Adaptação de protocolos distribuídos simétricos pelo texto à dinamicidade e heterogeneidade da carga do ambiente de execução através de oráculos de latência

Lívia Maria Rodrigues Sampaio

Tese de Doutorado submetida à Coordenação dos Cursos de Pós-Gradução em Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para obtenção do grau de Doutor em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Processamento da Informação

Francisco Vilar Brasileiro, Dr.
Orientador

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

S192a

Sampaio, Lívia Maria Rodrigues.

Adaptação de protocolos distribuídos simétricos pelo texto à dinamicidade e heterogeneidade da carga do ambiente de execução através de oráculos de latência / Lívia Maria Rodrigues Sampaio. — Campina Grande, 2007.

156f. : il.

Referências.

Tese (Doutorado em Engenharia Elétrica) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.

Orientador: Francisco Vilar Brasileiro, Dr.

1. Protocolos distribuídos. 2. Adaptação. 3. Desempenho. 4. Oráculos de latência. I. Título.

CDU-004.057.4

ADAPTAÇÃO DE PROTOCOLO DISTRIBUÍDOS SIMÉTRICOS PELO TEXTO À DINAMICIDADE E HETEROGENEIDADE DA CARGA DO AMBIENTE DE EXECUÇÃO ATRAVÉS DE ORÁCULOS DE LATÊNCIA

LÍVIA MARIA RODRIGUES SAMPAIO

Tese Aprovada em 17.04.2007

FRANCISCO VILAR BRASILEIRO, Ph.D., UFCG
Orientador

WAGNER MEIRA JUNIOR, Dr., UFMG (Ausência Justificada)
Componente da Banca

CARLOS ANDRÉ GUIMARÃES FERRAZ, Ph.D., UFPE
Componente da Banca

FABÍOLA GONÇALVES PEREIRA GREVE, Dr., UFBA
Componente da Banca

JORGE CÉSAR ABRANTES DE FIGUEIREDO, D.Sc., UFCG
Componente da Banca

CAMPINA GRANDE – PB
ABRIL - 2007

Agradecimentos

Este doutorado representa para mim uma grande conquista. Foram muitas experiências alcançadas, tanto profissionais quanto pessoais. Uma trajetória de altos e baixos, com um saldo muito positivo. Sinto-me feliz pela sensação de ter alcançado os objetivos traçados e vencido vários limites. Esta conquista não seria possível sem a participação de muitas pessoas, a quem gostaria de expressar meu agradecimento.

Primeiramente, a Deus, pelos dons que me concedeu e sabedoria para poder usá-los da melhor forma possível.

A minha família, por compartilharem de todas as alegrias e dificuldades enfrentadas ao longo da realização deste trabalho, além da presença constante e renovadora em minha vida.

Ao meu esposo, por todo o carinho, companheirismo e apoio ao meu crescimento profissional, não me deixando, também, esquecer que é importante reservar momentos de folga.

Ao professor Fubica, por ter direcionado com muita competência os meus estudos durante este trabalho. Ao longo destes anos em que trabalhamos juntos, o aprendizado foi muito valioso, seja através das reuniões técnicas, das críticas construtivas ou dos desafios vencidos.

Ao professor Michel Hurfin, por ter me orientado durante o período de doutorado sanduíche no *Institut de Recherche en Informatique et Systèmes Aléatoires*, Rennes, França. A experiência foi, sem dúvida, muito proveitosa.

A todos os que fazem o Laboratório de Sistemas Distribuídos, pelas experiências partilhadas e o exemplo de responsabilidade e dedicação na realização de um trabalho, além disso, pelas ótimas confraternizações. Gostaria de lembrar dos meus colegas de sala, atuais e antigos, além dos frequentadores assíduos, pelos quais cultivo toda admiração, em especial a Raquel, Ana Cristina, Marcelo Yuri, Ayla, Lauro, Mila, Andrey e Nelson.

Aos meus amigos, por todo o apoio e torcida. Seja de longe ou de perto, direta ou indiretamente, eles participaram dessa conquista. Em particular, agradeço aos amigos doutorandos (ou que já se tornaram doutores durante essa jornada) pela motivação e exemplo; sintam-se aqui lembrados através de Raquel, Ayla, Rodrigo Rebouças, Ana Karla, Franklin, Uirá, Roberta e Janine.

Por fim, mas não menos importante, à Universidade Federal de Campina Grande, através da Coordenação de Pós-Graduação em Engenharia Elétrica, pela oportunidade e o apoio necessário para realização deste trabalho.

Resumo

Protocolos distribuídos simétricos pelo texto podem ser configurados quanto aos papéis assumidos pelos seus participantes durante a execução dos mesmos. Além disso, a configuração do protocolo influencia no seu desempenho quando este executa em ambientes sujeitos à carga heterogênea e dinâmica, nesse caso, podem existir configurações eficientes ou ineficientes. Observa-se que não é possível garantir uma configuração eficiente a priori em ambientes dessa natureza. Dessa forma, protocolos simétricos pelo texto podem ter seu desempenho degradado em ambientes sujeitos à carga heterogênea e dinâmica.

Nesta tese investigou-se o uso de adaptação baseada em oráculos de latência, para fins de desempenho, em protocolos distribuídos simétricos pelo texto que executam em ambientes sujeitos à carga heterogênea e dinâmica. Tal investigação foi realizada através de estudos analítico e experimental sobre o desempenho de protocolos de consenso equipados com soluções adaptativas para o problema da ordenação de processos. O estudo analítico consistiu na elaboração de um modelo de desempenho para um protocolo de consenso adaptativo, dado em termos da definição formal de oráculos de latência proposta neste trabalho. A partir deste modelo de desempenho foi possível demonstrar a eficiência de uma solução adaptativa baseada em oráculos de latência e, por conseguinte, os ganhos de desempenho para o protocolo que a utiliza. Com esta mesma finalidade, porém usando um método diferente, o estudo experimental consistiu na realização de simulações e medições em um ambiente real. Para tal, foi projetado e implementado um subsistema de consenso adaptativo, inserido em uma aplicação para tolerância a intrusões na Internet. Realizou-se uma avaliação de desempenho comparativa, onde os resultados indicaram a superioridade do subsistema adaptativo sobre seu correspondente não-adaptativo. Ocorreram ganhos de desempenho nas simulações e nas medições, alcançando índices de até $76,1\%$ e $45,4\%$, respectivamente. Tanto no estudo analítico quanto no experimental utilizou-se adaptação para resolver o problema da ordenação de processos, que caracteriza muitos protocolos simétricos pelo texto.

Portanto, esta tese apresenta contribuições teóricas e práticas no contexto de protocolos simétricos pelo texto adaptativos para ambientes sujeitos à carga heterogênea e dinâmica. É importante enfatizar que o uso de oráculos de latência promove a separação de conceitos e a modularização na construção de soluções adaptativas baseadas nos mesmos, como também, favorece o estudo analítico sobre o desempenho de tais soluções.

Abstract

Distributed protocols with textual simmetry can be configured by the roles played by different participants during the execution of the protocols. Moreover, such a configuration may impact the performance of the protocol in a positive or negative way, depending on the configuration chosen. On the other hand, it is not possible to ensure an efficient configuration a priori when considering an execution environment with heterogeneous and dynamic workload. Consequently, distributed protocols with textual simmetry may suffer performance degradation on these environments.

In this thesis we investigated the use of adaptation by means of latency oracles to improve the performance of distributed protocols with textual simmetry that execute in environments subject to heterogeneous and dynamic workload. Such an investigation was conducted through both analytical and experimental studies on the performance of consensus protocols equiped with an adaptive solution to the process ordering problem. During the analytical study it was constructed a performance model for the consensus protocol using the formal definition of latency oracles, introduced in this work. From this performance model it was possible to demonstrate the efficiency of the adaptive solution and, consequently, the performance gains to the protocol that used it. Following the same objective, but considering a different method, the experimental study consisted of simulations and measurements in a real environment. In this case, we designed and implemented an adaptive consensus system encapsulated in an application for intrusion tolerance in the Internet. The performance of the adaptive consensus system was analyzed by means of comparison using its non-adaptive counterpart. The adaptive system outperformed the non-adaptive one in both simulations and measurements with performance gains of as much as $76,1\%$ and $45,4\%$, respectively. Note that, in the analytical and experimental studies, adaptation was used to solve the process ordering problem that characterizes a number of distributed protocols with textual simmetry.

Therefore, this thesis gives theoretical and practical contributions in the context of adaptive distributed protocols with textual simmetry that execute on environments subject to heterogeneous and dynamic workload. It is important to emphasize that the strategy of using latency oracles to construct adaptive solutions respects one of the most important principles of software engineering, which is the separation of concerns. Futhermore, it favors the analytical study on the performance of such adaptive solutions.

Conteúdo

Lista de Tabelas

Lista de Figuras

Lista de Algoritmos

Capítulo 1

Introdução

1.1 Motivação e Relevância

1.1.1 Protocolos distribuídos

Um sistema distribuído pode ser definido como uma coleção de computadores (hardware + software) independentes que trabalham de forma coordenada em prol de objetivos comuns, oferecendo aos seus usuários a visão consistente de um sistema único. Alguns exemplos típicos incluem a Internet, ambientes de grades computacionais e intranets de empresas. Tais sistemas tendem a ser mais eficientes (pela facilidade de compartilhamento de recursos), tolerante a faltas (pela redundância natural), econômicos (em relação aos computadores de grande porte usados para computação centralizada) e independentes (pela autonomia dos seus componentes). Por outro lado, existem desafios na área de sistemas distribuídos os quais têm motivado inúmeros trabalhos. Estes desafios se traduzem em problemas específicos (*e.g.*, alocação de recursos, detecção de impasses, detecção de falhas, comunicação entre processos, eleição de líder e acordo) que podem ser resolvidos através de protocolos distribuídos (RAYNAL, 1988; LYNCH, 1996).

Um protocolo distribuído é constituído por um conjunto de processos (ou participantes) que cooperam entre si para realizarem uma atividade comum. A comunicação entre os processos ocorre, comumente, por meio de troca de mensagens através de uma rede de comunicação. Entretanto, protocolos distribuídos também incluem aqueles protocolos projetados para sistemas multiprocessados com comunicação através de memória compartilhada (LYNCH, 1996).

Existem diferentes atributos (ou critérios) usados para caracterizar protocolos distribuídos, tais como grau de particionamento, suposições sobre a rede de comunicação e

resistência a falhas. Em particular, o grau de particionamento de um protocolo refere-se à simetria dos papéis desempenhados pelos seus participantes e está associado com a estrutura e comportamento do protocolo. Nesse caso, os protocolos distribuídos podem ser agrupados em quatro classes, a saber (RAYNAL, 1988):

- **assimétricos**: cada processo executa um programa (texto) diferente de acordo com o papel assumido pelo mesmo. Assim, a assimetria está no programa e gera comportamentos diferentes para os processos participantes do protocolo (*e.g.*, detecção de falhas (NUNES, 2003));

- **simétricos pelo texto**: todos os processos executam o mesmo programa, entretanto, as ações do programa possuem marcações indicando a identidade dos processos que devem executá-las. O papel assumido por cada processo está associado com as ações executadas pelo mesmo. Nesse caso, os processos devem ser identificados de forma consistente, comumente usando números únicos, permitindo a atribuição correta de papéis e, consequentemente, das ações a serem realizadas por cada um. Dessa forma, existe simetria no programa, mas os processos apresentam comportamentos diferentes de acordo com seus respectivos papéis (*e.g.* consenso baseado no paradigma do coordenador rotativo (CHANDRA; TOUEG, 1996));

- **fortemente simétricos**: todos os processos executam o mesmo programa e são indistinguíveis. Nesse caso, todas as ações do programa serão executadas por todos os processos. Entretanto, mensagens podem ser recebidas em ordens diferentes pelos processos fazendo-os apresentar comportamentos diferentes em tais situações. Assim, existe simetria no programa e pode ou não existir simetria no comportamento dos processos. Isto sugere que a definição dos papéis depende das características da execução do protocolo e pode variar em cada execução, não sendo possível identificar tais papéis a priori (*e.g.* difusão de mensagens (CHANDRA; RAMASUBRAMANIAN; BIRMAN, 2001)); e

- **totalmente simétricos**: todos os processos executam o mesmo programa e são indistinguíveis, apresentando ainda o mesmo comportamento. Assim, a simetria ocorre no programa e no comportamento dos processos, e todos os processos assumem o mesmo papel durante a execução do protocolo (*e.g.*, nomeação de processos (MOSTEFAOUI; RAYNAL; TRAVERS, 2006)).

Protocolos fortemente/totalmente simétricos caracterizam-se por seus processos executarem ações similares, as quais não dependem da identidade dos mesmos, assim, os

processos são considerados indistinguíveis. Protocolos desta natureza são denominados de anônimos. A motivação para o uso de anonimato na computação distribuída deve-se ao fato de que nem sempre é possível, necessário ou desejável, atribuir identificadores aos processos que executam uma determinada computação, possibilitando a construção de soluções mais simples (KRANAKIS, 1996); alguns exemplos de domínios de problema para soluções anônimas incluem redes de sensores e sistemas web com requisitos de privacidade (GUERRAOUI; RUPPERT, 2005; BUHRMAN et al., 2006). Por outro lado, muitos problemas de sistemas distribuídos não possuem soluções determinísticas neste contexto (FICH; RUPPERT, 2003), tais como eleição de líder e alocação de recursos. Portanto, "quebrar a simetria" de um conjunto de processos é um problema importante em sistemas distribuídos. Nesse sentido, algumas alternativas incluem: usar programas diferentes para os processos, usar identificadores únicos para os processos, usar o mesmo programa e associar estados iniciais diferentes para os processos e usar randomização (LYNCH, 1996; FLOCCHINI et al., 2004; BUHRMAN et al., 2006; MOSTEFAOUI; RAYNAL; TRAVERS, 2006).

Em protocolos assimétricos e simétricos pelo texto a simetria entre os processos é quebrada através do uso de programas diferentes e identidades únicas para os processos, respectivamente. Neste último caso, os identificadores são usados para controlar que partes do programa são executadas por quais processos. Pode-se citar como exemplo de protocolos assimétricos aqueles baseados no modelo cliente/servidor (comunicação remota, transferência de arquivos remota, linearização/restauração de dados, monitoramento em sistemas de detecção de falhas), enquanto protocolos simétricos pelo texto podem ser exemplificados através de soluções para problemas de acordo (CHANDRA; TOUEG, 1996; GUERRAOUI; RAYNAL, 2004; HURFIN et al., 2001; GREVE et al., 2001; AGUILERA et al., 2001; BRASILEIRO et al., 2001).

Um protocolo distribuído precisa satisfazer as propriedades do problema que o mesmo resolve. Estas propriedades podem ser de dois tipos: segurança (*safety*) e vivacidade (*liveness*) (LAMPORT, 1990). Segurança está relacionada às ações corretas que devem ocorrer no protocolo, enquanto vivacidade está relacionada com a ocorrência destas ações em algum momento. Portanto, vivacidade diz respeito à terminação do protocolo, ou seja, à garantia de que o protocolo produzirá uma saída desejada em um tempo finito. Então, o tempo de terminação de um protocolo significa o tempo transcorrido desde o início até o término de uma execução do mesmo. Sob o ponto de vista prático, a propriedade de terminação pode ser usada para medir o desempenho dos protocolos, nesse caso, quanto menor o tempo de terminação de um protocolo melhor será o seu desempenho.

A configuração de um protocolo distribuído pode influenciar no tempo de terminação do

mesmo. Por configuração entenda-se a associação de valores aos parâmetros do protocolo. Em se tratando de protocolos distribuídos onde seus participantes se comunicam através de troca de mensagens, um parâmetro importante é o padrão de comunicação[1], ou seja, a dinâmica da troca de mensagens de/para cada participante. Nesse caso, o tempo de terminação é determinado pelo atraso na comunicação fim-a-fim. Por exemplo, seja um protocolo constituído pelo conjunto de participantes $P = \{p_1, p_2, p_3\}$ os quais se comunicam através de 2 tipos de mensagens, m_1 e m_2. Então, uma configuração para o padrão de comunicação usado neste protocolo pode ser o descrito na Tabela 1.1.

