	A maior quantidade de mensagens perdidas em sequência observada
EA	Estimativa para o tempo de chegada de mensagens
$E(D)$	Média do atraso das mensagens
$V(D)$	Variância do atraso de mensagens
p_l	Taxa de perda de mensagens

Sumário

Introdução 27

1 FALHAS E DETECTORES DE FALHAS 29
1.1 Abstrações de um Sistema Distribuído 29
1.1.1 Processos e Falhas 30
1.1.1.1 Colapso 30
1.1.1.2 Omissão 30
1.1.1.3 Colapso com Recuperação 31
1.1.2 Enlaces e Mensagens 31
1.1.2.1 Enlace Perda Justa 31
1.1.2.2 Canais Perfeitos 32
1.1.3 Sincronia 32
1.1.4 Composição do Modelo Computacional 32
1.2 Detectores de Falhas 33
1.2.1 Eleição de Líder 34
1.3 Detectores de Falha com Qualidade de Serviço 34
1.3.1 Modelo *falha-e-pára silencioso* 35
1.3.1.1 O Algoritmo NFD-S 35
1.3.1.2 O Algoritmo NFD-U 36
1.3.1.3 O Algoritmo NFD-E 37
1.3.2 Modelo *falha-e-recuperação* 38
1.3.3 Trabalhos Relacionados 40

2 O ALGORITMO NFD-L 43
2.1 Visão Geral 43
2.2 Eleição de Líder 45
2.3 Lidando com Falha-e-Recuperação 45
2.4 Integrando NFD-L e Treplica 47
2.4.1 Treplica 47
2.4.2 Escalonador 48

3 VALIDAÇÃO EXPERIMENTAL 51
3.1 Configuração do Ambiente 51
3.1.1 Ambiente de execução 51
3.1.2 Configuração de NFD-L 51
3.1.3 Organização dos Experimentos 52

3.2 Análise da Precisão do Detector de Falhas 53
3.3 Análise da Velocidade do Detector de Falhas 56
3.4 Observações sobre a latência da aplicação e seu impacto na QoS . 59

Conclusão 61

Referências 63

Introdução

Sistemas distribuídos são um agrupamento coordenado de processos que se comunicam através de enlaces e cooperam para atingir um objetivo comum. Um destes objetivos é prover tolerância a falhas através de redundância e replicação, o sistema pode continuar provendo certo poder computacional a outras aplicações e fornecer persistência de dados mesmo na presença de um subconjunto de processos e enlaces. A base deste princípio é utilizar *software* tolerante a falhas sob *hardware* propenso a falhas.

Os processos e enlaces podem se comportar de modo *assíncrono*, sem qualquer limitação para o tempo de propagação das mensagens e para o tempo necessário para se realizar o processamento. Esta característica torna difícil determinar se um processo falhou ou apenas está demorando para responder às solicitações dos outros processos. Para contornar esta dificuldade, Chandra e Toueg (1996) apresentaram a noção de detectores de falhas *não confiáveis*. Estes detectores de falhas são responsáveis por apontar processos potencialmente defeituosos dentre o conjunto de processos participantes do sistema. Sua *não* confiabilidade deriva do fato de não existir limitação para a quantidade de enganos que podem cometer ao indicar um processo como defeituoso. Além disso sua precisão é apenas futura; o "palpite" sobre o estado de um processo só é esperado estar correto em algum instante de tempo futuro quando o sistema distribuído se comporte de modo *síncrono*. Neste instante futuro, haverá um limite de tempo factível para que as mensagens sejam entregues, possibilitando a determinação dos processos defeituosos.

Considerando a assincronia de um sistema distribuído, o detector de falhas é o componente central que abstrai o conceito de *tempo* no projeto de algoritmos distribuídos tolerantes a falhas (MARTíN; LARREA; JIMéNEZ, 2009; LAMPORT, 1998). Embora a não confiabilidade do detector de falhas não comprometa a correção destes algoritmos distribuídos (GUERRAOUI, 2000), ela pode ter efeitos negativos em seu desempenho. Paxos (LAMPORT, 1998) é um algoritmo distribuído para a solução do problema de consenso (BARBORAK; DAHBURA; MALEK, 1993) que utiliza os serviços de um detector de falhas para eleger um processo como coordenador. Este coordenador irá conduzir o protocolo, sequenciando e entregando mensagens ordenadamente. Entretanto, o algoritmo não pode progredir se não houver um coordenador eleito. Logo, o desempenho de Paxos depende da velocidade com que o detector de falhas substitui um coordenador defeituoso e de quão bem o detector de falhas evita acusar como defeituoso um coordenador correto.

Como modo de limitar este comportamento eventual dos detectores de falhas, em Chen, Toueg e Aguilera (2002) os autores propuseram métricas para quantificar o seu comportamento. Além disso, definiram o conceito de detectores de falhas com qualidade de

serviço (QoS), propondo um conjunto de algoritmos de detecção de falhas que podem ser configurados para atender requisitos da aplicação cliente, considerando o comportamento probabilístico da rede. Para ambientes assíncronos, o algoritmo apresentado depende de um componente *preditor* para estimar o tempo de chegada das mensagens. Esta característica foi extensivamente estudada em Nunes e Jansch-Porto (2004). Uma das limitações do trabalho de Chen, Toueg e Aguilera (2002) é o suporte a um modelo específico de falha de processos denominada *falha-e-para*, onde os processos defeituosos não podem voltar a operar. Este modelo embora seja factível, é incompatível com algoritmos de tolerância à falha como Paxos.

Em Ma, Hillston e Anderson (2010), os autores tentaram ampliar o modelo proposto por Chen, Toueg e Aguilera (2002) para contemplar processos que podem voltar a operar depois de uma falha (*falha-e-recuperação*). Este modelo introduziu novas métricas de QoS e conceitos de confiabilidade de processos. Entretanto, este modelo funciona apenas em ambientes síncronos e o comportamento experimental deste algoritmo não atende a alguns dos requisitos de QoS propostos pelos autores. Além disso, não há trabalhos que analisem a QoS de algoritmos de eleição de líder. Algoritmos de eleição de líder são algoritmos de detecção de falha que não se preocupam em apontar os processos defeituosos, mas sim em apontar um único processo distinto como correto. Estes algoritmos são estudados e utilizados largamente na prática (CHANDRA; HADZILACOS; TOUEG, 1996; MALKHI; OPREA; ZHOU, 2005; MARTíN; LARREA; JIMéNEZ, 2009), mas não há trabalhos que modelem o comportamento da eleição de líder com base em requisitos de QoS.

Neste trabalho, nós apresentamos um algoritmo de eleição de líder, NFD-L, que segue o modelo de QoS apresentado em Chen, Toueg e Aguilera (2002). Este algoritmo possui como característica a operação em ambientes *falha-e-recuperação* parcialmente síncronos. Para validar o atendimento às métricas de QoS definidas por Chen, Toueg e Aguilera (2002), inserimos nosso algoritmo em uma aplicação de replicação que utiliza Paxos e verificamos seu comportamento em um ambiente real com alta carga de processamento. Comparamos o seu comportamento com um algoritmo de eleição de líder que não é projetado para atender qualquer requisito de QoS e constatamos a diferença comportamental entre ambos. NFD-L foi capaz de se comportar de modo satisfatório em relação ao atendimento da QoS requisitada em nossos experimentos.

Este documento está dividido em três capítulos. O Capítulo 1 introduz conceitos básicos da teoria de Sistemas Distribuídos utilizados nos capítulos posteriores. O Capítulo 2 apresenta o algoritmo proposto e os mecanismos criados para suportar o modelo falha-e-recuperação, buscando a não violação dos requisitos de QoS estabelecidos por Chen, Toueg e Aguilera (2002). Por fim, o Capítulo 3 apresenta os resultados experimentais obtidos e discute o atendimento às métricas de QoS por parte do algoritmo

NFD-L.

1 Falhas e Detectores de Falhas

Neste capítulo serão apresentados conceitos fundamentais em que se baseia este trabalho. As abstrações de processos, enlaces e sistema distribuído serão apresentadas na Seção 1.1. O conceito de modelo computacional é apresentado na mesma seção como uma combinação particular de processos e enlaces, configurando um ambiente com características específicas. A Seção 1.2 apresentará o conceito de detectores de falhas e algumas das características que viabilizam o seu uso por algoritmos que priorizam tolerância a falhas. Por fim, este capítulo é encerrado ao apresentar o conceito de detectores de falhas com qualidade de serviço (QoS), definidos originalmente por (CHEN; TOUEG; AGUILERA, 2002).

1.1 Abstrações de um Sistema Distribuído

Um sistema distribuído S é composto por um conjunto $\Pi = \{p_1, ..., p_n\}$ de processos e um conjunto Λ contendo k enlaces, onde $n \geq 2, k \in \mathbb{N}$. O objetivo de um sistema distribuído é a realização de alguma tarefa que se beneficie da cooperação entre vários processos. A comunicação entre os processos acontece pelo envio e recebimento de mensagens, transmitidas através dos enlaces.

Existe um autômato em cada processo que define as instruções a serem executadas localmente. Este autômato também define os padrões para envio e recebimento de mensagens, de modo que todos os processos conheçam as instruções a serem executadas e o formato válido para as mensagens. A execução distribuída deste autômato em todos os processos do sistema é o que neste trabalho se refere por *algoritmo distribuído*.

Sistemas distribuídos implementados na prática são afetados diretamente pelas características apresentadas por seus enlaces e processos. Os processos podem executar em uma arquitetura variada de *software* e *hardware*, o que pode gerar comportamentos diferentes em cada processo devido a fatores difíceis de serem controlados como o escalonamento do sistema operacional, quantidade e velocidade da memória disponível, *drivers* de dispositivos, falhas, entre outros. De modo similar, a implementação dos enlaces utiliza tecnologias variadas como meio de propagação, tais como fios de cobre, fibras ópticas e propagação sem fio, cada qual variando em latência e vazão da transmissão de dados. Para poder descrever a grande diversidade de características apresentadas por processos e enlaces, a literatura de sistema distribuídos desenvolveu ao longo do tempo uma rica quantidade de *abstrações*, capazes de capturar as características essenciais destas entidades.

Para este trabalho, descreveremos as abstrações de um sistema distribuído uti-

lizando as definições encontradas em (CACHIN; GUERRAOUI; RODRIGUES, 2011), por apresentarem definição e notação consistentes.

1.1.1 Processos e Falhas

Processos são entidades que executam computações. As computações consistem em seguir as instruções definidas pelo algoritmo distribuído. Em implementações de sistemas distribuídos um processo pode corresponder a uma única máquina (*host*), a um programa em execução, a *threads* de uma mesma aplicação ou qualquer outra entidade que efetue processamento. Entretanto neste trabalho um processo corresponderá a uma única máquina (*host*).

O critério que diferencia as abstrações de processo são os tipos de *falha* que os processos podem apresentar. Uma falha ocorre quando um processo não segue as instruções definidas pelo algoritmo distribuído. Processos que apresentam falha durante a execução do algoritmo distribuído são denominados *defeituosos* e seu oposto são os processos *corretos*. Os tipos de falha abordados neste trabalho são *colapso*, *omissão* e *colapso com recuperação*.

1.1.1.1 Colapso

O colapso constitui a falha mais básica de um processo, a completa suspensão de sua execução. Um processo que sofre colapso não executa mais nenhuma instrução e não pode receber ou enviar mensagens. Em um algoritmo distribuído que supõe que cada processo envie periodicamente mensagens aos outros processos, o colapso é identificado pela ausência de mensagens enviadas pelo processo defeituoso. Nesta abstração, um processo defeituoso é aquele que sofre colapso durante a execução do algoritmo distribuído.

1.1.1.2 Omissão

Algumas vezes o processo será capaz de executar os passos definidos em seu algoritmo, mas será incapaz de executar algumas instruções, ou as executará com um perceptível atraso. O processo não sofre colapso, mas não irá realizar as tarefas para as quais foi projetado. Este comportamento pode ser causado por congestionamento nos enlaces ou escassez momentânea de recursos computacionais alocados ao processo. Esta falha pode ser percebida pela ausência ou atraso no recebimento de mensagens.

Os processos defeituosos desta abstração são aqueles que sofrem uma omissão permanente, semelhante a um colapso, ou que mudam seu comportamento infinitamente em ciclos de correta execução e omissão.