Padrão de comunicação
p_1 envia m_1 para $\forall p_i \in P$
p_1 recebe m_2 de $\forall p_i \in P$
p_2 envia m_2 para p_1
p_2 recebe m_1 de p_1
p_3 envia m_2 para p_1
p_3 recebe m_1 de p_1

Tabela 1.1: Exemplo de configuração para um padrão de comunicação em um protocolo distribuído

Considere o padrão de comunicação um parâmetro configurável cujos valores constituem diferentes configurações para o respectivo protocolo. No caso de um ambiente sujeito à carga homogênea, qualquer configuração do padrão de comunicação estará sujeita aos mesmos atrasos fim-a-fim, então, a escolha por uma ou outra configuração resultará no mesmo desempenho para o protocolo. Esta conclusão é válida tanto para carga homogênea estática quanto dinâmica; esta última condição implica em mudanças na carga ao longo do tempo, entretanto, tais mudanças ocorrem em todo o sistema de forma consistente. O mesmo não é válido para cargas heterogêneas. De modo geral, carga heterogênea implica em atrasos fim-a-fim diferentes para comunicação entre processos, consequentemente, a escolha por uma ou outra configuração de padrão de comunicação pode resultar em desempenhos diferentes para o protocolo. Voltando ao exemplo discutido anteriormente, a Tabela 1.2 ilustra 3 configurações diferentes de padrão de comunicação. Se os atrasos fim-a-fim entre os elementos em P representam uma carga heterogênea onde p_1 é o mais rápido (atrasos menores na comunicação com p_2 e p_3) e p_3 é o mais lento, a configuração 1 será mais eficiente do que a 3. Portanto, usar uma ou outra configuração irá influenciar no desempenho do

[1]De fato, padrão de comunicação não costuma ser um parâmetro explícito de um protocolo distribuído, mas é representado através de outros parâmetros, *e.g.*, a quantidade de elementos de um conjunto.

protocolo.

Padrão de comunicação 1	*Padrão de comunicação 2*	*Padrão de comunicação 3*
p_1 envia m_1 para $\forall p_i \in P$	p_2 envia m_1 para $\forall p_i \in P$;	p_3 envia m_1 para $\forall p_i \in P$
p_1 recebe m_2 de $\forall p_i \in P$	p_2 recebe m_2 de $\forall p_i \in P$	p_3 recebe m_2 de $\forall p_i \in P$
p_2 envia m_2 para p_1	p_1 envia m_2 para p_2	p_2 envia m_2 para p_3
p_2 recebe m_1 de p_1	p_1 recebe m_1 de p_2	p_2 recebe m_1 de p_3
p_3 envia m_2 para p_1	p_3 envia m_2 para p_2	p_1 envia m_2 para p_3
p_3 recebe m_1 de p_1	p_3 recebe m_1 de p_2	p_1 recebe m_1 de p_3

Tabela 1.2: Exemplo de um padrão de comunicação com 3 configurações possíveis

No contexto de ambientes sujeitos à carga heterogênea, se a carga for estática, é possível garantir um bom desempenho para o protocolo definindo-se uma configuração eficiente a priori, o que requer o conhecimento do estado de carga do ambiente antes da execução do protocolo. Por outro lado, ambientes sujeitos à carga heterogênea e dinâmica experimentam variações contínuas de carga. Nesse caso, podem existir diferentes configurações eficientes para o protocolo durante a mesma execução e para obter um melhor desempenho seria necessário mudar a configuração do protocolo sempre que a situação de carga do ambiente não favorecesse o uso da configuração corrente.

1.1.2 Protocolos distribuídos adaptativos

Heterogeneidade e dinamismo da carga são características cada vez mais comuns no contexto de sistemas distribuídos reais, tais como plataformas de testes para aplicações de larga escala (*e.g.*, PlanetLab) e sistemas de clusters (ex. MapReduce da Google) (RHEA et al., 2005; DEAN; GHEMAWAT, 2004; DABEK et al., 2004). É desejável que um sistema computacional possa apresentar o melhor desempenho possível. Em se tratando de sistemas heterogêneos e dinâmicos, o uso de adaptação é requisito fundamental para alcançar bons índices de desempenho (AKSIT; CHOUKAIR, 2003).

Protocolos adaptativos são capazes de mudar seu comportamento, em tempo de execução, como resposta às variações nas condições do ambiente no qual executam, tendo como objetivo minimizar os efeitos negativos de tais variações sobre o seu desempenho. Um exemplo de protocolo distribuído adaptativo é o TCP (*Transmission Control Protocol*), da pilha de protocolos TCP/IP, o qual utiliza mecanismos de adaptação para controle de fluxo, retransmissão e controle de congestão (JACOBSON, 1988).

Discute-se na literatura dois tipos de adaptação, a saber: por código e por valor. A

adaptação por código (ou por algoritmo) implica em mudanças na implementação do protocolo, seja introduzindo novos módulos ou alterando módulos existentes. Por outro lado, **adaptação por valor** significa alterar o valor de um parâmetro ou configuração do protocolo. No contexto de ambientes sujeitos à carga heterogênea e dinâmica, o uso de adaptação por valor busca fazer com que o protocolo seja executado com a melhor configuração possível a todo momento, favorecendo o seu bom desempenho em diferentes cenários de carga do sistema.

A seguir discute-se o uso de adaptação por valor nas classes de protocolos distribuídos apresentadas na seção anterior, considerando ambientes sujeitos à carga heterogênea e dinâmica com comunicação por troca de mensagens. Nesse caso, adaptação poderá ocorrer sobre parâmetros do protocolo cujos valores influenciem no seu desempenho. Estes parâmetros serão agrupados em duas categorias: comportamento e controle. Parâmetros de comportamento referem-se aos papéis assumidos por cada participante ao longo da execução do protocolo, enquanto qualquer outro aspecto do protocolo é representado por parâmetros de controle (*e.g.*, caminho percorrido pela mensagem durante a transmissão pela rede, quantidade de elementos de um conjunto). Parâmetros de comportamento e parâmetros de controle estão diretamente ou indiretamente ligados à definição do padrão de comunicação entre os processos.

Como foi discutido anteriormente, protocolos totalmente simétricos caracterizam-se pela similaridade de programa, identidade e comportamento dos processos, sendo assim, todos os processos desempenham o mesmo papel. O valor do parâmetro de comportamento é idêntico e permanente para todos os processos. Então, adaptação nesses protocolos pode ocorrer, apenas, sobre parâmetros de controle. O mesmo se aplica aos protocolos fortemente simétricos. Apesar da possibilidade dos processos desempenharem papéis diferentes em alguns momentos da execução do protocolo, estes não são conhecidos e a ocorrência dos mesmos depende da ordem em que as mensagens são recebidas pelos processos. Isto significa que a atribuição ou mesmo mudança de papéis em protocolos fortemente simétricos é imprevisível, tornando o parâmetro de comportamento destes protocolos não configurável ou passível de adaptação. É interessante observar que o parâmetro de comportamento não é configurável, mas pode assumir valores diferentes para processos diferentes durante a execução do protocolo.

É possível encontrar na literatura exemplos de protocolos totalmente e fortemente simétricos adaptativos. Estes exemplos se concentram, principalmente, no contexto de sistemas distribuídos com comunicação através de memória compartilhada, como também, comu-

nicação através de troca de mensagens utilizando redes *ad-hoc* móveis[2] (MOSTEFAOUI; RAYNAL; TRAVERS, 2006; CHANDRA; RAMASUBRAMANIAN; BIRMAN, 2001; ATTIYA; BORTNIKOV, 2002; ATTIYA; FOUREN, 2001) (não faz parte do escopo desta tese).

Em protocolos assimétricos os processos participantes assumem papéis diferentes, os quais são definidos a priori e, comumente, não mudam durante a computação. Sendo assim, os valores do parâmetro de comportamento do protocolo não são iguais para todos os processos e não mudam, caracterizando um parâmetro não configurável. Nesse caso, adaptação por valor poderá ser aplicada sobre parâmetros de controle. Note que poderão existir vários padrões de comunicação entre os processos (combinação de valores para os parâmetros de controle e de comportamento), resultando em configurações eficientes ou ineficientes. O uso de adaptação sobre parâmetros de controle impacta positivamente no desempenho dos protocolos, apesar de não garantir o uso de configurações eficientes a todo momento, pois estas também dependem do parâmetro de comportamento que não é configurável. Dessa forma, o valor do parâmetro de comportamento definido no início da execução do protocolo pode tornar-se inadequado quando a situação de carga sofrer variações. Nesse contexto tem-se alguns exemplos de trabalhos disponíveis na literatura (CATãO; BRASILEIRO; OLIVEIRA, 2005; DéFAGO et al., 2005; NUNES, 2003)

Protocolos simétricos pelo texto também caracterizam-se por seus participantes assumirem papéis diferentes durante a computação realizada tal como em protocolos assimétricos. A diferença é que os papéis podem ser alternados, em tempo de execução, de maneira imprevisível ou previsível. No primeiro caso, a mudança de papéis depende do estado dos processos ao longo da execução do protocolo ou da ocorrência de eventos em momentos não conhecidos *e.g.*, obtenção de uma permissão (YAN; ZHANG; YANG, 1996), alteração da carga de processamento ou capacidade do processo (NIEUWPOORT et al., 2006). Sendo assim, o parâmetro de comportamento do protocolo não é configurável e adaptação por valor ocorre, apenas, sobre parâmetros de controle. Por outro lado, quando as mudanças são previsíveis, a atribuição e mudança de papéis dos participantes do protocolo não é determinada por nenhum fator de caráter imprevisível, mas por uma função determinística e, normalmente, aplicada de forma periódica. Nesse caso, o parâmetro de comportamento do

[2]Uma rede *ad-hoc* móvel (em inglês, Mobile ad hoc Networks (MANET)) não exige nenhuma infraestrutura fixa para a operação normal da rede. Dessa forma, toda e qualquer operação necessária para a comunicação entre os nodos da rede (*e.g.*, controle de acesso ao meio, descoberta de vizinhança, roteamento) é realizada de uma forma totalmente descentralizada. Além disso, por se tratar de uma rede móvel, assume-se que a comunicação entre os nodos é via radiofrequência, *i.e.*, sem fio. Portanto, as características de uma rede *ad-hoc* móvel introduzem novas restrições e preocupações quando da definição de protocolos distribuídos.

protocolo é configurável, além disso, seus valores não são iguais para todos os processos e não são fixos. Dessa forma, pode-se aplicar adaptação tanto sobre parâmetros de comportamento quanto de controle. Isto significa que existem vários padrões de comunicação entre os processos durante uma execução do protocolo, e estes podem ser eficientes para uma determinada situação de carga do ambiente e ineficientes para outra, assim, adaptação por valor pode ser usada para escolher os padrões de comunicação de acordo com a situação de carga do ambiente, permitindo que o protocolo seja executado com a melhor configuração possível a cada momento (RODRIGUES; FONSECA; VERíSSIMO, 1996; ANTONIS et al., 2004).

Protocolos de consenso baseados no paradigma do coordenador rotativo são exemplos de protocolos simétricos pelo texto com mudança de papéis previsíveis (CHANDRA; TOUEG, 1996; GUERRAOUI; RAYNAL, 2004; HURFIN et al., 2001). Tais protocolos procedem em rodadas, onde em cada rodada existe um processo que desempenha o papel de coordenador e os demais colaboram com o coordenador para obter um valor consensual. Os papéis mudam periodicamente, a cada rodada, independente de qualquer condição. O desempenho do protocolo está relacionado com o desempenho do processo coordenador; se o mesmo estiver lento, o tempo de terminação do protocolo será degradado. Para este tipo de protocolo os parâmetros de comportamento são configuráveis, sendo assim, existem várias configurações possíveis para o protocolo durante sua execução. A escolha por uma ou outra configuração deve levar em consideração a situação de carga do ambiente quando o mesmo é heterogêneo e dinâmico, ou seja, é importante usar adaptação para garantir um bom desempenho para o protocolo.

Por um lado, o consenso constitui um importante bloco básico para a construção de sistemas distribuídos tolerante a faltas, além disso, permite um projeto baseado em uma abordagem modular e funcional (FRIEDMAN; RAYNAL, 2004). Entretanto, sabe-se que soluções para o consenso introduzem uma sobrecarga considerável no sistema onde executam. Assim, questões de desempenho do consenso têm sido fortemente discutidas na literatura, onde os trabalhos se concentram no estudo do impacto de fatores internos e externos ao protocolo sobre o seu desempenho. De fato, a influência de fatores internos ao protocolo (características estruturais) ainda é pouco explorada, o que motiva novas pesquisas na área. Neste trabalho considera-se o uso de adaptação sobre características estruturais (parâmetros de comportamento) de protocolos de consenso baseados no paradigma do coordenador rotativo em sistemas distribuídos heterogêneos e dinâmicos. Até onde se sabe, este trabalho é o primeiro a considerar o uso de adaptação para fins de desempenho nesse contexto (SAMPAIO et al., 2003; SAMPAIO; BRASILEIRO; MOREIRA, 2004; SAMPAIO;

BRASILEIRO, 2005; SAMPAIO et al., 2005).

1.1.3 Oráculos de latência em protocolos distribuídos adaptativos

Um requisito fundamental para construir protocolos distribuídos adaptativos por valor em ambientes sujeitos à carga heterogênea e dinâmica é disponibilizar informações sobre a carga do ambiente. Tal informação servirá para identificar configurações do protocolo ineficientes e substituí-las por outras mais eficientes. Nesse caso, a informação desejada pode ser provida por uma espécie de oráculo.

Oráculos podem ser descritos como entidades do sistema que conhecem e provêem informações específicas as quais auxiliam (ou mesmo tornam possível) a computação realizada pelo mesmo. Espera-se que o oráculo seja capaz de fornecer informações sobre o passado, presente e futuro. Existem disponíveis na literatura vários trabalhos sobre oráculos em sistemas distribuídos, os quais fornecem diferentes tipos de informações, tais como processos falhos (MOSTEFAOUI; MOURGAYA; RAYNAL, 2002; CHANDRA; TOUEG, 1996; LARREA; FERNáNDEZ; ARéVALO, 2000), ordenação de mensagens (CAMARGOS; PEDONE; MADEIRA, 2006) e características da rede de comunicação (*e.g.*, latência, largura de banda e carga de processamento) (SHARMA et al., 2006; WONG; SILVKINS; SIRER, 2005; ZADOROZHNY et al., 2004; WOLSKI; SPRING; HAYES, 1999; GUMMADI; SAROIU; GRIBBLE, 2002).

Idealmente, um oráculo deveria ser definido, projetado e implementado seguindo uma abordagem modular e funcional (FRIEDMAN; RAYNAL, 2004). Nesse caso, o oráculo encapsularia uma determinada funcionalidade (especificada de forma precisa e formal) e forneceria uma interface de acesso aos seus clientes. Sob o ponto de vista prático, tal abordagem contribui para o processo de desenvolvimento de mecanismos de adaptação introduzindo separação de conceitos e modularização. Por outro lado, a definição formal do oráculo favorece o estudo analítico sobre a eficiência dos mecanismos de adaptação e seu impacto sobre o desempenho dos protocolos que os utilizam, permitindo a avaliação de desempenho destes protocolos mesmo antes de sua implementação.

Oráculos de latência provêem informações sobre atrasos na comunicação entre os processos do sistema. É possível encontrar na literatura trabalhos sobre oráculos de latência que seguem a abordagem modular funcional (SHARMA et al., 2006; WONG; SILVKINS; SIRER, 2005). Entretanto, a partir do estudo realizado sobre o assunto, tais trabalhos estão no contexto de sistemas largamente distribuídos ou usam uma especificação simplificada (ou pouco precisa) para definir o oráculo. Por outro lado, na área de protocolos distribuídos adaptativos em ambientes sujeitos à carga heterogênea e dinâmica, o conceito

de oráculos de latência é usado de maneira informal e, assim, não segue uma abordagem modular funcional. Consequentemente, não é possível usufruir dos benefícios práticos e teóricos associados ao uso de tal abordagem, os quais foram discutidos anteriormente.

1.2 Apresentação da tese

1.2.1 O escopo

A pesquisa apresentada nesta tese concentra-se na área de protocolos distribuídos simétricos pelo texto para ambientes sujeitos à carga heterogênea e dinâmica onde os processos se comunicam por troca de mensagens.

1.2.2 O problema

Protocolos distribuídos simétricos pelo texto podem ser configurados quanto aos papéis assumidos pelos seus participantes (parâmetro de comportamento), os quais estão associados à definição do padrão de comunicação entre os participantes do protocolo e, por conseguinte, ao desempenho do mesmo. Em ambientes sujeitos à carga heterogênea podem existir configurações mais eficientes do que outras e, o uso de uma configuração ineficiente causa degradação de desempenho. Se além de heterogênea a carga também for dinâmica, torna-se impossível garantir uma configuração eficiente a priori para protocolos simétricos pelo texto sujeitos à tal carga. Portanto, protocolos simétricos pelo texto podem sofrer perdas de desempenho se usarem uma única configuração em ambientes sujeitos à carga heterogênea e dinâmica. Questões de desempenho de protocolos distribuídos desta natureza foram o objeto de estudo desta tese.