1.1.1.3 Colapso com Recuperação

No colapso com recuperação um processo que sofre colapso *pode* retomar sua execução após o acionamento de algum mecanismo externo de recuperação. Neste caso, seu comportamento no período entre o colapso até a recuperação seria semelhante à omissão. Uma vez ocorrido o colapso, o processo perde a informação em memória volátil e preserva apenas os dados gravados em memória persistente.

Neste modelo os processos defeituosos são aqueles que não se recuperam após um colapso ou que se comportam infinitamente em um ciclo de colapso e recuperação.

1.1.2 Enlaces e Mensagens

Os enlaces são abstrações que representam vias de comunicação entre os processos. Neste nível de abstração as implementações físicas que correspondem aos meios de propagação e elementos de rede (e por consequência qualquer *software* embarcado nestes equipamentos) são considerados partes do enlace.

A unidade elementar de transmissão de informação entre os processos é a mensagem. Cada mensagem m é enviada de um processo a outro e é identificada através de um identificador único i. Um processo p pode enviar repetidamente a mesma mensagem m_i a outro processo q sem infringir a unicidade das mensagens.

As abstrações de enlace aqui apresentadas não capturam características como tempo de propagação e vazão dos dados transmitidos, mas sim o comportamento dos enlaces em relação à perda e a probabilidade da entrega de mensagens. Os enlaces podem perder mensagens, mas não as geram de modo espontâneo; isto é, uma mensagem recebida por um processo, obrigatoriamente foi enviada por outro. Os tipos de enlace considerados neste trabalho são *enlaces perda justa* e *canais perfeitos*.

1.1.2.1 Enlace Perda Justa

A abstração de enlace perda justa considera que mesmo na presença de perda de mensagens, um processo correto que infinitamente envia uma mensagem a outro processo correto, mais cedo ou mais tarde obterá êxito na entrega desta mensagem. Em outras palavras, a suposição é que mesmo com uma taxa de perda de mensagens arbitrária, a probabilidade da mensagem não ser entregue nunca é igual a zero.

Na prática esta propriedade pode ser observada mesmo em situações severas de problemas de comunicação. Equipamentos defeituosos e enlaces danificados são eventualmente reparados e a comunicação torna-se novamente possível.

1.1.2.2 Canais Perfeitos

Canais perfeitos garantem que uma mensagem enviada por um processo correto a outro processo correto será entregue sem duplicidade. O processo emissor no entanto não precisa se preocupar com a retransmissão da mensagem como quando utiliza enlaces do tipo perda justa, uma vez que o enlace garante a entrega futura sem duplicidade.

É possível utilizar um enlace perda justa para implementar um outro enlace com as propriedades de canais perfeitos ao se utilizar um mecanismo de retransmissão e controle de duplicidade de mensagens.

1.1.3 Sincronia

Sistemas distribuídos síncronos são sistemas onde enlaces e processos operam dentro de limitações de tempo bem conhecidas. Neste tipo de sistema os enlaces sempre irão trafegar as mensagens enviadas com um tempo de propagação menor a um limite fixado. Os processos por sua vez tem acesso a um relógio global e irão efetuar suas computações dentro de um segundo limite de tempo também fixado. Por computação entende-se que um processo irá (i) tomar conhecimento de quaisquer mensagens a ele enviadas através dos enlaces, (ii) executar as devidas instruções relativas ao seu autômato local e em seguida, (iii) enviar quaisquer mensagens necessárias aos demais processos.

Sistemas assíncronos, por outro lado, não fazem nenhuma suposição sobre tempo. Não existe neste modelo uma limitação de tempo para execução das computações e propagação de mensagens. Não há nenhuma suposição que os processos tenham acesso a algum relógio global. Entretanto, ainda assim é possível criar uma ordenação parcial de eventos utilizando o conceito de relógios lógicos (LAMPORT, 1978).

Embora as implementações de sistemas distribuídos comportem-se boa parte do tempo de maneira síncrona, existem períodos de tempo onde o sistema pode se comportar de modo assíncrono. Tal comportamento pode acontecer por problemas momentâneos como congestionamento de enlaces ou falta de recursos computacionais alocados aos processos. Por este motivo, boa parte dos sistemas distribuídos são projetados como sistemas *parcialmente síncronos*. Este modelo híbrido supõe que enlaces e processos podem se comportar de modo assíncrono, mas que haverá um instante de tempo após o qual o sistema se comportará de modo síncrono, possibilitando a execução do algoritmo distribuído.

1.1.4 Composição do Modelo Computacional

Um modelo computacional é uma combinação particular de uma abstração de enlaces, processos e sincronia que descreve o comportamento de um sistema distribuído como um todo. Neste trabalho, serão considerados dois modelos computacionais, definidos de acordo com Cachin, Guerraoui e Rodrigues (2011):

Falha-e-pára ruidoso : Composto por enlaces do tipo canais perfeitos e processos no modelo falha por colapso, operando de modo parcialmente síncrono.

Falha-e-recuperação : Composto por enlaces do tipo perda justa e processos no modelo falha com recuperação, operando de modo parcialmente síncrono.

1.2 Detectores de Falhas

Detectores de falhas são mecanismos de um sistema distribuído cuja função é prover uma estimativa sobre quais processos deste sistema estão corretos em um determinado momento. A definição exata de estado correto de um processo depende do modelo computacional adotado (ver Seção 1.1.1), mas de modo geral caracteriza um processo que opera sem desviar-se das especificações definidas por seu algoritmo distribuído.

Para melhor ilustrar os desafios da construção de um detector de falhas, apresentaremos uma possível implementação idealizada: Um processo q que necessite saber o estado de outro processo p consulta o seu detector de falhas local e obtém como resposta uma indicação sobre o estado do processo de interesse. O detector de falhas estima o estado dos outros processos através do monitoramento de mensagens de *heartbeat*: o processo p envia mensagens de *heartbeat* para o processo q em intervalos de Δ unidades de tempo. Quando o processo q recebe algum *hearbeat* de p, o detector de falhas em q confia que p está correto. Após isso, o processo q espera pela próxima mensagem de *heartbeat* oriunda de p por um período máximo de tempo $\mathrm{T} > \Delta$. Quando um *heartbeat* de p não é recebido por q dentro do intervalo T, ocorre um *timeout* e o detector de falhas em q suspeitará que o processo p falhou.

Em sistemas assíncronos onde processos podem falhar este detector idealizado não funciona corretamente, devido à impossibilidade de se determinar se um processo realmente falhou ou se está apenas levando mais tempo que o habitual para enviar mensagens (FISCHER; LYNCH; PATERSON, 1985). Como exemplo, não é possível para q determinar se p falhou ou se sua mensagem está a caminho. Por este motivo um processo indicado como falho é dito *suspeito*. De modo análogo, um processo indicado como correto por um detector de falhas é dito como um processo *confiável*.

Ao fornecer este serviço aos processos do sistema distribuído, o detector de falhas atua como uma camada de abstração de tempo, encapsulando e interpretando eventos dependentes do modelo de sincronia do sistema. Ao abstrair a noção de passagem de tempo a qualidade das indicações do detector de falhas se torna variável. Este grau de variação é descrito por duas propriedades: *completude*, que caracteriza como o detector de falhas se comporta em relação a processos que falharam e *precisão*, que caracteriza seu comportamento em relação a divergências entre a indicação e o estado real de um processo. Neste trabalho consideramos apenas detectores de falhas que funcionam em um

ambiente parcialmente síncrono e que atendam as propriedades de um detector de falhas $\Diamond\mathscr{W}$ (CHANDRA; TOUEG, 1996):

Completude : Existe um tempo após o qual todo processo que falha é permanentemente suspeito por algum processo correto.

Precisão : Existe um tempo após o qual algum processo correto nunca é suspeito por todo processo correto.

1.2.1 Eleição de Líder

Um algoritmo de eleição de líder é um detector de falhas que indica *um único* processo confiável. Um algoritmo de eleição de líder ainda se comporta como um detector de falhas não confiável. Até que o sistema se comporte de modo síncrono é possível que o detector de falhas cometa um número arbitrário de enganos, fazendo com que processos distintos confiem em líderes distintos. Quando o sistema distribuído se comporta por tempo suficiente de modo síncrono e todos os processos corretos confiam uns nos outros, a utilização de um critério comum para escolha de um único processo deste conjunto possibilita que todos os processos escolham o mesmo processo, que pode atuar como um líder.

Para descrever o *acordo* atingido quando os processos concordam sobre a identidade de um líder, é necessário a definição de uma propriedade adicional. Um detector de falhas é denominado um algoritmo de eleição de líder Ω (CHANDRA; HADZILACOS; TOUEG, 1996) quando atende às seguintes propriedades (CACHIN; GUERRAOUI; RODRIGUES, 2011):

Precisão Futura : Existe um tempo após o qual todo processo correto confia em algum processo correto.

Acordo Futuro : Existe um tempo após o qual não há processos corretos que confiam em processos corretos distintos.

1.3 Detectores de Falha com Qualidade de Serviço

Ainda que seja possível criar aplicações baseadas em detectores de falhas não confiáveis (MALKHI; OPREA; ZHOU, 2005), é interessante poder quantificar e limitar o comportamento destes detectores através de métricas objetivas, fornecendo às aplicações que utilizam estes serviços certas garantias sobre a qualidade de serviço (QoS) prestada.

1.3.1 Modelo *falha-e-pára silencioso*

Em um trabalho seminal, Chen, Toueg e Aguilera (2002) definiram um conjunto de métricas que descrevem a QoS de um detector de falhas, capazes de quantificar sua *velocidade* e *precisão*. A velocidade de um detector de falhas se refere a quão rápido o detector percebe uma falha. A precisão a quão bem o detector evita cometer enganos. As métricas de QoS são:

Tempo de detecção de falha (T_D) : Esta métrica é calculada pelo tempo que uma falha legítima leva para ser percebida pelo detector de falhas.

Intervalo entre enganos (T_{MR}) : Esta métrica é obtida através da média do intervalo de tempo entre os enganos cometidos pelo detector de falhas.

Duração de engano (T_M) : Esta métrica quantifica o tempo médio que o detector de falhas leva para corrigir um engano cometido. É a média do tempo decorrido para a correção de falsas suspeitas cometidas pelo detector de falhas.

Essas métricas podem expressar requisitos de qualidade de serviço, representados por uma tupla (T_D^U, T_{MR}^L, T_M^U) de números positivos, onde T_D^U é um limite superior para o tempo de detecção de uma falha, T_{MR}^L é um limite inferior para o tempo de repetição de um engano cometido pelo detector de falhas e T_M^U é um limite superior para o tempo de duração deste engano.

Junto com as métricas de qualidade de serviço, Chen, Toueg e Aguilera (2002) apresentaram uma família de algoritmos para detecção de falhas que atendem essas métricas em modelos computacionais diferentes de funcionamento de um sistema distribuído:

- *New Failure Detector with Synchronized Clocks (NFD-S)*
- *New Failure Detector with Unsynchronized Clocks (NFD-U)*
- *New Failure Detector with Estimated Message Arrival (NFD-E).*

Ao longo desta seção, vamos apresentar cada um dos algoritmos listados, desde o detector de falhas para o modelo síncrono NFD-S até o detector de interesse para este trabalho, o detector de falhas assíncrono NFD-E.

1.3.1.1 O Algoritmo NFD-S

Vamos apresentar o princípio de funcionamento do algoritmo síncrono NFD-S. Este algoritmo é muito semelhante ao algoritmo idealizado na Seção 1.2. Seu funcionamento se dá pela troca de *heartbeats* rotulados de modo crescente, que são utilizados para determinar o estado (suspeito ou confiável) dos remetentes das mensagens.

Vamos supor um cenário com um sistema distribuído composto por dois processos, p e q, que trabalham com relógios sincronizados e sem flutuações de tempo. Esta suposição requer que em qualquer instante de tempo observado, os relógios de p e q exibam exatamente a mesma marcação de tempo um do outro, e diferente de uma marcação observada em um instante anterior. Em cada um destes processos também existe um detector de falhas NFD-S que monitora o funcionamento do outro processo. A cada η unidades de tempo, o processo p envia ao processo q uma mensagem de *heartbeat* $m_i, i \in \mathbb{N}$. Formalmente, o instante de envio σ_i de m_i é definido como:

$$\sigma_i = i.\eta$$

Em virtude da sincronia de relógios, q "conhece" cada um destes instantes de envio e precisa apenas aguardar por um intervalo de tempo δ pela chegada da mensagem enviada por p. Com isso, q pode obter o instante de chegada τ_i para cada mensagem m_i enviada por p:

$$\tau_i = \sigma_i + \delta$$

Para determinar se suspeita ou não que p falhou, o processo q verifica em τ_i se recebeu de p alguma mensagem m_j com $j \geq i$. Caso exista tal mensagem, q não suspeita de p durante todo o período $[\tau_i, \tau_{i+1})$. Caso contrário, q vai suspeitar da falha de p. Cada instante distinto τ_k, onde $k \in \mathbb{N}$, é chamado de *freshness point*.

Os parâmetros δ e η são calculados por um componente do algoritmo denominado *configurador* e estão relacionados aos requisitos de qualidade de serviço e ao comportamento probabilístico da rede. Um segundo componente, *estimador*, é responsável por medir e descrever o comportamento probabilístico da rede, expressado por meio da taxa de perda p_L, média $E(D)$ e variância $V(D)$ dos atrasos das mensagens. A Figura 1 ilustra a interação do detector de falhas com os seus parâmetros de configuração e requisitos de qualidade de serviço.

1.3.1.2 O Algoritmo NFD-U

Sem a garantia de relógios sincronizados em p e q, não é possível determinar cada *freshness point* τ_i com o mesmo procedimento utilizado pelo algoritmo anterior, pois o instante de envio σ_i de m_i em p não é conhecido por q. A falta de sincronia de relógios e o fato dos *heartbeats* serem mensagens unidirecionais tornam o cálculo exato da média $E(D)$ impraticável, mas não impossibilitam o cálculo de p_L e $V(D)$.

Ao invés de utilizar o instante de envio σ_i das mensagens de p, o processo q utiliza uma estimativa EA_i para a chegada das mensagens. De modo similar ao algoritmo anterior,

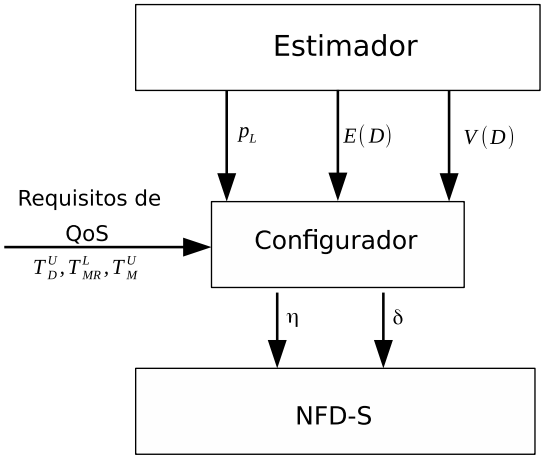

Figura 1: Organização do algoritmo NFD-S

é necessário deslocar a estimativa EA_i de cada mensagem por uma margem de segurança α, dependente do comportamento probabilístico da rede - calculado agora apenas em termos de p_L e $V(D)$ - e dos requisitos de QoS. O cálculo dos novos *freshness points* é:

$$\tau_i = EA_i + \alpha$$

O limite superior de tempo de detecção de uma falha T_D^U é substituído por um parâmetro similar $T_D^u = T_D^U - E(D)$. Este novo algoritmo está ilustrado na Figura 2. É importante notar que é suposto que cada EA_i é conhecido. O algoritmo seguinte, NFD-E, não faz essa suposição e utiliza as mensagens de *heartbeat* anteriores enviadas por p para estimar EA_i.

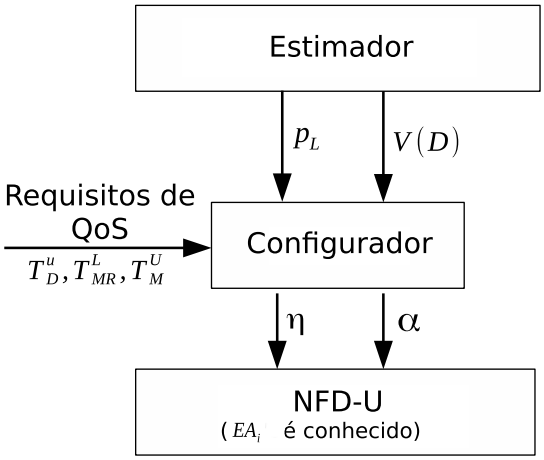

Figura 2: Organização do algoritmo NFD-U

1.3.1.3 O Algoritmo NFD-E

Uma suposição mais realista para ambientes assíncronos é que q não conhece cada EA_i e deve então estimá-los através de um *preditor*. Para estimar cada EA_i, q mantém

em histórico os últimos n *heartbeats* recebidos de p. Para cada mensagem de heartbeat enviado por p, q considera os n heartbeats mais recentes para estimar EA_i. Este algoritmo está ilustrado na Figura 3.

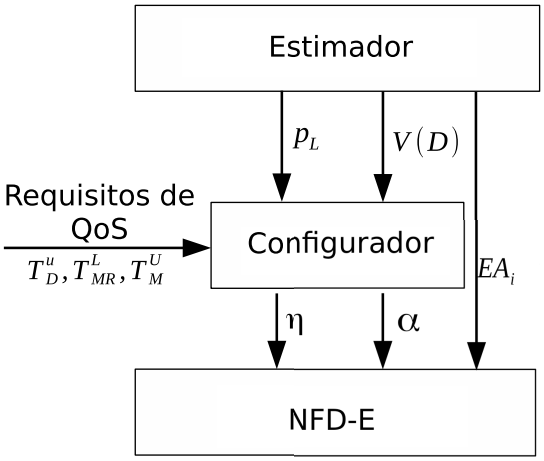

Figura 3: Organização do algoritmo NFD-E

Seja $s_1, \ldots, s_n$ o número de sequência de cada mensagem de *heartbeat* enviada por p, ℓ o maior número de sequência dentre estas mensagens e $A'_1, \ldots, A'_n$ o tempo de recebimento destas mensagens em relação ao relógio de q, então é possível estimar $EA_{\ell+1}$ do seguinte modo:

$$EA_{\ell+1} \approx \frac{1}{n}\left(\sum_{i=1}^{n} A'_i - \eta s_i\right) + (\ell+1)\eta$$

Adicionando a cada EA_i a margem de segurança α, obtêm-se τ_i:

$$\tau_i = EA_i + \alpha$$

O algoritmo NFD-E completo, com as rotinas de envio de mensagens, inicialização e detecção de falhas, como definido por seus autores em Chen, Toueg e Aguilera (2002), é apresentado no Algoritmo 1.

Nesta dissertação estamos interessados na utilização do algoritmo NFD-E, por não requerer a existência de relógios sincronizados entre os processos e oferecer um mecanismo preditor para a estimativa do instante de chegada de mensagens.

1.3.2 Modelo *falha-e-recuperação*

Ma, Hillston e Anderson (2010) expandiram o algoritmo NFD-S proposto originalmente por Chen, Toueg e Aguilera (2002) para que fosse possível utilizá-lo no modelo falha-e-recuperação. A necessidade da existência de relógios sincronizados não foi elimi-

algoritmo 1 Algoritmo NFD-E

```
1: procedimento EnviaHeartbeat                    ▷ Rotina exclusiva ao processo p.
2:     para todo i ≥ 1, no instante i · η faça envia heartbeat m_i para q
3:     fim para
4: fim procedimento
5:
6: procedimento Inicialização                     ▷ Rotinas exclusivas ao processo q.
7:     τ_0 ← 0
8:     ℓ ← −1
9: fim procedimento
10:
11: ao ocorrer τ_{ℓ+1} = now() faça
12:     saída ← Suspeito
13: ao ocorrer recebe messagem m_j no instante t faça
14:     se j > ℓ então
15:         ℓ ← j
16:         τ_{ℓ+1} ← EA_{ℓ+1} + α
17:         se t < τ_{ℓ+1} então
18:             saída ← Confiável
19:         fim se
20:     fim se
21:
```

nada, de modo que um mecanismo de sincronização de relógios, como o protocolo NTP, deve estar presente (MA; HILLSTON; ANDERSON, 2010).

O algoritmo expandido possui métricas adicionais de QoS e um conjunto de parâmetros de confiabilidade que descrevem o comportamento do processo monitorado. Os parâmetros de confiabilidade são gerados pela observação do comportamento do processo a ser monitorado e quantificam sua confiabilidade, disponibilidade e consistência.

Pela definição de Ma, Hillston e Anderson (2010) a confiabilidade é a probabilidade de que o processo se mantenha correto até um instante de tempo t; a disponibilidade é a probabilidade de que o processo esteja correto no tempo t e por fim, a consistência é a probabilidade de que o processo se recupere após um colapso ocorrido no instante t. Os parâmetros de confiabilidade são:

Tempo médio até a falha ($MTTF$) : parâmetro que quantifica o tempo médio que o processo executa sem falhar. Está relacionado à confiabilidade do processo monitorado.

Tempo médio entre falhas ($MTBF$) : parâmetro que quantifica o tempo médio entre duas falhas consecutivas. Quando $MTTF$ é dividido por $MTBF$, o quociente está relacionado a disponibilidade do processo monitorado.

Tempo médio até a recuperação ($MTTR$) : este parâmetro quantifica o tempo mé-

dio necessário para a recuperação de um processo. Está relacionado à consistência do processo monitorado.

Além dos parâmetros de confiabilidade que descrevem o comportamento do processo, existem métricas adicionais de QoS para descrever o comportamento do detector de falhas:

Probabilidade de Precisão de Consulta (P_A) : quantifica a probabilidade de que em um instante de tempo arbitrário, quando consultado, o detector de falhas indique corretamente o estado do processo monitorado.

Tempo de Detecção de Recuperação (T_{DR}) : quantifica o tempo necessário para que o detector de falhas perceba a recuperação do processo monitorado.

Proporção de Detecção de Falhas (R_{DF}) : quantifica a proporção de falhas percebidas em relação às falhas reais.

Assim como o algoritmo original NFD-S, este algoritmo necessita de uma etapa de aferição do comportamento probabilístico da rede, caracterizado pela probabilidade de perda p_L, variância do atraso $V(D)$ e tamanho máximo de perda em sequência X_L de mensagens. Os parâmetros p_L e $V(D)$ são os mesmos considerados pelo algoritmo NFD-S. O novo parâmetro, X_L, quantifica o tamanho da maior perda em rajada de mensagens, isto é, a maior sequência de mensagens perdidas observada.

O algoritmo de Ma *et al.* então utiliza (i) as métricas de QoS (P_A, T_{DR}, R_{DF}) e (T_D^u, T_{MR}^L, T_m^U), (ii) a descrição do comportamento probabilístico da rede (p_L, $V(D)$, X_L) e (iii) os parâmetros de confiabilidade ($MTTF$, $MTBF$, $MTTR$) para configurar um detector de falhas. A configuração de um detector de falhas consiste em definir um intervalo de envio de mensagens η e uma margem de segurança α que atenda a todos os requisitos enumerados em (i).

Embora seja uma expansão interessante do trabalho de Chen, Toueg e Aguilera (2002) por incorporar um modelo mais detalhado para o comportamento de enlaces e processos, Ma, Hillston e Anderson (2010) concluem que o algoritmo proposto precisa ser refinado uma vez que existem diferenças entre o modelo analítico de QoS e o comportamento observado experimentalmente para as métricas de T_M, T_{MR} e R_{DF}. Além disso, este algoritmo ainda está limitado a um ambiente com relógios síncronos por ser uma expansão do algoritmo síncrono NFD-S.

1.3.3 Trabalhos Relacionados

Desde que Chen, Toueg e Aguilera (2002) propuseram o conceito de detectores de falhas com qualidade de serviço, outros trabalhos foram realizados para analisar e estudar

a previsibilidade do comportamento destes detectores.