1.2.3 A solução

Usar um mecanismo de adaptação por valor baseado em oráculos de latência para melhorar o desempenho de protocolos distribuídos simétricos pelo texto que executam em ambientes sujeitos à carga heterogênea e dinâmica. Nesse caso, adaptação por valor significa adaptar o protocolo mudando o valor de seu parâmetro de comportamento, ou seja, escolhendo uma nova configuração de papéis para o protocolo. Tais configurações são escolhidas de acordo com a situação de carga do sistema, sendo esta informação provida por um oráculo de latência. Portanto, oráculos de latência fornecem as informações necessárias para estimar o custo de comunicação associado às possíveis configurações do protocolo, permitindo ao

mecanismo de adaptação escolher configurações eficientes que favoreçam o desempenho do protocolo nas diversas situações de carga do ambiente.

1.2.4 A hipótese de tese

Protocolos distribuídos simétricos pelo texto podem obter melhor desempenho em ambientes sujeitos à carga heterogênea e dinâmica quando associados a mecanismos de adaptação baseados em oráculos de latência, nesse caso, assume-se adaptação por valor sobre parâmetros de comportamento.

1.2.5 A validação da hipótese

A hipótese de tese foi validada em duas etapas. O objetivo da primeira etapa foi mostrar que um mecanismo de adaptação escolherá a melhor configuração possível para um protocolo durante a sua execução. Nesse caso, realizou-se um estudo analítico sobre um mecanismo de adaptação associado a um protocolo de consenso, usando a definição formal de um oráculo de latência. Já a segunda etapa teve o objetivo de comprovar os ganhos de desempenho com o uso de adaptação através de um estudo de caso. Para tal, realizou-se um estudo comparativo entre um protocolo de consenso adaptativo baseado em oráculos de latência e seu correspondente não adaptativo. Os protocolos foram avaliados por meio de simulação e medições em um ambiente real.

1.2.6 As atividades realizadas

No sentido de validar a hipótese de tese especificada foi realizado um conjunto de atividades as quais descrevem as metas deste trabalho:

1. investigação do problema em estudo nesta tese e fundamentação teórica;
2. definição formal de oráculos de latência;
3. estudo sobre o uso de oráculos de latência em protocolos de consenso;
4. proposição de um mecanismo de adaptação baseado em oráculos de latência para protocolos de consenso;
5. estudo analítico sobre a eficiência do mecanismo de adaptação proposto;
6. estudo de caso para avaliação de desempenho de protocolos de consenso adaptativos baseados em oráculos de latência;

(a) definição do estudo de caso: uma aplicação baseada em consenso para tolerância a intrusões na Internet;

(b) simplificação do estudo de caso para enfatizar o protocolo de consenso;

(c) projeto e implementação de versões adaptativa e não-adaptativa de um protocolo de consenso, incluindo uma implementação simples de um oráculo de latência o qual está associado à versão adaptativa do protocolo;

(d) preparação para a avaliação de desempenho dos protocolos de consenso: as atividades incluíram a definição de objetivos, escolha do processo (simulação e medições em um ambiente real), definição de métricas, escolha das ferramentas de apoio e escolha do ambiente de teste;

(e) definição e parametrização dos cenários: na etapa de parametrização, foram realizados experimentos preliminares para coleta de dados que caracterizam os cenários definidos;

(f) realização dos experimentos de simulação e medições em um ambiente real;

(g) análise estatística dos experimentos;

(h) análise comparativa dos resultados de simulação e medições em um ambiente real.

1.2.7 As contribuições

Em linhas gerais, as contribuições desta tese são as seguintes: 1) uso de oráculos de latência para definir mecanismos de adaptação; 2) formalização de oráculos de latência, permitindo o estudo analítico da eficiência de um mecanismo de adaptação e, consequentemente, do desempenho de protocolos adaptativos associados a tais mecanismos; 3) definição, projeto e implementação de um oráculo de latência seguindo uma abordagem modular funcional; 4) uso de adaptação por valor sobre parâmetros de comportamento em protocolos simétricos pelo texto para ambientes sujeitos à carga heterogênea e dinâmica, o que constitui um assunto pouco explorado na literatura e; 5) estudo sobre adaptação em protocolos de consenso, tema não abordado na literatura até então.

1.3 Organização da tese

O restante deste documento está organizado em 5 capítulos os quais incluem a fundamentação teórica para a tese, instanciação da solução proposta, implementação e validação da

instância da solução através de um estudo de caso, e ainda, as conclusões e avaliação final do trabalho.

No Capítulo 2 é abordado o estado da arte sobre questões de desempenho e adaptação em sistemas distribuídos, incluindo conceitos, características de um sistema adaptativo, relação entre adaptação e desempenho e exemplos de sistemas adaptativos em ambientes heterogêneos e dinâmicos. Além disso, apresentam-se alguns trabalhos disponíveis na literatura sobre protocolos distribuídos adaptativos e oráculos.

O Capítulo 3 define formalmente um oráculo de latência para sistemas distribuídos de carga heterogênea e dinâmica e mostra como usar oráculos de latência para construir soluções adaptativas para protocolos distribuídos simétricos pelo texto. Em particular, introduz-se o problema da ordenação de processos que caracteriza muitos protocolos de consenso e como resolver este problema de forma adaptativa usando oráculos de latência. Este capítulo ainda inclui um estudo analítico sobre a eficiência de soluções adaptativas usando a definição formal de oráculos de latência.

O Capítulo 4 descreve o estudo de caso proposto para validar o uso de oráculos de latência na construção de protocolos distribuídos adaptativos. Nesse caso, propõe-se uma aplicação para tolerância a intrusões na Internet a qual é baseada em um subsistema de consenso dependente de ordenação de processos. Apresenta-se uma implementação adaptativa para tal subsistema de consenso que engloba um oráculo de latência.

O estudo de caso proposto no Capítulo 4 foi avaliado através da análise de desempenho comparativa entre os protocolos de consenso adaptativo e não-adaptativo. Nesse caso, introduz-se uma implementação não-adaptativa do subsistema de consenso estudado. Os resultados da análise são discutidos no Capítulo 5. Este capítulo inclui descrição dos experimentos e cenários utilizados, informações sobre a coleta dos dados de entrada para os experimentos, descrição das ferramentas de suporte para os experimentos e análise estatística detalhada sobre os resultados obtidos.

Por fim, no Capítulo 6, apresentam-se as conclusões e contribuições da pesquisa realizada nesta tese, com base na hipótese de tese especificada e resultados do processo de validação da mesma. Além disso, são apontados alguns trabalhos futuros para o tema discutido.

Capítulo 2

Um estudo sobre questões de desempenho e adaptação em sistemas distribuídos

Desempenho é um requisito não funcional associado à qualidade de serviço dos sistemas computacionais. Nesse caso, melhorar o desempenho dos sistemas significa torná-los mais eficientes.

Em se tratando de sistemas distribuídos, observa-se que muitas aplicações e serviços não são eficientes ou, são eficientes mas não são escaláveis (SMITH; WILLIAMS, 2000). Estas limitações de desempenho causam diminuição da produtividade dos usuários, perda de clientes, custos adicionais para reprojetar o sistema, perda de lucro e prestígio. Sendo assim, existe a necessidade de investir em estudos sobre questões de desempenho de sistemas distribuídos. Nesse sentido, um dos principais desafios é elaborar modelos e métricas adequados para avaliação de desempenho dos sistemas, haja vista que os mesmos são projetados para obterem resultados ótimos considerando tais métricas (KEIDAR, 2003). É interessante enfatizar que o estudo adequado sobre o desempenho de um sistema permite identificar as causas de ineficiências do mesmo, ou seja, entender os fatores que influenciam no desempenho do sistema (JAIN, 1991). Isto irá facilitar a escolha das abordagens corretas para lidar com tais causas.

Em particular, sistemas distribuídos heterogêneos e dinâmicos (*e.g.* Internet, ambientes de grades computacionais, intranets de empresas, ambientes de clusters) apresentam características que podem impactar negativamente no desempenho dos mesmos e, assim, torná-los ineficientes. Nesse contexto, adaptação é um requisito fundamental para obter sistemas mais eficientes (AKSIT; CHOUKAIR, 2003). É possível encontrar na literatura vá-

rios trabalhos sobre sistemas distribuídos adaptativos (HILTUNEN; SCHLICHTING, 1996; BRUSILOVSKY; MAYBURY, 2002; BERMAN et al., 2002; SILVA; ENDLER; KON, 2003; RAVINDRAN; KWIAT; SABBIR, 2004). Muitos destes trabalhos enfatizam questões de projeto e implementação dos mecanismos de adaptação utilizados, no sentido de diminuir os custos e aumentar a transparência dos mesmos para os usuários. Além dos aspectos práticos, é importante explorar os aspectos teóricos dos mecanismos de adaptação os quais podem auxiliar no entendimento e confirmação dos benefícios de usar adaptação para fins de desempenho.

O objetivo deste capítulo é estudar questões de desempenho e adaptação em sistemas distribuídos heterogêneos e dinâmicos, enfatizando conceitos, resultados e desafios nessa área, tanto sob o ponto de vista prático quanto teórico. Além disso, deseja-se motivar o uso de adaptação para construir protocolos distribuídos eficientes em ambientes sujeitos à carga heterogênea e dinâmica.

O restante desse capítulo está organizado da seguinte forma. Inicialmente, na Seção 2.1, discute-se o caráter heterogêneo e dinâmico do sistemas distribuídos modernos, ressaltando ainda o modelo de representação para tais sistemas (assíncrono com detectores de falhas não confiáveis). Adaptabilidade de sistemas distribuídos será discutida na Seção 2.2, enfatizando a importância de adaptação para melhorar o desempenho dos sistemas. A Seção 2.3 inclui exemplos de protocolos distribuídos que usam adaptação para fins de desempenho, sobre os quais apresentam-se a definição do respectivo mecanismo de adaptação e características do processo de avaliação de desempenho empregado. Uma característica de protocolos adaptativos pouco explorada formalmente refere-se aos oráculos que fornecem informações sobre o ambiente de execução, na Seção 2.4 discute-se este tema e apresenta-se alguns exemplos de como oráculos são definidos na literatura de sistemas distribuídos. Finalmente, na Seção 2.5, aparecem alguns comentários sobre os principais resultados da literatura apresentados nesse capítulo e que motivam a pesquisa desta tese.

2.1 Sistemas distribuídos heterogêneos e dinâmicos

Os sistemas distribuídos modernos tendem a ser mais heterogêneos e dinâmicos. A heterogeneidade e dinamismo de tais sistemas pode se apresentar de várias formas, tais como, variação na disponibilidade de recursos (comunicação e processamento), diversidade de hardware e software, variações na carga da rede e carga de processamento, alterações nas configurações de hardware ou software, como também, diversidade de seus usuários e aplicações. Dois exemplos típicos de sistemas distribuídos heterogêneos e dinâ-

micos são a Internet e alguns ambientes de clusters. A Internet serve de cenário para várias aplicações, tais como aplicações de comércio eletrônico, e ainda é usada como infra-estrutura para redes sobrepostas, tais como plataformas para testes (ex. PlanetLab (PATERSON et al., 2003; SPRING et al., 2006), RON (ANDERSEN et al., 2001)), redes de grades computacionais (ex. Condor, `http://www.cs.wisc.edu/condor`; Globus, `http://www.globus.org` ; OurGrid, `http://www.ourgrid.org`) e redes par-a-par - P2P - (ex. Gnutella, `http://www.gnutella.com`; Skype, `http://www.skype.com`). Por outro lado, ambientes de clusters caracterizam-se por um conjunto de computadores interligados, normalmente, por uma rede rápida, e que proporciona maior capacidade/velocidade de processamento do que aquela oferecida por um único computador (ex. clusters da Google (DEAN; GHEMAWAT, 2004)).

Um exemplo de sistema distribuído moderno é o PlanetLab, uma rede sobreposta à Internet composta de centenas de máquinas distribuídas geograficamente em diversos domínios, sendo caracterizada, dentre outros aspectos, pela heterogeneidade e dinamismo da carga (processamento e comunicação) (RHEA et al., 2005). A rede reúne instituições acadêmicas, comerciais e governamentais, as quais são responsáveis por gerenciar, promover e oferecer suporte à infra-estrutura de hardware e software da mesma. Tais instituições doam recursos e em troca podem ter acesso a "fatias" da rede, nesse caso, utiliza-se o conceito de virtualização distribuída[1]. O desenvolvimento do PlanetLab é guiado pelos seguintes objetivos: 1) oferecer suporte para testes e validação de aplicações de escala planetária, particularmente, aplicações distribuídas de larga escala para a Internet idealizadas no contexto de pesquisa; 2) viabilizar o acesso a serviços de larga escala por parte de um grande contingente de usuários, fornecendo o suporte para implantação e manutenção de tais serviços; e 3) fomentar novas pesquisas no sentido de investigar a evolução da Internet na direção de uma arquitetura orientada a serviços. Alguns exemplos de uso do PlanetLab incluem: medições de rede, rede de distribuição de conteúdo, armazenamento em larga escala, alocação de recursos, tabelas hash distribuídas, estudos sobre virtualização e criação de ambientes de testes federados[2].

Como foi citado acima, o PlanetLab é um sistema distribuído sujeito à carga heterogênea e dinâmica. Em (RHEA et al., 2005) foi descrito os resultados de um estudo sobre o desempenho de um serviço de tabelas hash distribuídas sobre o PlanetLab. Observou-se

[1]Uma fatia do PlanetLab pode agrupar inúmeras máquinas compartilhadas por meio de virtualização distribuída. Os usuários associados a cada fatia são responsáveis por adicionar ou remover as máquinas que a constituem.

[2]Mais informações sobre o PlanetLab, incluindo status atual, podem ser encontradas em `http://www.planet-lab.org`.

que, mesmo usando diferentes técnicas para otimizar o desempenho de tal serviço, o desempenho do mesmo sofre muitas variações devido à existência de nodos lentos. O problema dos nodos lentos foi estudado em duas plataformas de testes distribuídas (PlanetLab e RON) e um ambiente de cluster usando como métricas a latência de leitura no disco, tempo de processamento e latência de comunicação - tempo de ida e volta para uma mensagem; os resultados demonstraram que o desempenho individual dos nodos e destes em relação à rede de comunicação sofre variações significativas mesmo em um ambiente de cluster, reforçando o argumento de que muitos sistemas distribuídos estão sujeitos à carga heterogênea e dinâmica. Considerando um dos experimentos no RON para medir a latência de comunicação, um conjunto de nodos pode apresentar latências de comunicação altas e baixas (carga heterogênea) e, uma grande parcela dos nodos pode experimentar latências altas, pelo menos uma vez, dentro de um intervalo de tempo limitado (carga dinâmica). Em um dos experimentos no PlanetLab, o tempo de processamento para uma determinada computação que dura cerca de $10ms$ em uma máquina pode durar até $9.3s$ em outra (500 vezes maior); isto demonstra a imprevisibilidade associada ao desempenho das máquinas. Para o experimento no ambiente de clusters que mediu a latência de leitura no disco de um bloco de 1Kbyte, observou-se latências 54 vezes maior do que a média (carga dinâmica), e ainda, a fração de leituras ao disco em um intervalo de $1ms$ varia de 47% a 89% entre as máquinas (carga heterogênea).

Um outro resultado sobre desempenho em clusters foi descrito em (DEAN; GHEMAWAT, 2004), onde é apresentado o sistema MapReduce, usado para processamento e geração de uma grande quantidade de dados da Google; os autores constataram a existência de nodos lentos no cluster que executa o sistema. A presença de nodos lentos pode impactar negativamente o desempenho do sistema ao qual os nodos estão conectados e, para contornar esse problema, é preciso considerar os nodos lentos e lidar com os mesmos de forma automática (RHEA et al., 2005; DEAN; GHEMAWAT, 2004). Portanto, carga heterogênea e dinâmica é uma característica presente em sistemas distribuídos reais e que pode impactar negativamente o desempenho dos mesmos.

Na próxima seção será discutido o modelo de sistema assíncrono para representar sistemas distribuídos heterogêneos e dinâmicos. Em particular, apresenta-se o modelo assíncrono com detectores de falhas não confiáveis, o qual incorpora requisitos mínimos de sincronismo necessários para resolver problemas fundamentais de sistemas distribuídos sem perder a simplicidade e portabilidade do modelo assíncrono.

2.1.1 Modelo de sistema assíncrono

A maioria dos sistemas distribuídos reais são assíncronos. Isto porque, em tais sistemas, não é possível garantir restrições de sincronismo sobre os atrasos na comunicação entre os processos e escalonamento de tarefas (velocidade de processamento).

O modelo assíncrono traz algumas vantagens sobre o síncrono[3], por possuir uma semântica simples e permitir a construção de aplicações com maior portabilidade do que aquelas baseadas em limites de tempo para comunicação entre os processos. Entretanto, sistemas "puramente" assíncronos não incorporam nenhuma noção de tempo (FISCHER; LYNCH; PATERSON, 1985), dessa forma, não é possível determinar se um processo falhou ou, apenas, está muito lento. Este fato impede que muitos problemas práticos tenham soluções determinísticas sobre os modelos em questão. Entre tais problemas pode-se destacar: problemas do consenso, eleição de líder e filiação a grupo (FISCHER; LYNCH; PATERSON, 1985; CHANDRA et al., 1996). Sendo assim, na prática, os sistemas reais são, comumente, representados por modelos assíncronos intermediários entre o assíncrono "puro" e o modelo síncrono. Nesse contexto pode-se citar os modelos parcialmente síncronos (DOLEV; DWORK; STOCKMEYER, 1987; DWORK; LYNCH; STOCKMEYER, 1988), assíncrono temporizado (CRISTIAN; FETZER, 1999) e assíncrono equipados com detectores de falhas não confiáveis (CHANDRA; TOUEG, 1996). Estes modelos incluem mecanismos capazes de aumentar o sincronismo necessário para permitir que os sistemas assíncronos sejam utilizados na prática.

Introduzir detectores de falhas não confiáveis em sistemas assíncronos no sentido de aumentar as restrições de sincronismo associadas aos mesmos é uma das abordagens mais discutidas na literatura. A razão é que esta abordagem provê uma forma elegante e modular de introduzir requisitos de sincronismo no modelo assíncrono "puro". Esta característica facilita a validação (prova de correção) das soluções projetadas sobre o modelo assíncrono, como também, a implementação das mesmas.

2.1.2 Modelo de sistema assíncrono com detectores de falhas não confiáveis

A abordagem dos detectores de falhas não confiáveis foi proposta por Chandra e Toueg e apresentada em (CHANDRA; TOUEG, 1996). A idéia dos autores é estender o modelo de sistema assíncrono com um mecanismo para detecção de falhas, passível de erros, mas, sufi-

[3]Um sistema síncrono caracteriza-se pela certeza na comunicação, *i.e.*, em tais sistemas os atrasos na transmissão de mensagens e escalonamento de tarefas são limitados e conhecidos (SCHEIDER, 1993).

cientemente confiável no sentido de fornecer informação sobre falhas por parada[4] ocorridas no sistema, permitindo contornar o resultado de impossibilidade FLP (FISCHER; LYNCH; PATERSON, 1985). Nesse caso, os detectores de falhas não confiáveis são componentes que servem de oráculos ao sistema, informando os processos falhos.

Um detector de falhas distribuído é composto por um conjunto de módulos locais, onde cada módulo monitora um subconjunto de processos do sistema e mantém uma lista de processos considerados suspeitos de terem sofrido uma falha por parada. Estes módulos podem cometer erros, seja suspeitando dos processos corretos ou não suspeitando dos processos que, realmente, falharam (FISCHER; LYNCH; PATERSON, 1985). Isto acontece porque, num sistema assíncrono não é possível assegurar se um processo falhou por parada ou está lento, mas permanece funcionando corretamente. Embora um módulo detector de falhas possa adicionar processos corretos a sua lista de suspeitos, estes também podem ser removidos posteriormente, se o módulo reconhecer o erro cometido. Então, cada módulo pode adicionar e remover processos de sua lista de suspeitos, repetidas vezes. Além disso, módulos associados a processos diferentes podem ter listas de suspeitos distintas.

Chandra e Toueg introduziram várias classes de detectores de falhas. Cada classe é definida em função de duas propriedades abstratas[5]: uma propriedade de abrangência (*completeness*) e uma propriedade de exatidão (*accuracy*). Estas propriedades garantem, respectivamente, a vivacidade e a segurança dos detectores de falhas. Abrangência assegura que falhas reais serão, em algum momento, identificadas pelo detector de falhas. Por outro lado, exatidão restringe os erros que um detector de falhas pode cometer.

As classes de detectores de falhas introduzidas por Chandra e Toueg diferem em relação ao nível das restrições impostas pelas suas respectivas propriedades de abrangência e exatidão. Ou seja, a diferença está na qualidade da informação sobre falhas nos processos do sistema que os detectores de cada classe podem prover. Note que, quanto maior o nível de restrição mais complexos os detectores de falhas, então é desejável usar os detectores de falhas menos restritivos para resolver um determinado problema[6]. Por exemplo, para resolver o problema do consenso em ambientes onde uma maioria dos processos não podem falhar por parada, a classe de detectores de falhas menos restritiva é identificada como $\Diamond S$ e apresenta as seguintes propriedades (CHANDRA; TOUEG, 1996):

[4]Se um componente (processo/processador) falha por parada, então, o mesmo falha parando de operar.

[5]Uma propriedade abstrata não depende de nenhuma implementação particular, seja um mecanismo em hardware ou software.

[6]Em (DELPORTE-GALLET et al., 2004) são apresentados os detectores menos restritivos para resolver alguns problemas importantes de sistemas distribuídos, considerando diferentes suposições de falhas para o sistema.

- **Abrangência forte**: existe um tempo, a partir do qual, todos os processos corretos irão sempre suspeitar dos processos que falharam por parada.

- **Exatidão forte após um tempo**: existe um tempo, a partir do qual, todos os processos corretos não irão suspeitar de algum processo correto.

Chandra e Toueg ainda discutem a equivalência entre classes de detectores de falhas. Dois detectores de falhas D e D' são considerados equivalentes quando é possível transformar D em D', e vice-versa, usando um algoritmo de redução apropriado[7]. Nesse caso, é dito que D' é redutível a D e D é redutível a D', respectivamente. Detectores equivalentes apresentam o mesmo poder computacional, podendo ser usados um no lugar do outro. Por exemplo, em (CHANDRA; HADZILACOS; TOUEG, 1996) é demonstrada a equivalência entre as classes de detectores $\Diamond S$, $\Diamond W$ e Ω.

Um detector de falhas Ω tem características bem peculiares. Os detectores desta classe, ao serem consultados, informam a identidade de um processo correto, o qual é denominado de líder. Diversos líderes podem coexistir durante um longo período de tempo, mas, em algum momento, todos os processos corretos devem eleger o mesmo líder. Além disso, não é possível prever quando este líder será eleito. Formalmente, um detector Ω é definido em termos da seguinte propriedade (CHANDRA; HADZILACOS; TOUEG, 1996):

- **Liderança após um tempo**: existe um processo correto p e um tempo t a partir do qual, qualquer processo correto que invocar o detector de falhas receberá como retorno o valor p.

2.2 Adaptação para fins de desempenho

Sistemas adaptativos são capazes de reagir a variações no ambiente onde executam. Uma das finalidades da adaptação é poder melhorar o desempenho dos sistemas, isto significa aumentar a eficiência ou diminuir o custo de execução dos mesmos (HILTUNEN; SCHLICHTING, 1996). Por exemplo, em sistemas distribuídos é comum o uso de temporizadores para decidir sobre retransmissão de mensagens perdidas ou reconfiguração do sistema no caso de falhas de algum componente (detecção de falhas). Nesse caso, protocolos baseados em temporizadores podem usar mecanismos de adaptação para ajustar os valores de seus temporizadores de acordo com as características do ambiente de execução (carga da rede e taxa

[7] Alguns exemplos de algoritmos de redução para detectores de falhas não confiáveis são apresentados em (CHANDRA; TOUEG, 1996; CHANDRA; HADZILACOS; TOUEG, 1996).

de falhas na rede decorrentes do número de colisões de mensagens). Note que adaptação é um tema fortemente associado a sistemas distribuídos, haja vista que tais sistemas tendem a ser heterogêneos e dinâmicos. Por esta razão, existem na literatura resultados sobre sistemas distribuídos adaptativos, em diferentes áreas, tais como: *World Wide Web* (BRUSILOVSKY; MAYBURY, 2002), tempo-real (RAVINDRAN; DEVARASETTY; SCHIRAZI, 2002; RAVINDRAN; KWIAT; SABBIR, 2004), computação em grades (HURFIN et al., 2003; BERMAN et al., 2002), segurança (ATIGHETCHI et al., 2003) e tolerância a faltas (KILLIJIAN; FABRE, 2003; RAVINDRAN; KWIAT; SABBIR, 2004; BONDAVALI; GIANDOMENICO; XU, 1993).

Projetar e implementar sistemas adaptativos não é uma tarefa fácil. Primeiramente, é preciso identificar quais componentes ou serviços do sistema devem tornar-se adaptativos (**O quê?**). Nesse caso, um aspecto fundamental é usar separação de conceitos, ou seja, os módulos (código) introduzidos no sistema para prover adaptação devem estar bem encapsulados e especificados, de modo a garantir que requisitos não-funcionais (por exemplo, adaptação) do sistema sejam tratados separadamente dos seus requisitos funcionais. A segunda preocupação é definir quais eventos irão disparar uma ação de adaptação (**Quando?**). Isto envolve o monitoramento do sistema para identificar situações não favoráveis para a execução do mesmo e, assim, iniciar o processo de adaptação. Finalmente, é necessário decidir de que maneira o sistema deve se adaptar (**Como?**). Idealmente, ações de adaptação devem ser transparentes para o usuário do sistema, sendo assim, a funcionalidade do sistema precisa ser preservada durante a execução de uma ação de adaptação. Nesse caso, ações de adaptação ocorrem durante o processamento normal do sistema, sem que o mesmo interrompa suas atividades. De fato, o custo de usar adaptação pode ser alto, o que exige alguns cuidados. Em um sistema adaptativo, idealmente, o custo de adaptação deve apenas ser percebido se ocorrer algum evento que cause uma ação de adaptação, isto significa que o usuário do sistema não deve pagar pela capacidade do sistema se adaptar, mas, pela adaptação de fato.

No sentido de facilitar o projeto e implementação de sistemas adaptativos, Chen, Hiltunen and Schlichting propõem uma arquitetura de software cujo principal atrativo é prover adaptação de forma transparente (CHEN; HILTUNEN; SCHLICHTING, 2001; HILTUNEN; SCHLICHTING, 1996). De acordo com esta arquitetura, um sistema adaptativo é composto por camadas adaptativas ou não-adaptativas. Cada camada adaptativa consiste de uma coleção de componentes adaptativos (CA), os quais podem interagir com componentes localizados em diferentes máquinas para prover um determinado serviço adaptativo (SA). Internamente, um componente adaptativo contém um controlador de adaptação (CRA) e

um conjunto de algoritmos diferentes (ALG) que implementam a funcionalidade do componente. Nesse caso, o controlador de adaptação monitora o ambiente de execução, identificando situações onde adaptação é requerida, e iniciando o processo de adaptação (troca de algoritmos ou mudança de parâmetros de configuração). A Figura 2.1 (CHEN; HILTUNEN; SCHLICHTING, 2001) ilustra a estrutura de um componente adaptativo.

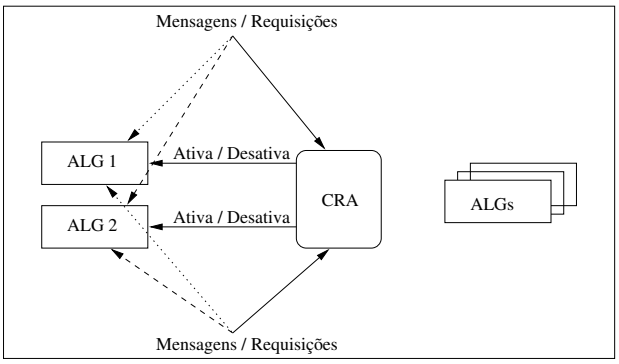


Figura 2.1: Arquitetura de um componente adaptativo

O processo de adaptação de um serviço adaptativo (SA) é dado em três etapas: detecção de mudanças, acordo e ação de adaptação. A primeira etapa envolve o monitoramento do ambiente de execução e identificação de eventos que podem gerar ações de adaptação (local a cada componente adaptativo). Na etapa de acordo, todos os componentes adaptativos do serviço irão entrar em consenso sobre o disparo de uma execução de adaptação. A etapa final do processo implementa a ação de adaptação propriamente, levando em consideração a transparência e, ainda, que o serviço continuará operando corretamente após a adaptação. Chen, Hiltunen and Schlichting ainda apresentam uma implementação da arquitetura proposta e exemplos de serviços de comunicação em grupo baseados nessa arquitetura (CHEN; HILTUNEN; SCHLICHTING, 2001; HILTUNEN; SCHLICHTING, 1996). Nesse caso, a adaptação ocorre pela troca de algoritmos.

Outros pesquisadores também têm se dedicado ao estudo de mecanismos que facilitem o uso e implementação de serviços adaptativos, tais como, arcabouços (SILVA; ENDLER; KON, 2003; ZINKY; BAKKEN; SCHANTZ, 1997), arquiteturas (BRUSILOVSKY, 2004; REN et al., 2003), middlewares (FITZPATRICK et al., 2001; SCHANTZ; SCHMIDT, 2002) e técnicas de programação (por exemplo, reflexão e aspectos) (KON et al., 2002; SOUZA, 2004). Estes mecanismos diferem quanto à tecnologia utilizada, recursos oferecidos, domínio de aplicação e complexidade.

2.3 Questões de desempenho e adaptação de protocolos distribuídos

Nesta seção deseja-se discutir questões de desempenho e adaptação no contexto mais específico de protocolos distribuídos, ou seja, uso de adaptação em protocolos distribuídos para fins de desempenho. Pode-se encontrar na literatura diversos exemplos de protocolos distribuídos adaptativos, seja por valor ou algoritmo, considerando ambientes heterogêneos e dinâmicos sob diferentes aspectos (*e.g.*, carga de processamento e comunicação, número de participantes do protocolo, nível de confiabilidade dos participantes do protocolo e do ambiente de execução), além de seguirem o modelo síncrono e assíncrono usando memória compartilhada ou troca de mensagens. Em particular, serão apresentados protocolos distribuídos adaptativos por valor em ambientes assíncronos com carga (processamento ou comunicação) heterogênea e dinâmica, onde os processos se comunicam por troca de mensagens.

Protocolos fortemente (totalmente) simétricos são caracterizados pelo anonimato entre os processos participantes, nesse caso, a execução do protocolo não depende das identidades de seus participantes. É possível encontrar na literatura exemplos de protocolos adaptativos com tais características (MOSTEFAOUI; RAYNAL; TRAVERS, 2006; CHANDRA; RAMASUBRAMANIAN; BIRMAN, 2001; ATTIYA; BORTNIKOV, 2002; ATTIYA; FOUREN, 2001), entretanto, normalmente, tais protocolos consideram comunicação por memória compartilhada, ambientes síncronos ou redes ad-hoc móveis que estão fora do escopo desta tese.

Em protocolos assimétricos os participantes assumem papéis diferentes durante a execução do protocolo, os quais, normalmente, são definidos a priori. Os participantes podem executar programas (ou textos) diferentes, de acordo com o papel assumido pelos mesmos. Protocolos baseados no modelo cliente-servidor possuem tais características. A assimetria de papéis também caracteriza os protocolos simétricos pelo texto, porém, utiliza-se o mesmo texto para todos os participantes e cada um executa as ações associadas com o seu papel. Além disso, os papéis podem mudar durante a execução do protocolo.