Em Nunes e Jansch-Porto (2004) os autores analisaram a variação do comportamento de detectores de falhas com qualidade de serviço em função do uso de diferentes tipos de margem de segurança e preditores. Um preditor é responsável por analisar o comportamento das mensagens recebidas para estimar o instante de chegada das mensagens futuras. A margem de segurança fornece uma quantidade adicional de tempo que é somada ao instante estimado pelo preditor. Foram analisados três tipos de preditores, *MEAN*, *WINMEAN* e *ARIMA*. Os preditores do tipo MEAN utilizam a média de todas as mensagens recebidas previamente, os do tipo WINMEAN utilizam apenas as últimas n mensagens recebidas e por fim, os preditores do tipo ARIMA utilizam um modelo baseado em séries temporais. As margens de segurança analisadas eram classificadas em duas categorias: fixas e variáveis. As margens de segurança fixas são calculadas previamente com base em alguma suposição sobre o comportamento das mensagens. As margens variáveis são recalculadas dinamicamente baseadas nas mensagens recebidas anteriormente. Nunes e Jansch-Porto (2004) observaram que preditores do tipo ARIMA são mais precisos que preditores do tipo WINMEAN, que por sua vez são mais precisos que preditores do tipo MEAN. Além disso, concluíram que a combinação preditor-margem de segurança é importante e deve ser escolhida com base na velocidade ou precisão desejadas para o detector de falhas: uma margem de segurança variável utilizada com um preditor menos preciso resulta em detectores de falhas mais velozes. Já uma margem de segurança constante utilizada em conjunto com um preditor preciso, resulta em detectores de falhas mais precisos.

Hayashibara et al. (2004) propuseram um detector de falhas que indica um intervalo contínuo para o grau de suspeita sobre o processo observado. Este algoritmo monitora o comportamento das mensagens recebidas para estimar uma distribuição para os instantes de chegada destas mensagens. A estimativa da distribuição das mensagens é então utilizada para calcular o grau de suspeita ϕ. O valor de ϕ pode então ser utilizado por aplicações distintas, representando estados diferentes (suspeito ou correto) para o processo monitorado em cada uma delas. Este detector de falhas se comporta bem com mudanças no comportamento da rede e pode prover serviços para aplicações distintas simultaneamente. Entretanto, como observado em Hayashibara et al. (2004), seu comportamento recai sobre a qualidade da estimativa da distribuição na qual se baseia ϕ e possui uma tendência em priorizar a velocidade sobre a precisão.

Em Sotoma e Madeira (2006) os autores utilizam cadeias de Markov para modelar um configurador que lida com perdas de mensagens em rajada. O cenário de perdas em rajada é comum em redes de longa distância (YAJNIK et al., 1999) e não é tratado de modo efetivo pelos detectores de falhas propostos por Chen, Toueg e Aguilera (2002). O configurador apresentado proporciona uma boa QoS mas requer que a distribuição da

probabilidade dos atrasos das mensagens seja conhecido.

Em Schiper e Toueg (2008) os autores utilizaram o algoritmo NFD-S proposto por Chen, Toueg e Aguilera (2002) para criar dois algoritmos de eleição de líder para o modelo falha-e-recuperação configurado para atender requisitos de QoS. Estes algoritmos controlam a entrada e saída de processos do sistema (*group membership*) e se reconfiguram automaticamente ao executar periodicamente as etapas de medição do comportamento da rede e configuração do detector de falhas existentes em NFD-S. Entretanto o uso do algoritmo NFD-S requer que os processos possuam um relógio sincronizado para determinar a média do atraso das mensagens ($E(D)$) e os processos devem conhecer a priori o instante de chegada EA_i de cada *heartbeat* enviado pelo líder. Entretanto, cada um dos algoritmos apresentados faz suposições distintas sobre os enlaces e deve ser selecionado previamente dependendo da aplicação a ser executada e do tipo de enlace do sistema.

No presente trabalho, nós apresentamos um algoritmo de eleição de líder com diferenças notáveis em relação ao apresentado em Schiper e Toueg (2008). Não realizamos gerenciamento de grupo pois este não é o escopo do presente trabalho. Nós utilizamos o algoritmo NFD-E como base para construção da eleição de líder. Deste modo, nosso algoritmo não requer que os processos sincronizem seus relógios e não requer que os valores de EA_i sejam reconhecidos a priori, pois são estimados em tempo real durante a execução do algoritmo. A suposição do comportamento dos enlaces é a mesma suposta por Chen, Toueg e Aguilera (2002), os enlaces podem perder e atrasar mensagens e não é necessário alterar o comportamento do algoritmo para variações de perda e atraso de mensagens, apenas reconfigurá-lo para refletir o novo comportamento da rede. Nós também tratamos um problema não abordado em (SCHIPER; TOUEG, 2008), a perda do rótulo i das mensagens de *heartbeat* após o colapso do processo líder. Para isso, utilizamos uma única escrita em memória persistente para tratar o colapso de um processo como perda de mensagens (Seção 2.3).

2 O Algoritmo NFD-L

No capítulo anterior apresentamos os fundamentos teóricos necessários para o entendimento de abstrações essenciais como detectores de falhas com qualidade de serviço e eleição de líder. Neste capítulo apresentaremos o algoritmo de eleição de líder NFD-L (*New Failure Detector as a Leader Election Algorithm*) baseado no algoritmo NFD-E proposto por Chen, Toueg e Aguilera (2002). Mostraremos a estratégia empregada para transformar o algoritmo NFD-E em NFD-L. Mostraremos também como o algoritmo NFD-L lida com a recuperação dos processos executando uma única gravação em memória persistente.

Para analisar o comportamento do algoritmo NFD-L (ver Capítulo 3) vamos utilizá-lo em conjunto com a biblioteca de replicação Treplica, baseada no algoritmo Paxos. Descreveremos brevemente o funcionamento de Treplica e o seu algoritmo de eleição de líder atual. Finalmente, descreveremos as alterações realizadas em Treplica para acomodar o NFD-L como o seu novo algoritmo de eleição de líder.

2.1 Visão Geral

NFD-L é um algoritmo de eleição de líder baseado no algoritmo de detecção de falhas NFD-E. Herda dele as etapas de configuração com base em requisitos de QoS e a suposição de assincronia entre os relógios dos processos. Como diferencial, expande seu modelo computacional para falha-e-recuperação e provê um serviço de eleição de líder, indicando um único processo como confiável. O NFD-L está especificado no Algoritmo 2.

Em NFD-L um único processo atua como líder enviando mensagens periódicas de *heartbeat* aos demais processos, que são apenas observadores do líder. Os observadores esperam pela mensagem ou a ocorrência de um *timeout*. Este comportamento é idêntico ao algoritmo NFD-E. Quando um observador percebe um *timeout*, ele suspeitará que o processo líder falhou e irá iniciar a difusão dos próprios *heartbeats* para informar aos demais processos que é o novo líder. Se um segundo processo observador também suspeitar que o líder atual falhou, irá difundir seus *heartbeats* e os dois processos disputarão a liderança seguindo o algoritmo de *bully* (GARCIA-MOLINA, 1982). Durante a disputa de liderança, o processo com mais tempo sem falhas é selecionado, uma estratégia adotada originalmente em Vieira e Buzato (2008).

Em NFD-L um processo confiará em um único líder em dado momento. Em momentos de disputa processos distintos poderão confiar em líderes distintos, mas nenhum observador ou processo disputante confiará em dois líderes simultaneamente. Uma vez agindo como líder, um processo enviará mensagens rotuladas m_i em intervalos de tempo iguais a

algoritmo 2 Algoritmo NFD-L

1: **procedimento** INICIALIZAÇÃO
2: $pid_{líder} \leftarrow \perp$; $uptime_{líder} \leftarrow 0$; $\ell \leftarrow 0$; $\tau_{\ell+1} \leftarrow 0$
3: $uptime_p \leftarrow 0$
4: $recupera(zerotime)$
5: **se** $zerotime = \perp$ **então**
6: $zerotime \leftarrow agora()$
7: $armazena(zerotime)$
8: **fim se**
9: $i \leftarrow \frac{agora() - zerotime}{\eta}$
10: **fim procedimento**
11:
12: **ao ocorrer** $i \cdot \eta \leq agora()$ **faça**
13: **se** $pid_p = pid_{líder}$ **então**
14: envia heartbeat m_i a todos os processos
15: **fim se**
16: $i \leftarrow i + 1$
17: $uptime_p \leftarrow uptime_p + 1$
18:
19: **ao ocorrer** recebimento de mensagem m_j de q **faça**
20: **se** $pid_{líder} = pid_q$ **então**
21: **se** $j > \ell$ e $uptime_q > uptime_{líder}$ e $pid_{líder} \neq pid_p$ **então**
22: $\ell \leftarrow j$
23: $uptime_{líder} \leftarrow uptime_q$
24: $\tau_{\ell+1} \leftarrow EA_{\ell+1} + \alpha$
25: **fim se**
26: **senão**
27: **se** $uptime_q > uptime_{líder}$ ou
$(uptime_q = uptime_{líder}$ e $pid_q > pid_{líder})$ **então** ▷ Desempate
28: $\ell \leftarrow j$
29: $pid_{líder} \leftarrow pid_q$
30: $uptime_{líder} \leftarrow uptime_q$
31: $\tau_{\ell+1} \leftarrow EA_{\ell+1} + \alpha$
32: $output \leftarrow pid_{líder}$
33: **fim se**
34: **fim se**
35:
36: **ao ocorrer** $\tau_{\ell+1} \leq agora()$ e $pid_{líder} \neq pid_p$ **faça**
37: $pid_{líder} \leftarrow pid_p$
38: $uptime_{líder} \leftarrow uptime_p$
39: $output \leftarrow pid_{líder}$

η. Toda a configuração do detector de falhas e estimativa do comportamento probabilístico da rede seguem as definições do algoritmo NFD-E (CHEN; TOUEG; AGUILERA, 2002).

2.2 Eleição de Líder

Ao iniciar sua operação, os processos não sabem da existência ou identidade de um líder. Caso não haja um líder ativo no momento da ativação de um processo p, o processo p terminará sua inicialização sem receber nenhuma mensagem de *heartbeat* de outro processo (Linhas 1 a 10 do Algoritmo 2). Com isso um *timeout* ocorre (Linha 36 do Algoritmo 2) e p se autoproclama o novo líder. Começará então a enviar mensagens rotuladas de *hearbeat* com seu identificador *pid* a cada η unidades de tempo (Linha 12 do Algoritmo 2). O processo p também armazena e envia junto das mensagens um contador, *uptime*, que é acrescido a cada η unidades de tempo e na prática quantifica o tempo que este processo permaneceu em operação (Linha 17 do Algoritmo 2).

Supondo que p tenha sido o primeiro processo a ser iniciado, tomará para si a liderança e continuará enviando *hearbeats*, incrementando o rótulo i das mensagens e seu contador *uptime*. Quando outros processos iniciarem, também não saberão sobre a identidade do líder. É possível que algum processo q inicie e, sem receber algum *heartbeat* de p, comece a agir como líder difundindo seus próprios *heartbeats*. Quando p receber algum *heartbeat* de q, perceberá que outro processo está agindo como líder e irá verificar o contador *uptime* que q inseriu em suas mensagens. Neste caso, o algoritmo de *bully* seleciona como líder o processo com maior *uptime* (Linha 27 do Algoritmo 2). Como o intervalo de envio de mensagens η é o mesmo para todos os processos e p concluiu sua inicialização em um instante de tempo anterior ao de q, $uptime_p > uptime_q$. Com isso, p irá ignorar a liderança de q (Linha 27 do Algoritmo 2). De modo análogo, ao receber um *heartbeat* de p, o processo q perceberá que existe um processo com um *uptime* mais prioritário e cederá a liderança a este processo.

Para casos onde vários processos inicializam simultaneamente e começam a disputar a liderança usando o mesmo *uptime*, o identificador do processo, *pid*, é usado como critério de desempate e garante a existência de um único líder (segunda cláusula, após a quebra de linha 27 do Algoritmo 2).

2.3 Lidando com Falha-e-Recuperação

O algoritmo NFD-E foi projetado para o sistemas onde processos seguem o modelo falha-e-para e não irá funcionar corretamente em um ambiente falha-e-recuperação. Cada mensagem m_i enviada por um processo em NFD-E é rotulada de modo crescente e estes

rótulos são usados para calcular cada *freshness point* τ_i. Supondo que um processo p falhe após um período longo de funcionamento ao enviar a k-ésima mensagem de *hearbeat*. Após a recuperação de p, os demais processos estarão esperando pela mensagem m_{k+1} para reconhecer p novamente como correto. Pelo algoritmo NFD-E, o processo não se lembrará de seu contador e começará a rotulação das mensagens do início (Linha 6 do Algoritmo 1, Seção 1.3). Como p envia mensagens a cada η unidades de tempo, será necessário ao menos $(k+1)\eta$ unidades de tempo para que p seja reconhecido novamente como correto. Se $(k+1)\eta$ for um período de tempo longo, o processo recuperado pode não ser reconhecido como correto até o fim da execução do algoritmo distribuído, potencialmente prejudicando sua execução. Além disso, o instante inicial de envio das mensagens é alterado a cada recuperação, afetando a previsão de chegada de mensagens futuras dada por $EA_{\ell+1}$.