Protocolos assimétricos adaptativos podem ser encontrados em diferentes áreas, tais como: detecção de falhas (NUNES, 2003; DéFAGO et al., 2005), exclusão mútua (CHOCKLER; MALKHI; REITER, 2001), provisão dinâmica (URGAONKAR; SHENOY, 2005) e balanceamento de carga (ANDRZEJAK et al., 2002). Adaptação por valor é usada para reconfigurar os protocolos modificando parâmetros de controle que influenciem no desempenho dos mesmos. Em tais protocolos, os parâmetros de comportamento, cujos valores estão associados aos papéis de cada participante, não são configuráveis. Por exemplo, uma

estratégia para implementar detectores de falhas não confiáveis em sistemas assíncronos é usando temporizadores. A detecção de falhas é baseada em um protocolo de monitoramento onde os participantes assumem dois tipos de papéis: monitores (cliente) e monitorados (servidor); os clientes monitoram os servidores periodicamente e se não receberem respostas dos mesmos por um determinado período de tempo (valor do temporizador) caracteriza-se uma suspeita de falha. O temporizador constitui um parâmetro de controle. A definição dos temporizadores influencia na qualidade de serviço dos detectores de falhas (COCCOLI et al., 2002; URBáN et al., 2004; PEDONE et al., 2002), então, para aumentar a qualidade de serviço utilizam-se mecanismos de adaptação que ajustam os temporizadores de acordo com a carga de comunicação do ambiente (NUNES, 2003; CHEN; TOUEG; AGUILERA, 2000).

Adaptação por valor em protocolos simétricos pelo texto também é usada para reconfigurar parâmetros do protocolo, nesse caso, podem ser parâmetros de controle e de comportamento. Alterar o comportamento do protocolo significa mudar os papéis dos participantes. Mais uma vez, a literatura contém exemplos de protocolos simétricos pelo texto adaptativos em diferentes áreas (RODRIGUES; FONSECA; VERíSSIMO, 1996; YAN; ZHANG; YANG, 1996; NIEUWPOORT et al., 2006; ANTONIS et al., 2004), como os que serão descritos na próxima seção.

2.3.1 Protocolos adaptativos simétricos pelo texto

Nesta seção apresenta-se alguns exemplos de protocolos simétricos pelo texto adaptativos, usando reconfiguração de parâmetros de controle ou de comportamento, diferentes tipos de informação sobre o ambiente de execução e mecanismos de coleta de dados, além, de métodos e métricas de avaliação de desempenho sobre o efeito da adaptação nos protocolos propostos.

Difusão atômica em grupo. A difusão atômica em grupo (*totally ordered multicast*) tem como objetivo disseminar mensagens em um subgrupo de processos do sistema garantindo que todos os processos (participantes) corretos entreguem a mesma sequência de mensagens em uma mesma ordem total. Duas abordagens propostas para resolver difusão atômica em grupo são denominadas de *token-site* e simétrica (RODRIGUES; FONSECA; VERíSSIMO, 1996; BRASILEIRO; EZHILCHELVAN; SPEIRS, 1995). Na abordagem *token-site* escolhem-se alguns processos (possuidores do token) para ordenar as mensagens emitidas por todos os participantes do protocolo; já na abordagem simétrica todos os processos realizam ordenação das mensagens de forma descentralizada. Tais abordagens são consideradas ineficientes em ambientes sujeitos à carga heterogênea e dinâmica (RODRIGUES;

FONSECA; VERíSSIMO, 1996).

Rodrigues *et al.* (RODRIGUES; FONSECA; VERíSSIMO, 1996; RODRIGUES, 2003) propuseram um protocolo de difusão atômica em grupo híbrido que combina as duas abordagens citadas acima. O protocolo procede da seguinte forma. Mensagens são difundidas a todos os participantes do grupo, os quais podem assumir dois papéis (modos de operação): ativo ou passivo. Cada mensagem é identificada com um número (token) emitido, apenas, por participantes ativos que se comunicam para ordenar as mensagens seguindo a abordagem simétrica e usando o valor do token como critério de ordenação. Processos ativos ordenam suas próprias mensagens e aquelas emitidas por processos passivos, assim, estes usam os processos ativos como sequenciadores (abordagem *token-site*). Adaptação é usada para configurar o modo de operação dos participantes de acordo com a taxa de emissão de mensagens e atrasos na transmissão das mensagens do protocolo, ou seja, aspectos associados à carga de processamento e comunicação. Todo processo guarda informações sobre sua taxa e atrasos na comunicação com o processo ativo mais próximo, este último é representado pelo tempo de ida e volta de mensagens do protocolo respondidas, periodicamente, pelos seus participantes. A mudança de papéis é realizada por cada processo usando a mesma heurística, a qual é baseada em informações locais sobre taxa e atrasos na comunicação; as mudanças de papel devem ser comunicadas aos demais participantes do grupo. Nesse caso, as ações de adaptação têm o objetivo de mudar os papéis dos participantes durante a execução do protocolo, ou seja, reconfigurar o parâmetro de comportamento do protocolo. O protocolo adaptativo proposto foi avaliado por meio de simulação em um ambiente com carga de processamento (taxa) e comunicação (atrasos/latência) heterogêneos e dinâmicos, sendo esta gerada através de distribuições de probabilidade adequadas. Comparou-se o protocolo híbrido adaptativo com protocolos puramente *token-site* ou simétricos. Os resultados demonstram a eficiência do protocolo híbrido que obteve melhor desempenho em todos os cenários avaliados.

Exclusão mútua. Exclusão mútua é um problema de alocação de recursos que consiste no compartilhamento de recursos (região crítica) para os quais um conjunto de processos (participantes) necessita de acesso exclusivo por um período determinado de tempo. Uma estratégia para resolver este problema é usar um token único, disponível a todos os processos do sistema, cuja propriedade garante acesso exclusivo a uma região crítica. O desempenho de protocolos de exclusão mútua é medido, normalmente, através do número de mensagens trocadas e tempo para ganhar acesso à região crítica.

Yan *et al.* (YAN; ZHANG; YANG, 1996) propuseram um protocolo de exclusão mútua,

baseado em token, adaptativo a diferentes tipos de topologias de rede, ao estado dos processos participantes do protocolo e à contenção na rede. Para entrar na região crítica cada participante deve "perseguir o token" usando informações dinâmicas sobre a localização do token e contenção na rede armazenadas localmente. Informações sobre contenção na rede são atualizadas periodicamente enquanto as informações sobre o token são atualizadas à medida que este passa de um participante para outro. Seguindo o protocolo, envia-se uma mensagem de requisição pelo token através de um caminho específico que é construído a partir de informações locais sobre a última localização do token e contenção na rede; o caminho pode mudar de acordo com informações locais dos processos intermediários até alcançar o token. O protocolo possui trechos executados apenas pelos possuidores do token e outros pelos candidatos ao mesmo, mas a definição destes papéis não é configurável. Portanto, ações de adaptação provocam reconfiguração de parâmetros de controle. A avaliação do protocolo foi feita através de medições em redes com topologias diferentes (construídas logicamente), considerando diferentes configurações para o tráfego de requisições à região crítica e tamanho da mesma. As métricas de desempenho utilizadas referem-se ao número médio de mensagens (quantidade de nodos intermediários) e tempo de espera até ganhar acesso à região crítica. O protocolo de exclusão mútua proposto obteve melhor desempenho do que outros protocolos de mesma natureza.

Balanceamento de carga distribuído. Balanceamento de carga também é um problema relacionado com alocação de recursos, tendo como objetivo favorecer o desempenho do sistema através do bom uso dos recursos do mesmo. Nesse caso, utiliza-se informação sobre o estado do sistema para balancear a carga entre todos os servidores disponíveis. Protocolos adaptativos tornam o balanceamento mais eficiente já que informações sobre o estado do sistema são usadas continuamente e não apenas durante a inicialização do protocolo.

van Nieuwpoort *et al.* (NIEUWPOORT et al., 2006) propuseram um protocolo de balanceamento de carga onde os participantes do protocolo estão organizados em árvore considerando localidade na rede, em outras palavras, utiliza-se uma estrutura de múltiplos clusters em que máquinas de um mesmo cluster comunicam-se através de uma rede local (LAN - *Local Area Network*) e diferentes clusters comunicam-se através de uma rede de longa distância (WAN - *Wide Area Network*). De acordo com o protocolo, participantes ociosos tentam "roubar" tarefas sendo processadas em outros participantes sobrecarregados. Nesse caso, escolhe-se um participante em um cluster remoto para o qual envia-se uma requisição de tarefas de forma assíncrona. Então, escolhe-se, aleatoriamente, um participante do

cluster local e envia-se uma requisição de tarefas de forma síncrona. O envio local de requisições de tarefas perdura até que uma resposta (local ou remota) seja recebida. O objetivo dessa estratégia é diminuir a comunicação na rede de longa distância sem sobrecarregar a rede local com muitas transferências de tarefas, garantindo o balanceamento de carga eficiente em toda a rede (clusters remotos e locais). Note que a diferença entre os papéis está na condição de participante ocioso ou sobrecarregado, cada um executa trechos de código do protocolo diferentes. Dessa forma, o parâmetro de comportamento do protocolo não é configurável. Adaptação é usada na escolha do cluster remoto para onde a requisição assíncrona será direcionada; um cluster remoto será escolhido se a comunicação com o mesmo for rápida considerando a duração da última requisição enviada para este cluster. Nesse caso, ações de adaptação irão reconfigurar parâmetros de controle (identificação do cluster remoto). As informações sobre a comunicação com clusters remotos é atualizada sempre que uma nova requisição é enviada para os mesmos. A avaliação do protocolo foi feita através de simulação em um ambiente com múltiplos clusters distribuídos em uma WAN, usando diferentes cenários de carga (latência e banda) e aplicações de divisão-e-conquista. Comparou-se o *speed-up* das aplicações usando o protocolo adaptativo e uma versão não-adaptativa[8] do mesmo protocolo onde os clusters remotos são escolhidos aleatoriamente. Os resultados para o protocolo adaptativo foram melhores em todos os cenários.

2.4 Oráculos em protocolos distribuídos adaptativos

Uma característica comum aos protocolos distribuídos adaptativos por valor, como pode ser comprovado através dos exemplos descritos na seção anterior, refere-se ao uso de informações sobre o ambiente de execução, tanto para avaliar se uma ação de adaptação deve acontecer quanto para implementá-la. Através destas informações é possível identificar as melhores configurações para um protocolo durante sua execução, garantindo a eficiência das ações de adaptação sobre o desempenho do mesmo. Pode-se definir o mecanismo responsável por prover informações sobre o ambiente de execução como sendo um oráculo, isto porque estas informações irão compor configurações eficientes para execuções do protocolo no futuro próximo, então, assume-se que as informações providas pelo oráculo refletem o estado futuro do sistema.

Normalmente, os trabalhos na área de protocolos adaptativos descrevem os oráculos de

[8]Esta versão não-adaptativa apresenta um desempenho muito bom quando comparado com o de outros protocolos de balanceamento de carga de mesma natureza, mas, que não se baseiam na estrutura de árvore (NIEUWPOORT; KIELMANN; BAL, 2001).

maneira informal, ressaltando o tipo de informação provida (*e.g.*, latência, banda e vazão) e como obtê-la (*e.g.*, usando serviços já existentes, informações sobre recursos do sistema coletadas localmente ou globalmente). Além disso, questões de engenharia sobre o projeto e implementação destes oráculos e seus custos de execução são pouco explorados. De fato, em alguns casos nem sequer utiliza-se tal abstração e considera-se que a coleta de informações sobre o ambiente é feita pelo próprio protocolo adaptativo.

Em se tratando de serviços existentes que implementem oráculos, é possível encontrar na literatura de sistemas distribuídos alguns exemplos, em particular na área de monitoramento de redes e detecção de falhas. A seguir serão abordados alguns desses exemplos.

2.4.1 Oráculos em sistemas distribuídos

Oráculos para monitoramento de redes. Uma característica presente em muitos sistemas distribuídos (*e.g.*, sistemas par-a-par e grades computacionais) é a abundância de recursos disponíveis, permitindo dar preferência aos recursos que apresentem melhor desempenho. Ao mesmo tempo, estes ambientes, denominados de larga escala, são heterogêneos e dinâmicos, o que torna o desempenho dos seus recursos imprevisível. De fato, prever um desempenho adequado é um desafio para o desenvolvimento de aplicações em ambientes de larga escala. Nesse sentido, oráculos para o monitoramento de rede servem para diminuir tal imprevisibilidade fornecendo dados sobre o comportamento da rede ou previsões sobre o mesmo.

Um exemplo de oráculo em produção é o NWS (*Network Weather Service*) (WOLSKI; SPRING; HAYES, 1999) o qual fornece dados e previsões sobre o desempenho de um sistema. Os recursos monitorados incluem CPU (disponibilidade) e rede (latência e largura de banda). O NWS é composto por 4 componentes: servidor de nomes, memória, sensores e preditor. O servidor de nomes provê um serviço de diretórios que possibilita a associação de nomes a uma localização (número de porta ou endereços TCP/IP); os sensores coletam, periodicamente, informações sobre os recursos aos quais estão associados e enviam para a memória que funciona como um repositório de dados persistentes; os preditores fornecem estimativas sobre o desempenho dos recursos, aplicando modelos estatísticos de previsão sobre os dados armazenados na memória. Este oráculo pode ser usado em sistemas de grades computacionais para construir serviços de escalonamento eficientes (BERMAN et al., 2002; WOLSKI, 2003). O maior foco do trabalho sobre NWS refere-se às questões práticas de implementação, documentação, implantação e uso do oráculo, sendo assim, o objetivo

desta pesquisa é fornecer à comunidade uma ferramenta estável[9].

Outro trabalho no mesmo contexto do NWS foi proposto em (RASCHID et al., 2003; ZADOROZHNY et al., 2004) e corresponde a um modelo conceitual para representar atrasos fim-a-fim entre clientes e servidores em aplicações de larga escala. De acordo com o modelo, atrasos fim-a-fim são definidos através de distribuições de latência dependentes de tempo, denominadas de *Latency Profiles - LP*, e refletem as variações na comunicação entre os componentes da aplicação. Cada par cliente-servidor possui um *LP* cujos dados são coletados pelo cliente de forma contínua. Para tornar a solução escalável os autores sugerem agregar *LPs* usando informação mútua e correlação. Apesar de ser um trabalho mais teórico, a viabilidade da solução para monitoramento de redes é comprovada através de medições em uma plataforma de testes usando como aplicação um serviço de nomes global.

Ainda no contexto dos trabalhos citados acima tem-se as pesquisas sobre serviços de localização na rede, os quais encapsulam um oráculo de latência. De modo geral, estes trabalhos descrevem soluções para selecionar recursos baseado na localização dos mesmos na rede (idéia de proximidade na rede). Por exemplo, dois trabalhos recentes nessa área são o arcabouço Meridian (WONG; SILVKINS; SIRER, 2005) e a ferramenta Netvigator (SHARMA et al., 2006). Estimar proximidade na rede significa ordenar recursos de acordo com as suas distâncias (latências) em relação a outros recursos, nesse caso, uma estratégia comum é considerar os recursos inseridos em um espaço global de coordenadas Euclidiano, onde a distância entre quaisquer dois recursos é dada sem medições diretas, mas, através de medições entre estes recursos e alguns alvos fixos. Tal estratégia pode introduzir erros significativos nas estimativas (devido à escolha incorreta de valores para os parâmetros empregados, por exemplo), sendo considerada inexata e incompleta. No sentido de amenizar estas desvantagens pode-se considerar espaços de coordenadas locais e introduzir alvos criados dinamicamente como é feito na ferramenta Netvigator.

A ferramenta Netvigator responde a consultas sobre k recursos próximos e ainda provê estimativas de latência entre quaisquer dois recursos. O oráculo de latência usado no Netvigator guarda informações sobre tempo de ida e volta entre os recursos da rede e alvos fixos, como também, alvos dinâmicos. Estas informações são coletadas localmente por cada recurso e armazenadas de forma centralizada. O processo de seleção dos nodos próximos também ocorre de forma centralizada usando algoritmos específicos, os quais são definidos em função da distância entre os nodos, *i.e.*, do oráculo de latência. A definição do oráculo é dada através de uma função não dependente de tempo. A ferramenta foi

[9]Mais informações sobre obtenção, instalação e uso da ferramenta NWS em `http://nws.cs.ucsb.edu`.

avaliada por meio de simulação e medições em dois ambientes reais (PlanetLab e intranet de uma grande empresa), demonstrando eficiência para encontrar nodos próximos (90% de confiança nos resultados obtidos). É possível usar a ferramenta online para obter dados do PlanetLab (`http://www.networking.hpl.hp.com/cgi-bin/s3nvform-new.cgi`), entretanto, a ferramenta não está disponível para outros fins.

Por outro lado, o arcabouço Meridian realiza localização de recursos baseado em medições diretas. O arcabouço, que é genérico, foi aplicado a três classes de problemas: seleção de recursos próximos a um determinado alvo, seleção de um recurso próximo a k alvos e, seleção de recursos que satisfazem restrições de latência. Cada participante (recurso) do Meridian associa-se a um conjunto finito de outros participantes organizados em anéis concêntricos de acordo com valores de latência. Além disso, o arcabouço inclui um serviço de filiação a grupo para gerenciar os anéis e um serviço de disseminação para propagar mudanças nos anéis a todo o sistema. A localização de recursos é feita de forma distribuída: as consultas são roteadas através de membros dos anéis concêntricos usando informações diretas de latência, calculadas por cada membro durante a execução da consulta. Então, tem-se um oráculo de latência local a cada participante do Meridian. Por meio de um estudo analítico avaliou-se o desempenho do arcabouço em termos de exatidão (qualidade dos anéis concêntricos), escalabilidade (quantidade de passos para descobrir recursos próximos) e balanceamento de carga (quantidade de consultas para cada participante). Neste estudo, considerou-se duas definições para um oráculo de latência e ambas equivalem a funções não dependentes de tempo. Os resultados demonstram a eficiência do sistema. Também foram realizadas simulações e medições em um ambiente real (PlanetLab) cujos resultados confirmam aqueles obtidos no estudo analítico. Embora apresente vários atrativos, o Meridian ainda não é um sistema em produção.

Oráculos de detecção de falhas. Detectores de falhas são oráculos que fornecem informações sobre falhas no sistema, constituindo um importante bloco básico para sistemas distribuídos tolerante a faltas. Um resultado de grande relevância nessa área são os detectores de falhas não confiáveis para sistemas assíncronos (CHANDRA; TOUEG, 1996; LIN; HADZILACOS, 1999; MOSTEFAOUI; MOURGAYA; RAYNAL, 2002; LARREA; FERNáNDEZ; ARéVALO, 2000). Primeiramente, detectores de falhas não confiáveis possibilitam a resolução de problemas em sistemas assíncronos, os quais seriam impossíveis em sistemas puramente assíncronos. A segunda contribuição é sobre aspectos teóricos e práticos de oráculos. Na Seção 2.1.2 apresentou-se a definição formal de detectores de falhas não confiáveis; a seguir, enfatizam-se os benefícios destes oráculos para o desenvolvimento das

aplicações dependentes dos mesmos.

O fato de um detector de falhas não confiável ser definido em termos de propriedades abstratas (abrangência e exatidão) permite projetar protocolos simples e, consequentemente, mais fáceis de validar. Por outro lado, tal detector de falhas pode ser considerado uma "caixa-preta" à qual se tem acesso através de uma interface bem definida e com uma semântica precisa. Além disso, esta "caixa-preta" encapsula requisitos de sincronismo, os quais são essenciais para resolver determinados problemas em sistemas assíncronos. Estas características facilitam a portabilidade dos protocolos, pois a parte crítica da solução que os mesmos descrevem é encapsulada em um módulo bem definido. Pelas razões citadas acima, a abordagem de detectores de falhas não confiáveis descreve uma forma simples, modular e poderosa de projetar e implementar protocolos distribuídos tolerantes a faltas em sistemas assíncronos.

Realmente, a abordagem de detectores de falhas não confiável constitui um exemplo de abordagem modular funcional para sistemas distribuídos. De acordo com esta abordagem, um domínio de problema é decomposto em módulos, cada um deles com uma interface que define como o acesso aos mesmos é feito pelos outros módulos do sistema. Além disso, cada módulo tem uma especificação funcional precisa que é independente da implementação do módulo. Os benefícios da abordagem modular funcional recaem tanto sobre o campo prático (engenharia de software) quanto o campo teórico. Em um trabalho recente, Friedman e Raynal discutiram tais benefícios no contexto de problemas de acordo e sistemas par-a-par (FRIEDMAN; RAYNAL, 2004).

A Figura 2.2 (FRIEDMAN; RAYNAL, 2004) ilustra o projeto de um protocolo de consenso baseado em detectores de falhas não confiáveis. O módulo de consenso tem acesso ao detector de falhas (DF) através de uma função OBTEMSUSPEITOS() que retorna uma lista de processos suspeitos. A especificação funcional do módulo de detecção de falhas é definida por propriedades de abrangência e exatidão.

2.5 Conclusões parciais

Neste capítulo apresentou-se aspectos relevantes sobre adaptação em sistemas distribuídos heterogêneos e dinâmicos para fins de desempenho. Observou-se que os sistemas distribuídos modernos tendem a ser mais heterogêneos e dinâmicos, a exemplo da Internet e alguns ambientes de clusters. Por outro lado, tais características podem impactar o desempenho dos sistemas, tornado-os ineficientes. Para lidar com este problema, adaptação tem sido empregada com sucesso (HILTUNEN; SCHLICHTING, 1996; BRUSILOVSKY; MAYBURY,

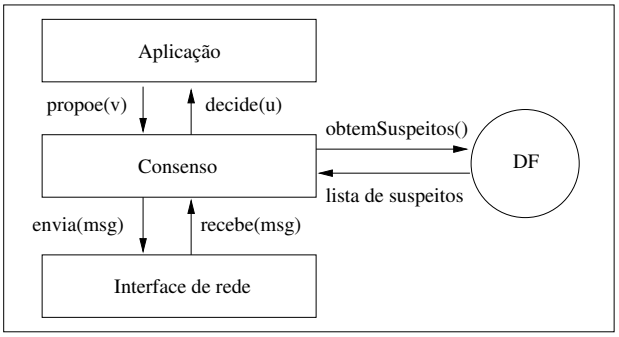


Figura 2.2: Visão modular do consenso em um sistema assíncrono com um detector de falhas não confiável

2002; BERMAN et al., 2002; SILVA; ENDLER; KON, 2003; RAVINDRAN; KWIAT; SABBIR, 2004).

Muitos pesquisadores têm contribuído com a área de adaptação em sistemas distribuídos, propondo metodologias, modelos e ferramentas de suporte que facilitem o projeto e implementação de sistemas adaptativos. Já as questões teóricas são pouco exploradas.

No contexto de protocolos distribuídos, pode-se encontrar na literatura alguns trabalhos sobre soluções adaptativas aplicadas aos mesmos. Em particular, tem-se interesse pelos protocolos distribuídos simétricos pelo texto, objeto de estudo desta tese. Nesse caso, protocolos distribuídos simétricos pelo texto em ambientes de carga heterogênea e dinâmica. A maioria dos trabalhos já publicados, enfoca aspectos práticos das soluções adaptativas propostas e resultados de desempenho associados às mesmas. Porém, aspectos práticos ainda são tratados de maneira informal, não valorizando princípios básicos da engenharia de software, tais como separação de conceitos e modularização. Uma observação importante refere-se ao uso de oráculos de latência, elementos do sistema responsáveis por informar sobre a situação de carga do ambiente de execução; tais informações são essenciais para construir mecanismos de adaptação. É possível encontrar trabalhos sobre oráculos de latência, de maneira isolada, e, assim, não associados a protocolos simétricos pelo texto adaptativos. Dessa forma, abstrai-se a idéia de oráculos de latência, considerando que as informações de latência estão disponíveis de alguma maneira e são utilizadas pelos mecanismos de adaptação para decidir sobre ações de adaptação.

Portanto, o uso de mecanismos de adaptação associados a oráculos de latência é um tema ainda não estudado na literatura. Acredita-se que oráculos de latência podem facilitar o projeto e implementação de mecanismos adaptativos. Além disso, a definição formal do oráculo de latência pode ser usada para estudar analiticamente o desempenho dos mecanismos adaptativos baseados nos mesmos, como será discutido nos próximos capítulos. Um

outro aspecto importante refere-se ao uso de adaptação em protocolos de consenso simétricos pelo texto para fins de desempenho, que constitui um domínio de problema ainda não explorado nesse contexto. Note que questões de desempenho do consenso são relevantes tanto pela sua importância em sistemas distribuídos quanto pela sobrecarga introduzida para resolvê-lo, tornando o consenso pouco eficiente por natureza.

Capítulo 3

Oráculos de latência em protocolos distribuídos

Oráculos são elementos fundamentais para construir protocolos distribuídos adaptativos por valor em ambientes sujeitos à carga heterogênea e dinâmica, sendo responsáveis por prover informações sobre a carga do ambiente que vão orientar as ações de adaptação no protocolo.

Oráculos deveriam ser definidos como componentes do sistema que encapsulam funcionalidades do interesse de outros componentes e que auxiliam (ou mesmo tornam possível) a computação realizada pelos mesmos. Nesse sentido, seria preciso atribuir-lhes uma semântica precisa, acompanhada de uma especificação formal, e definir interfaces de acesso para os seus clientes. Em outras palavras, valorizar os aspectos teóricos associados aos oráculos. Dessa forma, sistemas baseados em oráculos seriam construídos à luz de princípios da engenharia de software (*e.g.*, modularidade e separação de conceitos), tornando-se mais simples e fáceis de validar. Um dos principais exemplos de oráculos no contexto de sistemas distribuídos que obedecem aos requisitos citados acima são os detectores de falhas para ambientes assíncronos (CHANDRA; TOUEG, 1996) (ver Capítulo 2). No contexto de oráculos de latência não se observa o mesmo (SHARMA et al., 2006; WONG; SILVKINS; SIRER, 2005; ZADOROZHNY et al., 2004; WOLSKI; SPRING; HAYES, 1999). Aspectos teóricos de oráculos de latência têm sido pouco explorados, observando-se maior ênfase a questões de implementação. De fato, alguns trabalhos apresentam resultados nesse sentido, mas usam definições simplificadas para os oráculos (SHARMA et al., 2006; WONG; SILVKINS; SIRER, 2005) ou não exploram o poder da definição formal para analisar o comportamento dos protocolos que os utilizam antes de implementá-los (ZADOROZHNY et al., 2004).

O objetivo deste capítulo é explorar aspectos teóricos de oráculos de latência para pro-

tocolos distribuídos adaptativos. Por latência entenda-se atraso fim-a-fim na comunicação entre processos. O principal foco será a definição formal do oráculo e como usufruir deste formalismo para entender e obter comprovações iniciais sobre a influência da adaptação sobre o desempenho dos protocolos que utilizam o oráculo.

O restante desse capítulo está organizado da seguinte forma. Na Seção 3.1 descreve-se o modelo de sistema adotado para esta tese, o qual representa um sistema distribuído sujeito à carga heterogênea e dinâmica. Então, na Seção 3.2, apresenta-se a definição formal de um oráculo de latência. Em seguida, na Seção 3.3, discute-se sobre adaptação, por meio de oráculos de latência, em protocolos distribuídos simétricos pelo texto. Nesse caso, introduz-se uma solução adaptativa genérica para ordenação de processos, que constitui um problema presente em muitos protocolos simétricos pelo texto. Na Seção 3.4, descreve-se uma aplicação para a solução genérica proposta, a qual será usada como estudo de caso para demonstrar a importância da formalização do oráculo de latência na avaliação de desempenho dos protocolos adaptativos. A análise de desempenho analítica com base na definição formal do oráculo de latência é mostrada na Seção 3.5. Por fim, na Seção 3.6, apresentam-se os resultados deste capítulo e alguns comentários sobre os mesmos.

3.1 Modelo do sistema

Considera-se um modelo assíncrono acrescido de detectores de falhas da classe $\Diamond S$ (CHANDRA; TOUEG, 1996) (ver Capítulo 2). Nesse caso, o modelo consiste de um conjunto finito Π de n processos participantes, onde $\Pi = \{p_1, p_2, ..., p_n\}$ e $n > 1$. Um processo pode falhar por parada e processos falhos não se recuperam. O comportamento de um processo é correto (ou seja, de acordo com sua especificação) até que o mesmo, possivelmente, falhe. Apenas f, $f < n/2$, processos podem falhar.

Os processos se comunicam por meio de troca de mensagens através de canais confiáveis: não ocorrem criação, alteração, duplicação ou perda de mensagens na transmissão das mensagens pelo canal de comunicação. Os processos estão conectados via canais de comunicação unidirecionais, ou seja, p_i envia mensagens para p_j via o canal de comunicação c_{ij}, enquanto p_j envia mensagens para p_i via c_{ji}. Sendo assim, um processo p_i pode 1) enviar mensagens para outro processo p_j via c_{ij}; 2) receber mensagens de outro processo p_j via c_{ji}; 3) realizar alguma computação local; ou 4) falhar. Outra informação pertinente é que o grafo de comunicação é completo.

Não existe nenhuma suposição sobre a velocidade relativa de cada processo ou atraso na transmissão de mensagens. Entretanto, considera-se que os processos têm acesso a um

relógio local, o qual pode ser usado para medir a passagem do tempo. Além disso, cada processo tem acesso a um serviço de detecção de falhas da classe $\Diamond S$.

3.2 Formalizando um oráculo de latência

Um oráculo de latência é uma função *Lat* que retorna uma estimativa de latência entre dois processos p_i e p_j, no tempo t, a qual representa uma aproximação do atraso fim-a-fim na transmissão de uma mensagem de p_i para p_j no instante de tempo t. Dessa forma, $Lat : \Pi \times \Pi \times \mathbb{R}_+ \rightarrow \mathbb{R}_+$ é uma relação que associa cada tupla $(p_i, p_j, t) \in \Pi \times \Pi \times \mathbb{R}_+$ a $Lat(p_i, p_j, t) = L$, $L \in \mathbb{R}_+$, então:

$$\forall (p_i, p_j, t) \in \Pi \times \Pi \times \mathbb{R}_+, \exists L \in \mathbb{R}_+ : Lat(p_i, p_j, t) = L. \tag{3.1}$$

O estudo analítico sobre o desempenho de protocolos simétricos pelo texto adaptativos, baseados em oráculos de latência, pode ser considerado uma etapa preliminar no processo de avaliação de desempenho de tais protocolos. Seu principal objetivo é investigar a viabilidade de usar adaptação para fins de desempenho; por viabilidade entenda-se eficiência. Então, a importância deste estudo é no sentido de ajudar a obter comprovações iniciais acerca dos benefícios da adaptação sobre o desempenho dos protocolos que a utilizam. Nesse caso, o estudo analítico é fundamentado em um modelo de desempenho do protocolo adaptativo, descrito em função da definição formal do oráculo de latência.

O modelo de desempenho para o protocolo adaptativo representa a sequência de troca de mensagens (padrão de comunicação) entre os processos, desde o início até o término de uma execução do protocolo. Este modelo será parametrizado com diferentes configurações de troca de mensagens, e o custo de comunicação associado a cada configuração determinará o tempo de terminação de uma execução do protocolo. Por sua vez, a definição formal do oráculo de latência será usada para representar as estimativas de latência entre os participantes do protocolo no modelo de desempenho especificado. Dessa forma, o modelo de desempenho permite estimar o custo de comunicação para todas as possíveis configurações do protocolo, de modo que a configuração mais eficiente esteja relacionada ao menor custo de comunicação estimado.

Ao usar o modelo de desempenho especificado para decidir sobre as ações disparadas por um mecanismo de adaptação demonstra-se a eficiência de tal mecanismo, pois suas ações de adaptação sempre irão considerar configurações eficientes de acordo com a situação de carga corrente do sistema.

Nas próximas seções apresentam-se uma solução adaptativa genérica para protocolos

simétricos pelo texto dependentes de ordenação de processos, como também, uma aplicação para a mesma. Esta aplicação será usada como estudo de caso para a avaliação de desempenho analítica baseada na definição formal de oráculos de latência, mostrada em seguida. Note que a avaliação de desempenho analítica fundamenta-se na discussão sobre modelo de desempenho apresentada anteriormente.

3.3 Adaptação através de oráculos de latência em protocolos simétricos pelo texto

Como já foi enfatizado, protocolos simétricos pelo texto podem sofrer perdas de desempenho se usarem uma única configuração em ambientes sujeitos à carga heterogênea e dinâmica. Nesse contexto, adaptação baseada em oráculos de latência pode ser usada para garantir bons índices de desempenho para o protocolo em diferentes situações de carga do ambiente de execução.

Considere os protocolos simétricos pelo texto que seguem o paradigma do coordenador rotativo[1]. A principal característica destes protocolos é a existência de um processo que desempenha o papel de coordenador, enquanto os demais participantes do protocolo, simplesmente, cooperam com o mesmo para realizarem uma determinada tarefa. Normalmente, tais protocolos procedem em rodadas, onde cada rodada possui um coordenador conhecido por todos os participantes. Além disso, o padrão de comunicação entre processos é centralizado no coordenador, ou seja, mensagens são enviadas de/para o coordenador. No sentido de tolerar faltas do processo coordenador, os protocolos em questão dependem de ordenação consistente de processos[2]. Sendo assim, a escolha dos participantes que desempenham papéis "especiais" (coordenador) é feita através de uma lista ordenada de processos; quando o coordenador corrente falha, o próximo processo da lista é selecionado para assumir este papel.