Uma solução natural para este problema consiste na gravação em memória persistente a cada incremento do rótulo i, de modo que o processo recuperado possa ler desta memória o último rótulo i enviado em alguma mensagem. Embora correta, esta é uma solução cara em termos do uso de memória persistente, mais lenta que memória volátil. Outra solução, encontrada em Ma, Hillston e Anderson (2010) consiste no uso de marcações de tempo ao invés de rótulos sequenciais. Os processos enviariam mensagens rotuladas pelos instantes de tempo de acordo com seus relógios locais. Para determinar o quão antiga uma mensagem é, bastaria comparar o tempo de envio da mensagem com o tempo de recebimento de acordo com o relógio local do processo receptor. Entretanto, esta abordagem requer o uso de relógios sincronizados entre processos, uma suposição forte para sistemas no modelo falha-e-recuperação parcialmente síncrono.

Neste trabalho, propomos uma solução que não requer que os relógios dos processos estejam sincronizados. Esta solução utiliza memória persistente para recuperação dos rótulos das mensagens, mas está limitada a uma única escrita. Ela consiste em fazer com que cada processo grave em memória persistente apenas o instante de tempo t (em relação a seu relógio local) de sua primeira execução. A cada recuperação de uma falha, o processo apenas realiza a leitura deste instante de tempo da memória persistente. A diferença Δ entre o instante atual t' e t pode ser usada em conjunto com η para determinar precisamente o rótulo da mensagem atual esperada pelos outros processos. O número de sequência i a ser determinado para a mensagem m_i pode então ser obtido pela Equação 2.1.

$$i = \lfloor \frac{\Delta}{\eta} \rfloor \tag{2.1}$$

Deste modo, é possível recuperar o rótulo correto das mensagens, limitando o acesso a memória persistente a apenas uma única gravação, no início da operação do sistema, e a uma leitura por recuperação de falha.

2.4 Integrando NFD-L e Treplica

Um algoritmo de eleição de líder atua como provedor de serviços para uma segunda aplicação distribuída que necessita que um processo distinto desempenhe uma função específica. Nesta seção descrevemos como integramos o algoritmo NFD-L com a biblioteca de replicação Treplica. Treplica utiliza Paxos como ferramenta de replicação e necessita que um processo distinto atue como coordenador das instâncias de consenso. Esta integração possibilita uma análise experimental interessante por permitir observar como a execução da aplicação e a carga por ela gerada no sistema influenciam o comportamento do algoritmo NFD-L.

2.4.1 Treplica

Treplica (VIEIRA; BUZATO, 2008) é uma biblioteca para replicação que fornece ao desenvolvedor de aplicações distribuídas uma interface de programação simplificada. O desenvolvedor não precisa se preocupar em implementar e gerenciar mecanismos que controlem a persistência e replicação dos dados. Ao invés disso, projeta aplicações capazes de ler e escrever informações em uma fila de estados que é distribuída de modo transparente pela biblioteca para todos os processos participantes do sistema.

Treplica utiliza Paxos (LAMPORT, 1998) para garantir que todos os processos corretos do sistema recebam as mesmas mensagens e que estas mensagens sejam entregues na mesma ordem. Ao lerem o mesmo conjunto de informações proveniente da fila de estados distribuída, os processos do sistema então farão a transição para o mesmo estado de modo determinista (LAMPORT, 1978), replicando o estado de todos os processos.

Em Treplica um único processo é constituído pela associação de vários objetos, pertencentes a Paxos ou exclusivos à implementação. Um objeto denominado **`Router`** direciona eventos do protocolo aos objetos responsáveis por tratá-los. **`Router`** também gera eventos regulares de passagem de tempo, *process tick*, que os outros objetos utilizam para perceber a passagem de tempo no protocolo de consenso. Os objetos então precisam executar seu processamento interno em intervalos de tempo múltiplos de *process tick*, pois é apenas a cada *process tick* que um objeto pode ser acionado a partir do **`Router`**. A Figura 4 ilustra o esquema geral da interação entre os vários componentes de Treplica.

A execução do **`Router`** ocorre de maneira sequencial, isto é, um evento endereçado a algum objeto é completamente processado pelo objeto em questão para que o controle do processamento seja retomado pelo **`Router`**. Esta arquitetura fornece o escalonamento de tarefas necessário sem aumentar a complexidade da implementação de Treplica.

Em Treplica, o algoritmo atual de eleição de líder é implementado no objeto **`Election`**. A cada *process tick* em que é invocado, **`Election`** instrui o líder a enviar uma mensagem de *hearbeat*. Nos demais processos monitorando líder, **`Election`** aguarda

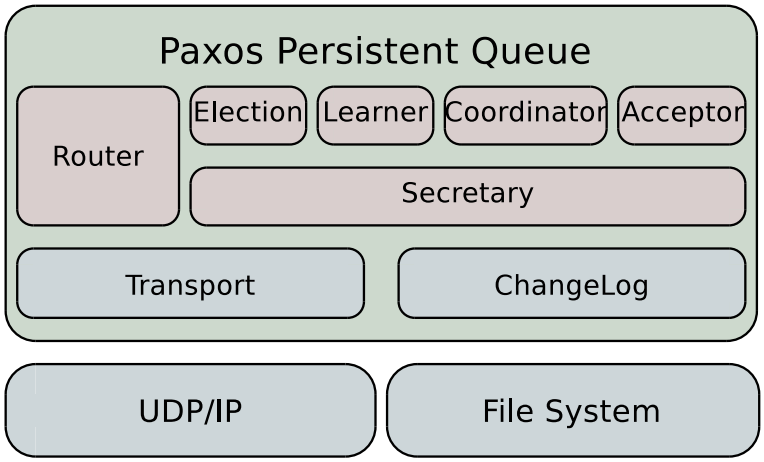

Figura 4: Organização de componentes em Treplica (VIEIRA; BUZATO, 2008)

por até dois *process tick* pela chegada da mensagem de *heartbeat* enviada pelo líder antes de gerar um evento de *timeout*. O processo que gerou o evento de *timeout* irá considerar o antigo líder como defeituoso e inicia então o envio de mensagens de *heartbeat* a todos os processos, se identificando como novo líder.

Processos distintos que iniciam a difusão de seus *heartbeats* em instantes próximos, irão disputar a liderança. Treplica então utiliza o algoritmo de *bully* (GARCIA-MOLINA, 1982) com base nas informações contidas nos *heartbeats* dos processos disputantes, onde o critério utilizado para a escolha é o processo com maior tempo sem falhas. Em casos de empate, o *pid* dos processos é utilizado como critério de desempate. Deste modo, Treplica evita que processos que falhem em demasia sejam reconhecidos como líderes.

2.4.2 Escalonador

A estrutura baseada em intervalos regulares de tempo definidas por *process tick* funciona para o modelo atual de Treplica, onde o detector de falhas não foi implementado levando-se em consideração o atendimento à requisitos de QoS. Esta mesma estrutura entretanto, não atende às especificações em Chen, Toueg e Aguilera (2002): um processo monitor q calcula dinamicamente o instante de chegada de mensagens futuras enviadas pelo líder com base no histórico do recebimento de mensagens anteriores. Logo, para um processo que monitora um processo líder, o instante de chegada de mensagens é variável, baseado unicamente nas estimativas geradas pelo preditor presente no detector de falhas e não relacionado a *process tick*.

Para garantir que as mensagens de *heartbeat* fossem enviadas e recebidas dentro do período de tempo correto, criamos um escalonador circular, onde cada componente possui um instante de tempo associado. Um componente que deseje ser invocado é registrado no escalonador juntamente com um instante de tempo para a invocação (Linha 1 do Algoritmo 3). O escalonador aciona aqueles componentes cujo tempo de invocação foi

atingido (Linha 12 do Algoritmo 3). Após a execução, o componente então informa um novo instante de tempo para ser acionado no futuro (Linha 14 Algoritmo 3). Ao final da execução de todos os componentes, o escalonador atualiza o instante de tempo para a execução do próximo componente (Linha 19 do Algoritmo 3). Este instante de tempo se refere ao próximo acionamento de algum componente, isto é, o próximo componente a ser executado.

algoritmo 3 Escalonador

```
1:  procedimento INICIALIZAÇÃO
2:      T ← ⊥
3:      para todos os componentes escalonáveis c_i faça
4:          T ← T ∪ {(c_i, agora())}
5:      fim para
6:      tProximaExec ← agora()
7:  fim procedimento
8:
9:  ao ocorrer tProximaExec ≤ agora() faça
10:     tProximaExec ← ∞
11:     para todos os componentes registrados (c_i, tempo_i) ∈ T faça
12:         se tempo_i ≤ agora() então
13:             T ← T \ {(c_i, tempo_i)}
14:             tempoFuturo_i ← processar(c_i)
15:             T ← T ∪ {(c_i, tempoFuturo_i)}
16:         senão
17:             tempoFuturo_i ← tempo_i
18:         fim se
19:         tProximaExec ← min(tProximaExec, tempoFuturo_i)
20:     fim para
```

Utilizando a estrutura descrita no Algoritmo 3, registramos os componentes internos de Treplica e o serviço de eleição de líder provido por NFD-L. Deste modo, o processo líder p então informa ao escalonador quando deseja ser acordado para enviar a próxima mensagem de *heartbeat*. De modo análogo, um processo q que monitora as mensagens do líder, informa o instante de tempo em que deseja ser acordado para verificar a ocorrência de um *timeout*.

3 Validação Experimental

Este capítulo apresenta os resultados experimentais observados pela execução de NFD-L como algoritmo de eleição de líder integrado a Treplica. Os experimentos foram divididos em dois conjuntos principais, com a finalidade de analisar a QoS apresentada pelo algoritmo NFD-L em relação à sua precisão e velocidade. A precisão foi analisada medindo-se as métricas de duração de engano (T_M) e intervalo entre enganos (T_{MR}) observadas durante a execução do algoritmo. Analogamente, a velocidade foi analisada medindo-se as métricas de tempo de detecção de falha (T_D) e tempo de detecção de recuperação (T_{DR}). Para cada análise de QoS, apresentamos comparações entre o comportamento observado para o algoritmo NFD-L e o algoritmo de eleição de líder existente em Treplica.

3.1 Configuração do Ambiente

Nesta seção descrevemos a configuração do ambiente utilizado nos experimentos, o procedimento utilizado para obter a configuração (valores para η e α) para NFD-L utilizada nos experimentos e como os experimentos foram executados e analisados.

3.1.1 Ambiente de execução

A execução dos experimentos foi realizada no aglomerado computacional Maritaca, localizado no campus Sorocaba da Universidade Federal de São Carlos. Utilizamos 5 nós do aglomerado, interligados em uma topologia estrela através de enlaces *Gigabit Ethernet*. Os nós são idênticos, cada um composto por um processador *Intel(R) Xeon(R) CPU E5-2665* rodando à $2,4$ GHz, equipados com 16 GB de memória RAM e 512 GB de disco rígido. Os experimentos foram executados em um ambiente *Cent OS Linux 6* 64 *bit*. O detector de falhas foi implementado na linguagem *Java* e executado na versão 8 da *Java Virtual Machine*. A análise estatística do comportamento do detector de falhas foi realizada através do *software R* versão 3.3.2.

3.1.2 Configuração de NFD-L

O procedimento de configuração do algoritmo NFD-L é idêntico ao procedimento utilizado para o algoritmo NFD-E, como definido em Chen, Toueg e Aguilera (2002): é necessário que se observe o comportamento probabilístico da rede e se definam requisitos de QoS para encontrar valores compatíveis para η e α.

Geramos volume de tráfego na rede e carga de processamento com o objetivo de obter medidas realistas do comportamento probabilístico da rede. Esta carga no sistema foi

gerada a partir da execução de uma aplicação sobre Treplica com 5 processos requisitando 1000 operações por segundo cada. Com o sistema sob carga, medimos então a taxa média de perda e variância do tempo de propagação de mensagens trocadas por estes 5 processos. Após 3 observações de 1 hora cada, a taxa média de perda de mensagens ($\overline{p_L}$) e a variância média do tempo de propagação de mensagens ($\overline{V(D)}$) observadas foi $\overline{p_L} = 0,01759$ e $\overline{V(D)} = 25,3356$. Definimos então os requisitos de QoS para NFD-L como tempo de detecção de falha $T_D^u = 1000$ ms, tempo médio de intervalo entre enganos $T_{MR}^L = 3600000$ ms e tempo médio de duração do engano $T_M^U = 1000$ ms. Por fim, este conjunto de requisitos e o comportamento probabilístico observado para a rede resultaram em uma configuração para NFD-L com $\eta = 330$ ms e $\alpha = 770$ ms. Para comparar NFD-L com o algoritmo de eleição de líder existente em Treplica, configuramos ambos os algoritmos para enviar seus *heartbeats* em intervalos de tempo iguais.

3.1.3 Organização dos Experimentos

Os experimentos consistiram em executar uma aplicação de replicação construída usando-se Treplica enquanto o comportamento do processo líder, atuando como coordenador de Paxos, era monitorado pelos outros processos. Utilizamos além do algoritmo NFD-L, o algoritmo de eleição de líder existente em Treplica. Os experimentos foram realizados em 5 máquinas, mapeadas como 5 processos. Um dos processos, após eleito líder pelo algoritmo de eleição de líder, tinha seus *heartbeats* monitorados pelos outros 4 processos que atuavam como seguidores. Esta rotina de monitoramento foi executada 10 vezes, de modo que um total de 40 observações foram coletadas (10 repetições com 4 processos seguidores) para cada métrica de QoS e algoritmo analisados. Durante a execução dos experimentos geramos a mesma carga no sistema utilizada durante a etapa de medição do comportamento probabilístico da rede (Seção 3.1.2).

Para quantificar a diferença de comportamento entre os algoritmos analisados, comparamos NFD-L com o algoritmo de eleição de líder existente em Treplica métrica a métrica através do teste não paramétrico de *Mann-Whitney-Wilcoxon*, uma vez que não foi possível garantir a normalidade dos dados pelo teste de *Shapiro-Wilk*. O teste de *Mann-Whitney-Wilcoxon* analisa dois conjuntos de amostras de uma distribuição qualquer e calcula um p-valor que indica o nível de segurança de que as amostras sejam originárias de populações diferentes. De modo informal, quanto menor o p-valor calculado, maior a chance de que os dois conjuntos de dados representem populações distintas. Para estas análises, consideramos significativos p-valores calculados inferiores a $0,001$.

Por fim, apresentamos os resultados dos experimentos para cada métrica de QoS baseada nos quartis das observações dos processos monitores.

3.2 Análise da Precisão do Detector de Falhas

Estes experimentos foram executados por 1 hora utilizando o detector de falhas tradicional de Treplica e o detector de falhas NFD-L. Não foram injetadas falhas no sistema, de modo que todas as suspeitas de falha caracterizaram um engano cometido pelo algoritmo de eleição de líder. Após o fim da execução dos experimentos, calculamos a média para as métricas de tempo entre enganos (T_{MR}) e tempo para correção de engano (T_M) de cada processo do seguinte modo: seja $t_{m0}, ...t_{mn}$ os instantes de tempo dos enganos cometidos por algum processo e $t_{r0}, ..., t_{rn}$ os instantes de correção destes enganos, então T_{MR} foi calculado como:

$$T_{MR} = \frac{1}{n}\left(\sum_{i=1}^{n}(t_{mi} - t_{mi-1})\right)$$

E T_M foi calculado como:

$$T_M = \frac{1}{n}\left(\sum_{i=0}^{n}(t_{ri} - t_{mi})\right)$$

Com base nas observações de T_{MR} e T_M de cada processo, criamos *boxplots* para analisar cada conjunto de métricas separadamente. A Figura 5 ilustra o comportamento dos detectores de falhas analisados em relação ao intervalo entre enganos (T_{MR}). A linha horizontal tracejada delimita o requisito utilizado para a configuração de NFD-L, $T_{MR}^{L} =$ 3600000 ms, o que significa que o comportamento ideal é caracterizado por uma execução de 1 hora sem enganos. Os valores numéricos dos percentis ilustrados na Figura 5 estão descritos na Tabela 1. O p-valor que indica a diferença entre os resultados obtidos por NFD-L e o algoritmo de eleição de líder de Treplica para T_{MR} foi de $6,721 \cdot 10^{-10}$, um valor significativo (inferior a $0,001$), o que significa que os resultados obtidos por cada algoritmo foram muito diferentes.

Tabela 1: Percentis repetição entre enganos

Eleição de líder	0%	25%	50%	75%	100%
NFD-L	30	3600000	3600000	3600000	3600000
Treplica *ad-hoc*	119143,7	270128,6	1576409,5	1606604,8	1641421,0

É possível observar que o comportamento do detector de falhas NFD-L apresentou algumas variações, caracterizadas pelos *outliers* na região de 30 ms. Este comportamento se deve à sobrecarga do processo que operava como coordenador em 2 das 10 repetições executadas. A sobrecarga gerada pela replicação efetuada por Treplica fez com que o processo líder enviasse as mensagens de *heartbeat* com atrasos, ocasionando uma falsa suspeita nas réplicas que observavam o líder devido a este atraso. Este resultado sugere que a aplicação que utiliza os serviços de um detector de falhas deve ser levada em

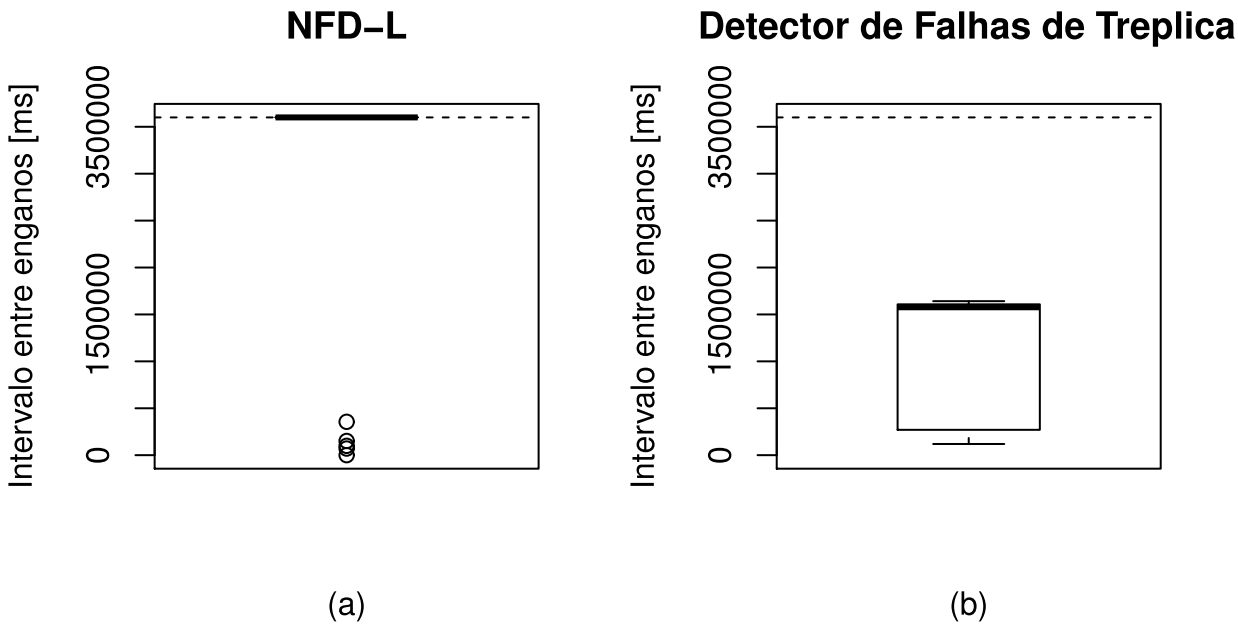

Figura 5: Intervalo de tempo entre enganos

consideração durante o processo de configuração deste detector. Esta é uma observação que pode ser encontrada em (MA; HILLSTON; ANDERSON, 2010), onde os autores tentaram caracterizar o comportamento do processo em termos de métricas de confiabilidade $(MTTF, MTBF, MTTR)$.

Desconsiderando os *outliers* gerados por uma sobrecarga inesperada no processo que atuava como coordenador, é possível observar que o algoritmo NFD-L apresentou um valor para T_{MR}^{L} igual a 3600000 ms. Este valor é calculado obtendo-se a média do intervalo de tempo entre dois enganos consecutivos (CHEN; TOUEG; AGUILERA, 2002). Portanto um valor igual a 3600000 ms significa que o detector de falhas cometeu, em média, 1 engano por hora. Nas amostras onde o detector de falhas não cometeu enganos, este valor foi definido para 3600000, a duração do experimento. O número total de enganos observados nas 40 observações foi 7.

Por outro lado, o detector de falhas utilizado por Treplica apresentou um comportamento mais variado e cometeu ao menos 2 enganos em todas as observações em nosso experimento. A Figura 5 e Tabela 1 demonstram que o detector tradicional cometeu em média 1 engano a cada 1576409.5 ms, isto é, 1 engano a cada 26 minutos.

A métrica de tempo de correção de engano (T_M) também apresentou diferenças entre ambos os algoritmos, o p-valor para resultados obtidos por NFD-L e o algoritmo de eleição de líder de Treplica foi de $1,819 \cdot 10^{-15}$. A Figura 6 ilustra o comportamento demonstrado para os detectores de falhas analisados em relação a métrica de tempo de

correção de engano (T_M). A linha horizontal pontilhada delimita o valor máximo aceito para esta métrica, definida por $T_M^U = 1000$ ms. A Tabela 2 apresenta os valores numéricos dos percentis dos valores observados para T_M.

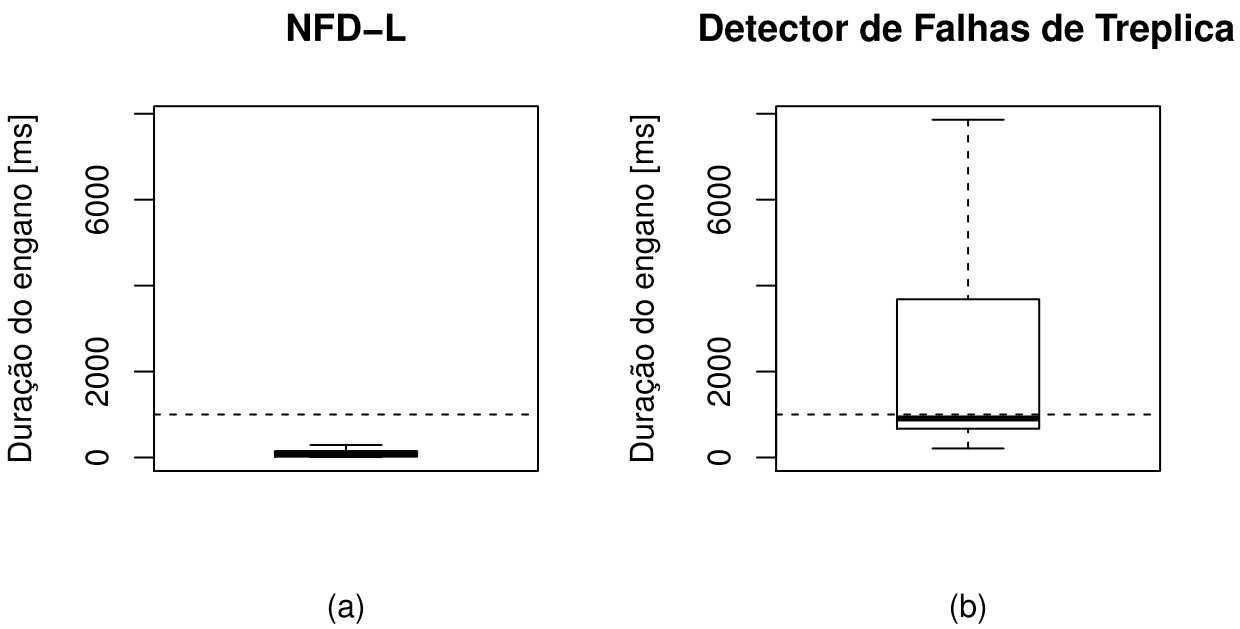

Figura 6: Duração dos enganos

Tabela 2: Percentis duração dos enganos

Eleição de líder	0%	25%	50%	75%	100%
NFD-L	11,000	20,000	98,222	146,030	291,500
Treplica *ad-hoc*	210,7690	681,9375	908,2500	3621,3750	7860,5000

O comportamento do detector de falhas NFD-L implementado foi nitidamente mais uniforme ao corrigir enganos cometidos, com uma amplitude de variação de $280,5$ ms. As correções dos enganos se concentraram em torno da mediana $98,23$ ms e todas as correções estão dentro do limite estabelecido de 1000 ms. O detector de falhas tradicional utilizado por Treplica apresentou uma variação maior, com uma amplitude de $7649,74$ e mediana $908,25$.