Muitos dos protocolos simétricos pelo texto disponíveis na literatura ordenam os processos usando um mecanismo bastante simples (CHANDRA; TOUEG, 1996; LARREA; FERNáNDEZ; ARéVALO, 2000; RICCIARDI; BIRMAN, 1991; AGUILERA et al., 2001; HURFIN et

[1]Deste ponto em diante, protocolos simétricos pelo texto identificam os protocolos desta classe que são baseados no paradigma do coordenador rotativo.

[2]Em um trabalho inicial Sampaio e Brasileiro (SAMPAIO; BRASILEIRO, 2005) descreveram e formalizaram o problema da ordenação de processos usando uma idéia da engenharia de software, denominada de separação de conceitos. Esta estratégia contribui para entender melhor o conceito de ordenação de processos e como este influencia no comportamento dos protocolos dependentes do mesmo. Além disso, favorece o uso de uma abordagem modular no projeto e implementação de tais protocolos.

al., 1999; GUERRAOUI; LARREA; SCHIPER, 1995). Este mecanismo consiste em ordenar a lista a priori, *i.e.*, antes da execução do protocolo, com base nos identificadores dos processos. Em outras palavras, o mecanismo para ordenação de processos normalmente utilizado não é adaptativo e, portanto, pode causar degradação de desempenho no protocolo que o utiliza quando este executa em ambientes sujeitos à carga heterogênea e dinâmica. Dessa forma, argumenta-se que protocolos simétricos pelo texto podem obter ganhos de desempenho nesses ambientes quando equipados com uma solução adaptativa para ordenação de processos, como a que será descrita na próxima seção. Note que a solução é genérica e serve como base para diferentes implementações, além disso pode ser aplicada a diversos protocolos simétricos pelo texto.

3.3.1 Proposição de uma solução genérica para ordenação de processos adaptativa usando oráculos de latência

A descrição de uma solução adaptativa deve responder a três questionamentos (ver Capítulo 2): **o que** precisa ser adaptativo? 2) **quando** uma ação de adaptação será disparada? e 3) **como** ocorrerá a adaptação? Respondendo ao primeiro questionamento, deseja-se tornar adaptativa a função de ordenação de processos utilizada em um protocolo simétrico pelo texto. A função deve retornar a lista ordenada de processos que permitirá bons índices de desempenho para o protocolo em qualquer situação de carga do ambiente de execução. A lista de processos é fornecida no início da execução do protocolo. Então, ações de adaptação podem ocorrer a cada nova execução do protocolo, quando a ordem dos processos mudar em decorrência das variações na situação de carga do sistema (isto diz respeito ao segundo questionamento). Faz-se necessário decidir como representar a situação de carga do sistema. Considerando o terceiro questionamento, sempre que for iniciada uma nova execução do protocolo todos os processos requisitam uma lista ordenada de processos à função de ordenação. Esta função irá usar informações sobre a situação de carga do ambiente para estimar o desempenho do protocolo parametrizado com cada um dos possíveis valores de listas ordenadas de processos, então a função retornará a lista associada ao melhor desempenho do protocolo. Informações sobre a situação de carga do ambiente são fornecidas por um oráculo de latência. Portanto, o oráculo permite estimar o custo de comunicação para cada configuração (valor de lista ordenada de processos) do protocolo e, por conseguinte, seu desempenho.

Do ponto de vista do protocolo que usa ordenação de processos, a introdução de um mecanismo de adaptação deve ser transparente para o mesmo. Isto permitirá que os proto-

colos adaptativo e não-adaptativo sejam diferentes essencialmente na forma como a lista de processos é ordenada, mas seus algoritmos serão similares. As principais vantagens desta abordagem equivalem à 1) diminuição dos custos de implementação de protocolos adaptativos, pois pode-se utilizar soluções adaptativas já existentes (reutilização) com poucas modificações no protocolo original, além disso, 2) promove-se a separação de conceitos (aspectos funcionais e não-funcionais) e modularização, o que facilita a manutenção e evolução do protocolo adaptativo implementado, como também, da aplicação que o utiliza.

Outra característica da solução proposta é que esta permite adaptação por valor sobre parâmetros de comportamento, ou seja, as variações nas condições do ambiente de execução irão refletir sobre o conteúdo da lista ordenada de processos e, por conseguinte, sobre a configuração do coordenador de cada rodada (parâmetro de comportamento). De fato, a função que gera a lista ordenada de processos não muda quando ocorre uma adaptação, mas os resultados gerados pela mesma, os quais refletem a situação de carga do ambiente. Além disso, ações de adaptação ocorrem tanto no contexto da ordenação de processos, através da mudança na ordem dos processos, quanto no contexto das aplicações (protocolos simétricos pelo texto) que utilizam a solução proposta. Nesse último caso, a ação é escolher um coordenador diferente da execução anterior do protocolo.

A solução adaptativa proposta possui três requisitos funcionais: 1) atomicidade sobre o valor da lista ordenada de processos; 2) validade sobre os componentes da lista ordenada de processos e 3) terminação. O primeiro requisito garante que todos os processos terão acesso à mesma lista ordenada de processos para escolherem o mesmo coordenador em cada rodada do protocolo associado à solução. O segundo requisito garante que os componentes da lista serão os mesmos do conjunto de participantes do protocolo associado à solução. Já o terceiro requisito garante um tempo finito para cada consulta à solução. Todos estes requisitos estão associados à correção da solução adaptativa. Outro requisito da solução é adaptação, que constitui um requisito não-funcional. Note que ações de adaptação (mudanças na ordem da lista de processos) só serão possíveis se a solução para ordenação tiver acesso a informações atualizadas sobre a situação de carga do sistema e usar tais informações no sentido de escolher configurações eficientes para o protocolo. Por situação de carga entenda-se latência entre os processos. Tais informações são coletadas, periodicamente, por um oráculo de latência e disponibilizadas através de uma interface bem definida. Portanto, a responsabilidade do oráculo de latência na definição de uma solução adaptativa é fornecer informação sobre a carga do sistema, permitindo a elaboração de soluções adaptativas eficientes no sentido de executarem ações de adaptação que beneficiem o desempenho dos respectivos protocolos em qualquer situação de carga.

3.4 Ordenação de processos adaptativa em um protocolo de consenso simétrico pelo texto: *consenso-CT*

A solução adaptativa apresentada na seção anterior é genérica e pode ser aplicada a protocolos simétricos pelo texto, dependentes de ordenação de processos, em diferentes domínios de problemas, tais como acordo (CHANDRA; TOUEG, 1996; GUERRAOUI; RAYNAL, 2004; HURFIN et al., 2001, 1999; GUERRAOUI; LARREA; SCHIPER, 1995), detecção de falhas baseada em oráculo de liderança (LARREA; FERNáNDEZ; ARéVALO, 2000) e eleição de líder (AGUILERA et al., 2001). Deseja-se aplicar a solução proposta no contexto de acordo, mais especificamente, protocolos de consenso simétricos pelo texto.

Nesta seção descreve-se um estudo de caso sobre um protocolo de consenso para ambientes assíncronos equipado com detectores de falhas não-confiáveis, o qual foi denominado de *consenso-CT* (CHANDRA; TOUEG, 1996). A adaptação é sobre a lista ordenada de processos, através da qual são definidas as identidades dos processos que assumem o papel de coordenador em cada rodada do protocolo. No caso, o primeiro elemento da lista será o coordenador da primeira rodada do protocolo e, assim, sucessivamente. A opção pelo consenso tem duas motivações: 1) pela importância do problema do consenso no contexto de sistemas distribuídos e, 2) pelo fato de não existir na literatura exemplos de protocolos de consenso simétricos pelo texto adaptativos.

3.4.1 O problema do consenso

O problema do consenso envolve um grupo de processos, onde cada processo propõe um valor e todos os processos corretos devem decidir sobre um valor comum que será um dos valores propostos (GUERRAOUI; SCHIPER, 1997, 2001). Formalmente, o consenso é definido em termos de algumas propriedades, as quais devem ser garantidas pelas primitivas que implementam a solução para tal problema. Estas propriedades são as seguintes (FISCHER; LYNCH; PATERSON, 1985; CHANDRA; TOUEG, 1996):

- **Terminação**: todo processo correto, em algum momento, decide um valor para o consenso.
- **Validade**: se um processo decidir o valor v, então este valor foi proposto por algum processo.
- **Acordo uniforme**: quaisquer dois processos nunca decidem valores diferentes.

O consenso é considerado um bloco básico para a implementação de soluções de problemas de acordo em sistemas distribuídos, tais como confirmação atômica, difusão atômica e filiação a grupo (CHANDRA; TOUEG, 1996; TUREK; SHASHA, 1992; GALLENI; POWELL, 1996; GREVE; NARZUL, 2004; GREVE et al., 2001; HURFIN et al., 1999). O acordo, por sua vez, constitui uma importante abstração para construir sistemas distribuídos tolerante a faltas. O problema do consenso tem sido estudado no contexto de sistemas síncronos e assíncronos. Em particular, existem diferentes trabalhos sobre o consenso em sistemas assíncronos com detectores de falhas (LAMPORT, 2001; CHANDRA; TOUEG, 1996; GUERRAOUI; RAYNAL, 2004; GUERRAOUI et al., 2000; HURFIN et al., 2001).

3.4.2 O protocolo *consenso-CT*

Considere o modelo de sistema descrito na Seção 3.1. Um dos protocolos de consenso proposto por Chandra-Toueg em (CHANDRA; TOUEG, 1996) (*consenso-CT*) usa um serviço de detecção de falhas da classe $\Diamond S$ e requer que a maioria dos processos participantes estejam corretos, $n \geq 2f + 1$.

O protocolo *consenso-CT* é baseado no paradigma do coordenador rotativo e procede em rodadas assíncronas. Em cada rodada existe um processo que desempenha o papel de coordenador (ver Algoritmo 1 (CHANDRA; TOUEG, 1996)), sendo este conhecido por todos os processos. O coordenador de uma rodada r é o processo $c = (r \mod n) + 1$, onde c é o identificador do processo[3]. De acordo com o algoritmo, cada coordenador tenta obter consenso sobre um valor consistente e, então, decide pelo mesmo. Se o coordenador de uma rodada estiver correto e não for considerado suspeito por nenhum dos processos corretos, será obtido consenso sobre um determinado valor o qual será difundido a todos os processos participantes do protocolo através de algum protocolo de difusão confiável (CHANDRA; TOUEG, 1996). Caso contrário, o processo avança para a próxima rodada, onde será escolhido um novo coordenador.

O Algoritmo 1 é divido em duas etapas que executam em paralelo. Na primeira etapa, os processos se comunicam a fim de decidir sobre um valor consensual. A segunda etapa é responsável por garantir que o valor decidido em consenso será entregue por cada processo participante do protocolo, de forma confiável. A primeira etapa do consenso é composta de 4 fases, a saber:

- **Fase1:** cada processo participante envia seu valor (estimativa), identificado com o número da rodada na qual esta estimativa foi adotada (marca de tempo ts_p), para o

[3]Considere que o identificador c caracteriza o processo p_c.

Algoritmo 1 O protocolo *consenso-CT* executado pelo processo p_p

```
% Primeira etapa
when propose(v_p)
  estimate_p = v_p
  state_p = undecided
  r_p = ts_p = 0
  while state_p = undecided do
    r_p = r_p + 1
    c_p = (r_p  mod n) + 1
    % Fase 1: Envio das estimativas para o coordenador corrente
    send(p, r_p, estimate_p, ts_p) to c_p
    % Fase2: Envio da proposição do coordenador corrente
    if p == c_p then
      waituntil for ⌈(n + 1)/2⌉ processes q: received (q, r_p, estimate_q, ts_q) from q
      msgs_p[r_p] = {(q, r_p, estimate_q, ts_q) | p received (q, r_p, estimate_q, ts_q) from q}
      t = largest ts_p such that (q, r_p, estimate_q, ts_q) ∈ msgs_p[r_p]
      estimate_p = select one estimate_q such that (q, r_p, estimate_q, t) ∈ msgs_p[r_p]
      send(p, r_p, estimate_p) to all
    end if
    % Fase 3: Todos os processos esperam pela proposição do coordenador corrente
    waituntil received (c_p, r_p, estimate_{c_p}) from c_p or c_p ∈ D_p
    if received (c_p, r_p, estimate_{c_p}) from c_p then
      estimate_p = estimate_{c_p}
      ts_p = r_p
      send (p, r_p, ack) to c_p
    else
      send(p, r_p, nack) to c_p
    end if
    % Fase 4: Coordenador corrente verifica se decisão do consenso foi alcançada
    if p == c_p then
      waituntil for ⌈(n + 1)/2⌉ processes q: received (q, r_p, ack) or (q, r_p, nack)
      if for ⌈(n + 1)/2⌉ processes q: received (q, r_p, ack) then
        R_broadcast(p, r_p, estimate_p, decide)
      end if
    end if
  end while
  || % Segunda etapa
  when R_deliver(q, r_q, estimate_q, decide)
    if state_p = undecided then
      decide(estimate_q)
      state_p = decided
    end if
  end when
end when
```

coordenador da rodada corrente, p_c.

- **Fase2:** o coordenador p_c reúne, no mínimo, $\lceil (n+1)/2 \rceil$ das estimativas recebidas, escolhe uma destas e a envia para todos os processos como sendo uma nova proposta de valor consensual ou uma nova estimativa (a proposta do coordenador). Esta escolha deve respeitar um mecanismo de travamento que garante a propriedade de **Acordo uniforme** do problema do consenso. Tal mecanismo é baseado no valor da marca de tempo associada a cada estimativa recebida. Após receber um número majoritário de estimativas, o coordenador corrente deve escolher uma estimativa cuja marca de tempo possua o maior valor dentre as estimativas recebidas. Este valor é também usado para atualizar a estimativa do coordenador corrente.

- **Fase 3:** cada participante espera até receber a proposta enviada pelo coordenador. Para evitar a possibilidade de bloqueio devido a falhas do coordenador, os processos consultam seus oráculos de detecção de falhas constantemente a fim de obter informações sobre o estado do processo p_c, coordenador da rodada corrente. Se o coordenador é considerado suspeito por um processo, este envia uma mensagem de confirmação negativa para o coordenador (note que uma suspeita não significa que o coordenador realmente falhou). Por outro lado, se um processo recebe a proposta do coordenador, ele adota esta nova estimativa como sua própria estimativa e envia uma mensagem de confirmação positiva para o coordenador.

- **Fase 4:** o coordenador p_c espera receber, no mínimo, $\lceil (n+1)/2 \rceil$ respostas (positivas ou negativas). Caso todas as respostas sejam positivas, o coordenador difunde uma requisição aos outros processos a fim de que estes possam adotar a proposta do coordenador como valor de decisão do consenso. Quando um processo receber esta requisição, o consenso será decidido pelo mesmo corretamente. Os processos são informados da decisão do consenso através da execução de um protocolo de difusão confiável. Um processo correto finaliza a execução do protocolo quando recebe e processa o valor decidido em consenso (proposta do coordenador).

3.4.3 Aplicando a solução adaptativa genérica para ordenação de processos no *consenso-CT*

Como foi dito anteriormente, não existe na literatura trabalhos sobre adaptação em protocolos de consenso simétricos pelo texto (e dependentes de ordenação de processos). Nesse

caso, a ordem dos processos na lista é definida a priori, *i.e.*, antes da execução do protocolo, com base nos identificadores dos processos.

No protocolo *consenso-CT* (CHANDRA; TOUEG, 1996), a lista ordenada de processos é implementada através de uma função pré-definida, $f(r, n) = (r \mod n) + 1$, especificada em termos do número da rodada corrente (r) e da quantidade de processos participantes do protocolo (n). Então, para cada rodada r do protocolo existe um coordenador pré-definido, conhecido por todos os participantes do protocolo. Em outras palavras, considera-se que a lista de processos é ordenada de acordo com os identificadores dos processos. Seja $\Pi_{e1} = \{p_1, p_2, p_3\}$ os processos participantes de uma determinada instância do protocolo, então a lista ordenada de processos acessada por cada p_i é dada por $l = \{p_2, p_3, p_1, p_2, p_3, p_1, ...\}$. Para $r = 1$, $f(1, 3) = 2$, sendo assim, o coordenador da primeira rodada desta instância do *consenso-CT* é o processo p_2 e assim sucessivamente. É importante ressaltar que todos os participantes do protocolo escolhem o mesmo coordenador para cada rodada r do protocolo. Note que os elementos da lista ordenada de processos se repetem seguindo o mesmo padrão, o qual é definido pela função $f(r, n)$. A quantidade de elementos da lista l é finito, porém, desconhecido; por esta razão, a lista é representada por um conjunto infinito. Uma forma de representar a lista ordenada de processos de forma finita é através do padrão de repetição, considerando uma varredura circular. Assim, a lista l seria representada pelo conjunto $\{p_2, p_3, p_1\}$.