Estes resultados demonstram o quão consistente o algoritmo NFD-L pode ser na correção de enganos. O detector de falhas tradicional utilizado por Treplica não apresentou tendências numéricas para a correção de enganos, justificável pelo fato de não ser construído levando-se em consideração algum requisito de qualidade de serviço.

3.3 Análise da Velocidade do Detector de Falhas

A velocidade dos algoritmos de eleição de líder foi medida através da observação das métricas de tempo de detecção de falha (T_D) e tempo de detecção de recuperação (T_{DR}). Para medir T_{DR}, um processo foi escolhido aleatoriamente como líder prioritário, violando as especificações de NFD-L de tal modo que este processo fosse eleito líder após uma recuperação. A definição de um líder prioritário foi realizada atribuindo-se ao processo líder um valor de *uptime* garantidamente mais alto que os demais processos. Injetamos uma falha neste processo e após algum tempo, ativamos sua rotina de recuperação para medir o tempo necessário para que os demais processos percebessem a falha injetada e a eventual recuperação do líder. Estes experimentos foram executados por 300 segundos utilizando o detector de falhas já existente em Treplica e o detector de falhas NFD-L. Após 120 segundos de experimento uma falha era injetada no sistema, causando o colapso do processo que atuava como coordenador. Após 60 segundos da injeção de falha, o processo líder foi recuperado.

Ao final da execução deste experimento, pudemos observar o tempo de detecção da falha (T_D) e tempo de detecção de recuperação (T_{DR}, proposta por Ma, Hillston e Anderson (2010)) apresentado por cada processo. Calculamos T_D e T_{DR} para cada processo do seguinte modo: Seja t_c o instante de tempo do colapso do processo líder e t_d o instante de tempo de detecção desta falha por um processo que monitorava o líder, o tempo de detecção T_D foi obtido por:

$$T_D = t_d - t_c$$

De modo similar, o tempo de detecção de recuperação (T_{DR}) foi calculado como:

$$T_{DR} = t_{dr} - t_r$$

Onde t_r representa o instante de tempo da recuperação do processo líder e t_{dr} é o tempo de detecção desta recuperação por um processo que monitorava o processo líder.

Para calcular a diferença de tempo entre os eventos de colapso e recuperação do processo líder e os instantes de tempo em que estes eventos eram percebidos nos processos monitores, utilizamos o protocolo NTP para sincronizar os relógios locais dos processos. Este procedimento não é necessário para o funcionamento do algoritmo NFD-L e foi utilizado apenas para facilitar o cálculo da diferença de tempo entre os eventos ocorridos nos processos. Entretanto, observamos com o uso do protocolo NTP que o tempo de propagação das mensagens (*round-trip time*) é cerca de $0,250$ ms, mais de 1000 vezes menor que o intervalo η de envio de mensagens. Por este motivo, em nossos experimentos, tomamos como desprezível o tempo de propagação das mensagens.

O resultado da observação do tempo de detecção de falha (T_D) pode ser encontrado na Figura 7 e os valores numéricos dos percentis estão descritos na Tabela 3. O resultado apresentado por ambos os algoritmos para esta métrica apresentou diferenças, como indica o p-valor calculado pelo teste de *Mann-Whitney-Wilcoxon*, igual a $4,158 \cdot 10^{-12}$.

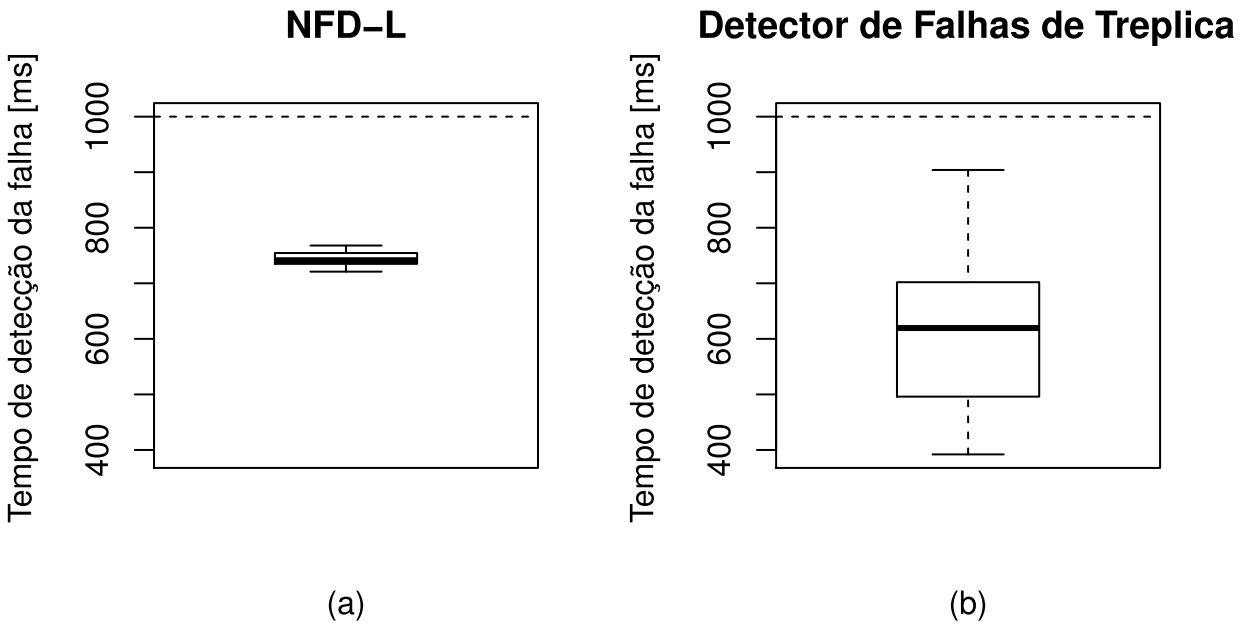

Figura 7: Tempo de detecção de falha

Tabela 3: Percentis tempo de detecção de falha

Eleição de líder	0%	25%	50%	75%	100%
NFD-L	721,00	735,00	741,50	754,25	768,00
Treplica *ad-hoc*	392,0	498,0	619,5	702,0	904,0

Na Figura 7 o valor requisitado $T_D^u = 1000$ está destacado por uma linha pontilhada. Pela representação desta figura e dos valores na Tabela 3 é possível observar que (i) os detectores de falhas NFD-L e o utilizado por Treplica detectaram as falhas injetadas em um tempo menor que 1000 ms e (ii) o detector de falhas NFD-L se mostrou mais consistente por apresentar menor variabilidade no tempo de detecção da falha.

A Figura 8 ilustra o comportamento observado para ambos os detectores de falhas analisados em relação à métrica de tempo de detecção de recuperação (T_{DR}), definida por (MA; HILLSTON; ANDERSON, 2010). Os valores numéricos dos percentis estão descritos na Tabela 4. Uma linha horizontal de referência está demarcando 1000 ms. O p-valor calculado para esta métrica para foi de $2,556 \cdot 10^{-06}$, indicando como em todas as métricas anteriores, diferenças significativas de comportamento entre os algoritmos.

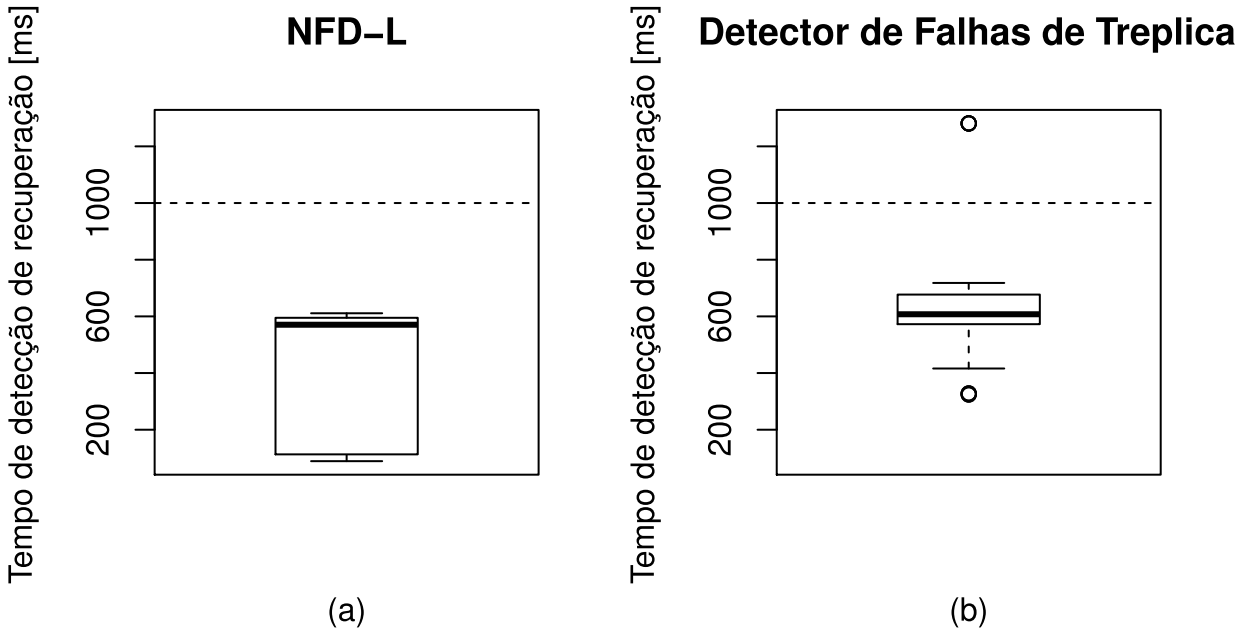

Figura 8: Tempo de detecção da recuperação

Tabela 4: Percentis tempo de detecção de recuperação

Eleição de líder	0%	25%	50%	75%	100%
NFD-L	89,0	113,0	570,5	595,0	611,0
Treplica *ad-hoc*	324,00	572,75	607,50	677,00	1281,00

Embora o detector implementado não seja explicitamente configurado para atender T_{DR}, apresentamos aqui a sua análise por julgarmos importante uma caracterização do comportamento dos detectores de falhas em relação a percepção da recuperação do processo monitorado. Além disso, definimos seu valor de referência (1000 ms) para o mesmo valor adotado pela métrica de duração do engano (T_M), uma vez que quando o processo líder se recupera, os processos monitores permanecem em um estado de engano até que percebam a recuperação.

É possível observar que a variação do detector de falhas tradicional utilizado por Treplica foi menor ao desconsiderarmos os *outliers* existentes, entretanto, os mesmos *outliers* podem indicar que o tempo de detecção de recuperação deste algoritmo é menos previsível. O detector de falhas NFD-L apresentou maior variação de T_{DR}, mas também foi mais rápido para detectar as recuperações (sua detecção mais rápida foi de 89 ms contra 324 do detector tradicional de Treplica). Além disso, o detector de falhas NFD-L apresentou uma tendência de realizar suas detecções em torno de 610 ms, mesmo não sendo projetado para atender um valor específico para T_{DR}.

Em Schiper e Toueg (2008) os autores calculam o tempo para a eleição de um novo líder baseado no tempo em que se detecta o colapso de um coordenador e no tempo necessário para um novo processo se eleger como líder. Não medimos diretamente esta métrica em nosso trabalho, mas é possível adicionar a mediana do tempo de detecção de falha (T_D) à mediana do tempo de detecção de recuperação (T_{DR}) para obter um valor estimado médio para esta métrica. Deste modo, o tempo médio aproximado para eleição de um novo líder para o algoritmo NFD-L nos experimentos realizados, é 1312 ms. De modo análogo, a mesma medida para o algoritmo de eleição de líder utilizado em Treplica, é 1227 ms. Considerando o pior caso observado, o tempo máximo para eleição de um novo líder para NFD-L é 1379 ms e para o algoritmo de eleição de líder de Treplica, 2185 ms.

3.4 Observações sobre a latência da aplicação e seu impacto na QoS

Durante a elaboração destes experimentos, definimos requisitos de QoS iniciais com valores menores dos que os utilizados. Os tempos de detecção de falha (T_D^u) e duração do engano (T_M^U) foram inicialmente definidos para 200 ms, o que resultou em valores de $\eta = 55$ ms e $\alpha = 145$ms. Embora o modelo matemático proposto em Chen, Toueg e Aguilera (2002) tenha gerado uma configuração válida para um detector de falhas com estes requisitos, na prática o detector de falhas não as atendia. Observamos que isto acontecia pois o processo líder era incapaz de enviar mensagens de *heartbeat* com a frequência necessária e os demais processos eventualmente sofriam atrasos de processamento das mensagens recebidas.

Investigamos este comportamento e constatamos que o ambiente de execução Java gerava interrupções na aplicação. Quando uma aplicação fica sem memória disponível, o ambiente de execução da *Java Virtual Machine (JVM)* precisa interromper a aplicação em execução (neste caso, o processo todo), incluindo todas as possíveis *threads* em execução e invocar o *garbage collector* para limpar a memória alocada e alocar mais memória caso necessário. A Tabela 5 contém os valores para as interrupções de coleta de lixo mais longas obtidas a partir de 30 observações de 1 hora em um ambiente com a carga de consenso utilizada nos experimentos.

Tabela 5: Interrupções longas geradas pela *JVM*

Mínima	Máxima	Média	Desvio padrão
1,9080 s	63,3732 s	9,1868 s	14,96 s

Na média, as pausas mais longas observadas duraram 9,18 segundos. Uma vez que um processo não pode enviar ou receber mensagens enquanto está em estado de pausa, o

atendimento aos requisitos de QoS pode ser comprometido se as pausas forem frequentes.

Para minimizar o tempo de interrupção ocasionado pela invocação do *garbage collector*, ajustamos o tamanho do *heap* da *JVM* para que as pausas se tornassem menores e menos frequentes. A Tabela 6 contém os valores das interrupções mais longas causadas pela *JVM* para 30 observações de 1 hora após o ajuste do tamanho e política de alocação do heap. A carga do sistema é a mesma utilizada durante os experimentos.

Tabela 6: Interrupções longas geradas pela *JVM* após ajuste

Mínima	Máxima	Média	Desvio padrão
$0,3263$ s	$10,9138$ s	$0,7861$ s	$1,9434$ s

Após o ajuste da *JVM*, houve uma redução no tempo de pausa dos processos e a média das pausas mais longas foi de $0,7861$ segundos. Com isso, decidimos adotar para os experimentos requisitos de QoS compatíveis com as pausas geradas pelo ambiente de execução. Abandonamos os valores iniciais para $T_D^u = 200$ ms e $T_M^U = 200$ ms e adotamos os valores definitivos utilizados neste trabalho, $T_D^u = 1000$ ms e $T_M^U = 1000$ ms.

Estes resultados sugerem que o comportamento do algoritmo de eleição de líder depende, assim como o comportamento probabilístico da rede, da confiabilidade dos processos do sistema. Observamos que cargas elevadas de processamento e interferências geradas pelo ambiente de execução da aplicação podem interferir diretamente na periodicidade de envio de mensagens e na velocidade com que são processadas. Deste modo, acreditamos que a modelagem da QoS dos algoritmos de eleição de líder e detecção de falhas devam levar em consideração não apenas o modelo computacional do sistema, mas características de confiabilidade dos processos, como argumentado em Ma, Hillston e Anderson (2010).

Conclusão

Neste trabalho nós apresentamos um algoritmo de eleição de líder, NFD-L, modelado a partir do algoritmo de detecção de falhas NFD-E originalmente apresentado por Chen, Toueg e Aguilera (2002). O algoritmo NFD-L foi projetado para suportar um ambiente parcialmente síncrono no modelo falha-e-recuperação e atender a requisitos de QoS da aplicação, considerando o comportamento probabilístico da rede. Além disso a implementação de NFD-L é eficiente ao utilizar a memória persistente para recuperar um estado após uma falha. Esta estratégia mantém corretamente os rótulos das mensagens de *heartbeat* e reduz uma falha de processo a uma falha de mensagens do ponto de vista dos processos que observam o processo líder.

Utilizamos o algoritmo NFD-L em uma aplicação baseada em Paxos com uma alta carga propositalmente inserida para analisar o atendimento aos requisitos de QoS em uma ambiente sob *stress*. Pudemos verificar a diferença nítida comportamental do algoritmo NFD-L em relação ao algoritmo de eleição de líder já existente na aplicação de replicação utilizada. O algoritmo NFD-L apresentou um comportamento menos variado e foi capaz de atender às métricas de QoS, mas a precisão do algoritmo é afetada em momentos de alta carga de processamento. Este comportamento sob carga não é previsto no modelo original de Chen, Toueg e Aguilera (2002) e nossos experimentos sugerem que uma modelagem ideal de QoS deve considerar além do comportamento probabilístico da rede, a confiabilidade dos processos, como observado em Ma, Hillston e Anderson (2010).

Trabalhos futuros baseados na atual implementação de NFD-L podem atacar o problema da redução de precisão do detector de falhas em momentos de elevada carga no sistema. Uma abordagem simplista para a mitigação deste problema é a utilização do estimador como parte integral da aplicação, de modo que a latência de processamento possa ser medida e utilizada pelo configurador para gerar valores adequados de η e α para NFD-L. Deste modo não haveria distinção entre latência originada pela aplicação e latência originada pelos enlaces, deixando a cargo do configurador lidar com atrasos e perdas de mensagens de qualquer natureza.

Outra possibilidade de pesquisa consiste em analisar o comportamento de NFD-L em redes de longa distância. Para a execução dos experimentos utilizamos um aglomerado computacional equipado com enlaces *gigabit ethernet* de baixa latência, mas em redes de longa distância os enlaces podem apresentar características de perda e atraso de mensagens distintas das observadas neste trabalho. Um dos problemas que podem ser enfrentados neste cenário é a perda de mensagens em rajada. Em Sotoma e Madeira (2006) os autores utilizam cadeias de *Markov* para atender os requisitos de QoS de um detector de falhas

na presença deste tipo de comportamento de mensagens, mas a sua aplicação em NFD-L não foi verificada.

Referências

BARBORAK, M.; DAHBURA, A.; MALEK, M. The consensus problem in fault-tolerant computing. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 25, p. 171–220, June 1993. ISSN 0360-0300. Citado na página 27.

CACHIN, C.; GUERRAOUI, R.; RODRIGUES, L. *Introduction to Reliable and Secure Distributed Programming*. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-15259-7. Disponível em: <http://dx.doi.org/10.1007/978-3-642-15260-3>. Citado 3 vezes nas páginas 29, 32 e 34.

CHANDRA, T. D.; HADZILACOS, V.; TOUEG, S. The weakest failure detector for solving consensus. *J. ACM*, ACM, New York, NY, USA, v. 43, n. 4, p. 685–722, 1996. ISSN 0004-5411. Citado 2 vezes nas páginas 28 e 34.

CHANDRA, T. D.; TOUEG, S. Unreliable failure detectors for reliable distributed systems. *J. ACM*, ACM, New York, NY, USA, v. 43, n. 2, p. 225–267, mar. 1996. ISSN 0004-5411. Disponível em: <http://doi.acm.org/10.1145/226643.226647>. Citado 2 vezes nas páginas 27 e 34.

CHEN, W.; TOUEG, S.; AGUILERA, M. On the quality of service of failure detectors. *Computers, IEEE Transactions on*, v. 51, n. 5, p. 561–580, May 2002. ISSN 0018-9340. Citado 17 vezes nas páginas 11, 13, 27, 28, 29, 35, 38, 40, 41, 42, 43, 45, 48, 51, 54, 59 e 61.

FISCHER, M. J.; LYNCH, N. A.; PATERSON, M. S. Impossibility of distributed consensus with one faulty process. *J. ACM*, ACM, New York, NY, USA, v. 32, n. 2, p. 374–382, abr. 1985. ISSN 0004-5411. Disponível em: <http://doi.acm.org/10.1145/3149.214121>. Citado na página 33.

GARCIA-MOLINA, H. Elections in a distributed computing system. *IEEE Trans. Comput.*, IEEE Computer Society, Washington, DC, USA, v. 31, n. 1, p. 48–59, jan. 1982. ISSN 0018-9340. Disponível em: <http://dx.doi.org/10.1109/TC.1982.1675885>. Citado 2 vezes nas páginas 43 e 48.

GUERRAOUI, R. Indulgent algorithms (preliminary version). In: *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*. New York, NY, USA: ACM, 2000. (PODC '00), p. 289–297. ISBN 1-58113-183-6. Disponível em: <http://doi.acm.org/10.1145/343477.343630>. Citado na página 27.

HAYASHIBARA, N. et al. The ϕ accrual failure detector. In: *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, 2004.* [S.l.: s.n.], 2004. p. 66–78. ISSN 1060-9857. Citado na página 41.

LAMPORT, L. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, ACM, New York, NY, USA, v. 21, n. 7, p. 558–565, jul. 1978. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/359545.359563>. Citado 2 vezes nas páginas 32 e 47.

LAMPORT, L. The part-time parliament. *ACM Trans. Comput. Syst.*, ACM Press, New York, NY, USA, v. 16, n. 2, p. 133–169, 1998. ISSN 0734-2071. Citado 2 vezes nas páginas 27 e 47.

MA, T.; HILLSTON, J.; ANDERSON, S. On the quality of service of crash-recovery failure detectors. *IEEE Trans. Dependable Secur. Comput.*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 7, n. 3, p. 271–283, jul. 2010. ISSN 1545-5971. Disponível em: <http://dx.doi.org/10.1109/TDSC.2009.35>. Citado 10 vezes nas páginas 28, 38, 39, 40, 46, 54, 56, 57, 60 e 61.

MALKHI, D.; OPREA, F.; ZHOU, L. Ω meets Paxos: Leader election and stability without eventual timely links. In: *DISC '05: Proceedings of the 19th International Conference on Distributed Computing*. [S.l.]: Springer, 2005. (Lecture Notes in Computer Science, v. 3724), p. 199–213. Citado 2 vezes nas páginas 28 e 34.

MARTíN, C.; LARREA, M.; JIMéNEZ, E. Implementing the omega failure detector in the crash-recovery failure model. *Journal of Computer and System Sciences*, Elsevier, v. 75, n. 3, p. 178–189, 2009. Citado 2 vezes nas páginas 27 e 28.

NUNES, R. C.; JANSCH-PORTO, 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*. Washington, DC, USA: IEEE Computer Society, 2004. (DSN '04), p. 753–. ISBN 0-7695-2052-9. Disponível em: <http://dl.acm.org/citation.cfm?id=1009382.1009791>. Citado 3 vezes nas páginas 28, 40 e 41.

SCHIPER, N.; TOUEG, S. A robust and lightweight stable leader election service for dynamic systems. In: IEEE. *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*. [S.l.], 2008. p. 207–216. Citado 3 vezes nas páginas 41, 42 e 59.

SOTOMA, I.; MADEIRA, E. R. M. *A Markov Model for Providing Quality of Service for Failure Detectors under Message Loss Bursts*. [S.l.], 2006. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.72.5110>. Citado 2 vezes nas páginas 41 e 61.

VIEIRA, G. M. D.; BUZATO, L. E. Treplica: Ubiquitous replication. In: *SBRC '08: Proc. of the 26th Brazilian Symposium on Computer Networks and Distributed Systems*. Rio de Janeiro, Brasil: [s.n.], 2008. Disponível em: <http://www.lbd.dcc.ufmg.br/bdbcomp/servlet/Trabalho?id=7450>. Citado 4 vezes nas páginas 15, 43, 47 e 48.

YAJNIK, M. et al. Measurement and modelling of the temporal dependence in packet loss. In: IEEE. *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. [S.l.], 1999. v. 1, p. 345–352. Citado na página 41.