O problema da solução descrita acima é que a escolha da lista ordenada de processos não considera as condições do ambiente ao longo da execução do protocolo. Sendo assim, em situações de carga heterogênea e dinâmica (típico cenário para sistemas distribuídos reais), a lista ordenada de processos pode se tornar inadequada, causando perdas de desempenho ao protocolo. Isto porque, em protocolos simétricos pelo texto, os processos que desempenham papéis "especiais" influenciam no desempenho do protocolo como um todo, consequentemente, quando tais processos estão lentos, a execução do protocolo torna-se lenta. Portanto, a solução em questão, apesar de ser simples não é eficiente.

A solução adaptativa descrita na Seção 3.3.1 pode ser aplicada ao *consenso-CT* a fim de garantir bons índices de desempenho para o mesmo. Nesse caso, a escolha da lista ordenada de processos usada em cada execução do protocolo deve considerar a situação de carga do ambiente de execução. Uma lista eficiente (ou configuração eficiente) é aquela onde coordenadores mais "rápidos" são preferidos em relação aos mais "lentos" diminuindo o tempo de terminação do consenso; os processos são dispostos na lista em ordem decrescente de rapidez para uma dada situação de carga do ambiente. Rapidez ou lentidão são descritas a partir das informações providas pelo oráculo de latência. Então, o mecanismo de orde-

nação deve 1) construir o modelo de desempenho do protocolo a partir do seu padrão de comunicação (fluxo de mensagens), o qual é dado em função da identidade do coordenador (parâmetro de comportamento); 2) identificar as possíveis configurações de lista ordenada de processos para uma execução do protocolo, ou seja, os possíveis valores para o parâmetro de comportamento e; 3) usar o oráculo de latência para estimar o desempenho do protocolo, considerando o seu modelo de desempenho parametrizado com cada uma das configurações identificadas.

No *consenso-CT*, a comunicação entre os processos é centralizada no coordenador, ou seja, todas as mensagens (exceto a decisão do consenso) são enviadas de/para o coordenador de cada rodada. Seja uma rodada onde não existem suspeitas sobre o coordenador, então tem-se o seguinte padrão de comunicação: (passo 1) todos os participantes do protocolo enviam estimativas para o coordenador da rodada, (passo 2), o coordenador envia sua proposta de estimativa para os demais participantes a partir das estimativas recebidas, (passo 3) os participantes enviam mensagem de aviso ao coordenador aceitando a proposta sugerida pelo mesmo, (passo 4) o coordenador inicia difusão confiável da proposta de estimativa aceita por uma maioria de participantes.

3.5 Estudo analítico sobre o desempenho do protocolo *consenso-CT* adaptativo usando oráculos de latência

Nesta seção apresenta-se um estudo analítico sobre o desempenho do *consenso-CT* usando como métrica o tempo de execução do protocolo. Considera-se uma versão adaptativa do protocolo equipada com um oráculo de latência e um mecanismo de adaptação que fornece a lista ordenada de processos usada em cada execução do consenso (ver descrição da solução adaptativa na Seção 3.4.3), além disso, existe um protocolo de difusão confiável cuja finalidade é garantir que todos os participantes do consenso irão obter o valor consensual decidido. Adaptação serve para tornar a lista de processos sensível às variações de carga do ambiente de execução. Nesse caso, para cada execução do protocolo é fornecida uma lista ordenada de processos de acordo com as informações providas pelo oráculo de latência. A lista ordenada de processos determina os papéis assumidos por cada participante ao longo da execução do protocolo, portanto, adaptação no *consenso-CT* ocorre através da reconfiguração do seu parâmetro de comportamento.

O objetivo da análise é mostrar que usando um oráculo de latência é possível estimar o tempo de execução do consenso a priori para diferentes configurações (valores para a lista ordenada de processos), favorecendo a escolha de configurações eficientes para uma dada

execução do protocolo. Consequentemente, o mecanismo de adaptação baseado em um oráculo de latência será eficiente no sentido de escolher uma configuração do protocolo que permitirá bons índices de desempenho para qualquer execução do mesmo.

Por questões de simplicidade, neste estudo analítico sobre o tempo de execução do *consenso-CT* assume-se execuções do protocolo em cenários sem falhas ou falsas suspeições[4]. Note que tais cenários ocorrem com frequência na prática e constituem, assim, uma suposição realista para a análise realizada. Ao longo deste estudo analítico utiliza-se a definição de oráculo de latência apresentada na Seção 3.2, assim como, o modelo de sistema descrito na Seção 3.1. A lista de processos fornecida pelo mecanismo de adaptação para cada execução do consenso é nomeada de l_c e representada de forma finita, usando um padrão de repetição com varredura circular (como definido na Seção 3.4.3). Além disso, adota-se uma notação específica para representar variáveis de tempo, como é descrito na Tabela 3.5. Nesse caso, considera-se tempo real, abstraindo questões de sincronização de relógios.

Notação	**Descrição**
$t_{e(r)}^{p_i,l_c}$	tempo real no qual o evento $e(r)$ ocorre em p_i, dentro da rodada r do protocolo, em uma execução com a lista ordenada l_c.
$T_{e_1(r),e_2}^{p_i,l_c}$	duração do intervalo de tempo real $[t_{e_1(r)}^{p_i,l_c}, t_{e_2}^{p_i,l_c}]$.
$T_{e_1(r),e_2}^{\Pi,l_c}$	duração de tempo real entre as ocorrências dos eventos $e_1(r)$ e e_2 disparados por $\forall p \in \Pi$, ou seja, duração do intervalo de tempo real $[t_{e_1(r)}^{p_i,l_c}, t_{e_2}^{p_j,l_c}]$, $p_i, p_j \in \Pi$.

Tabela 3.1: Notação para o estudo analítico sobre o tempo de execução do *consenso-CT*

O protocolo *consenso-CT* procede em rodadas assíncronas e cada rodada possui um coordenador, conhecido pelos demais participantes (ver descrição do protocolo na Seção 3.4.2). Os coordenadores são escolhidos seguindo a lista ordenada de processos. O protocolo termina quando todos os participantes obtêm um valor consensual proposto pelo coordenador de uma rodada e acordado, pelo menos, por uma maioria de processos corretos. Então, o tempo de execução do *consenso-CT* pode ser definido formalmente através da duração de tempo entre os eventos de recebimento de uma requisição de consenso na primeira rodada

[4]Em cenários com falhas ou falsas suspeições, os participantes do *consenso-CT* executam um número finito, porém desconhecido, de rodadas até alcançarem um valor consensual. Isto dificulta a estimativa do tempo de execução do consenso de forma analítica, pois seria necessário incluir na análise o comportamento do detector de falhas utilizado.

do protocolo e obtenção de um valor consensual, os quais sinalizam, respectivamente, o início e o término de um consenso.

Os eventos de início e término do consenso podem ter escopo local e/ou global, por exemplo, existe o início do consenso tanto para p_i (local) quanto para o conjunto Π (global), enquanto o término de um consenso tem escopo local. Da mesma forma, existe a duração do consenso para p_i (local) e para o conjunto Π (global). Considere, inicialmente, a duração do consenso para p_i, nesse caso, utiliza-se o tempo inicial global de consenso e o tempo final de consenso (não precisa especificar o escopo pois existe apenas um). O tempo inicial local de um consenso é o tempo real no qual p_i recebe uma requisição de consenso. Por outro lado, o tempo inicial global equivale ao menor tempo no qual um processo recebe a requisição de consenso. Para obter tal informação, é preciso conhecer as latências entre a aplicação (entidade externa requisitante do serviço de consenso) e os participantes do consenso. Uma solução seria fazer o oráculo de latência prover estas informações o que requer a modelagem da aplicação neste estudo de desempenho do *consenso-CT*. Entretanto, optou-se por abstrair a aplicação, assim, as requisições de consenso são modeladas como mensagens instantâneas com latência nula. Então, uma requisição de consenso é recebida no mesmo instante de tempo real por qualquer participante. Isto significa que o tempo inicial local do consenso é o mesmo para qualquer participante e, consequentemente, corresponde ao menor valor dentre aqueles observados no conjunto Π. Portanto, pode-se assumir a equivalência entre os tempos iniciais de consenso local e global para qualquer p_i. Ao longo do texto será utilizado o termo tempo inicial do consenso para designar tanto o escopo local quanto global.

Seja $t^{p_i,l_c}_{inicialC(1)}$ e t^{p_i,l_c}_{finalC} os limites do intervalo de duração do consenso para um processo p_i, usando a lista ordenada l_c, os quais representam, respectivamente, o tempo inicial e o tempo final do consenso. Então, o tempo transcorrido $T^{p_i,l_c}_{inicialC(1),finalC}$ até p_i obter um valor consensual é dado pela Equação 3.2.

$$T^{p_i,l_c}_{inicialC(1),finalC} = t^{p_i,l_c}_{finalC} - t^{p_i,l_c}_{inicialC(1)}. \tag{3.2}$$

Um processo p_i decide o consenso por difusão confiável em duas situações, a saber:

1. p_i executa a rodada r e recebe a difusão confiável iniciada por ele mesmo como coordenador desta rodada, após obter um número majoritário de confirmações para sua proposta de valor consensual;

2. p_i executa a rodada r e recebe a difusão confiável iniciada pelo coordenador de uma rodada r', $r \neq r'$.

Como em todo protocolo baseado em rodadas assíncronas, os processos participantes do *consenso-CT* podem executar rodadas diferentes durante a execução do protocolo e, consequentemente, alcançarem um valor consensual em paralelo, o que aumenta as chances de algum processo decidir pela situação (2) apontada acima. Portanto, a partir das situações (1) e (2) originam-se várias possibilidades para um processo p_i decidir o consenso e, assim, diferentes valores para o tempo de execução do mesmo. Especificamente, a variação ocorre sobre o tempo no qual o valor consensual é obtido (t^{p_i,l_c}_{finalC}) já que o tempo inicial do consenso ($t^{p_i,l_c}_{inicialC(1)}$) é invariável. As possibilidades de duração do consenso para p_i são agrupadas no conjunto $EXEC\_LOCAL_{p_i,l_c}$ definido pela Equação 3.3. Nesse caso, o conjunto $FINALC_{p_i,l_c}$ reúne as opções de tempo final do consenso, ou seja, valores para a variável t^{p_i,l_c}_{finalC}.

$$\begin{aligned} EXEC\_LOCAL_{p_i,l_c} = \{T^{p_i,l_c}_{inicialC(1),finalC} \mid T^{p_i,l_c}_{inicialC(1),finalC} \in \mathbb{R}_+, \\ T^{p_i,l_c}_{inicialC(1),finalC} = t^{p_i,l_c}_{finalC} - t^{p_i,l_c}_{inicialC(1)}, \\ \forall (t^{p_i,l_c}_{finalC}, r_{finalC}) \in FINALC_{p_i,l_c}\}. \end{aligned} \tag{3.3}$$

O desempenho do *consenso-CT* para uma configuração de lista ordenada de processos l_c corresponde à menor duração do consenso obtida por qualquer participante do protocolo, nesse caso, a métrica usada é o k-ésimo menor tempo de execução, onde $1 \leq k \leq |\Pi|$. Seja $EXEC\_GLOBAL_{\Pi,l_c}$ o conjunto das menores durações de consenso para todos os processos $p_i \in \Pi$, então o desempenho do *consenso-CT* parametrizado com l_c é representado pela Equação 3.4.

$$\begin{aligned} T^{\Pi,l_c}_{inicialC(1),finalC} = PegaItemPosId(EXEC\_GLOBAL_{\Pi,l_c}, k) \quad onde, \\ EXEC\_GLOBAL_{\Pi,l_c} = \{(T^{p_i,l_c}_{inicialC(1),finalC}, p_i) \mid (T^{p_i,l_c}_{inicialC(1),finalC}, p_i) \in \mathbb{R}_+ \times \mathbb{N}, \\ T^{p_i,l_c}_{inicialC(1),finalC} = PegaItemPos(EXEC\_LOCAL_{p_i,l_c}, 1), \\ \forall p_i \in \Pi\}. \end{aligned} \tag{3.4}$$

A função *PegaItemPos* retorna o i-ésimo menor elemento de um conjunto de valores reais, sendo representada por $PegaItemPos : P(\mathbb{R}_+) \times \mathbb{N} \rightarrow \mathbb{R}_+$, onde $P(\mathbb{R}_+) = \{A | A \subseteq \mathbb{R}_+\}$ é o conjunto potência de $\mathbb{R}_+$, *i.e.*, o conjunto dos subconjuntos de $\mathbb{R}_+$. A função *PegaItemPos* associa cada tupla $(A, i) \in P(\mathbb{R}_+) \times \mathbb{N}$ a um valor real a, $a \in \mathbb{R}_+ \wedge a \in A$, dessa forma:

$$\begin{aligned} PegaItemPos(A, i) = a : a \in A, i = |Min\_a| \quad onde, \\ Min\_a = \{x \mid x \in A, x \leq a\}. \end{aligned} \tag{3.5}$$

A função $PegaItemPosId$ é uma variação de $PegaItemPos$ cuja diferença está no conjunto origem A. Nesse caso, o conjunto origem será representado por $B = \{(y, id) \mid y \in \mathbb{R}_+, id \in \mathbb{N}\}$, onde, B é ordenado de forma crescente a partir do valor de y e, em caso de empate, utiliza-se o valor de id. Dessa forma, a função $PegaItemPosId$ retorna um valor $b' \in \mathbb{R}_+$, associado ao i-ésimo menor elemento de B, (y, id), tal que, $b' = y$ e $(y, id) \in B$, assim:

$$\begin{aligned} PegaItemPosId(B, i) = b' : b' = y, (y, id) \in B, i = |Min\_(y, id)| \quad onde, \\ Min\_(y, id) = \{(y_i, id_i) \mid (y_i, id_i) \in B, y_i \leq y, id_i \leq id\}. \end{aligned} \tag{3.6}$$

Os valores possíveis para a lista ordenada de processos utilizada em uma execução do consenso resultam das permutações dos participantes do protocolo. Seja $PE(n)$ o conjunto de todas as permutações para Π, onde $n = |\Pi|$ e $|PE(n)| = n!$, assim, o desempenho do *consenso-CT* para todas as configurações possíveis da lista ordenada de processos pode ser representado pelo conjunto $EXEC\_GLOBAL_{\Pi,PE(n)}$ definido através da Equação 3.7.

$$\begin{aligned} EXEC\_GLOBAL_{\Pi,PE(n)} = \{(T^{\Pi,l_c}_{inicialC(1),finalC}, l_c) \mid (T^{\Pi,l_c}_{inicialC(1),finalC}, l_c) \in \mathbb{R}_+ \times P(\mathbb{N}), \\ T^{\Pi,l_c}_{inicialC(1),finalC} = PegaItemPosLista(EXEC\_GLOBAL_{\Pi,l_c}, k), \\ \forall l_c \in PE(n), 1 \leq k \leq |\Pi|\}. \end{aligned} \tag{3.7}$$

A função $PegaItemPosLista$ é uma variação de $PegaItemPosId$ cuja diferença está no tipo de tuplas que compõem o conjunto origem B. Nesse caso, o conjunto origem será representado por $C = \{(y, lista) \mid y \in \mathbb{R}_+, lista \in P(\mathbb{N})\}$, onde, C é ordenado de forma crescente a partir do valor de y e, em caso de empate, utiliza-se o valor de $lista$. Note que a idéia de ordem crescente aplicada a valores do tipo $\mathbb{R}_+$ não é a mesma aplicada a valores do tipo $P(\mathbb{N})$, então é preciso definir um critério de ordenação específico. Dessa forma, a função $PegaItemPosLista$ retorna um valor $c' \in \mathbb{R}_+$, associado ao i-ésimo menor elemento de C, $(y, lista)$, tal que, $c' = y$ e $(y, lista) \in C$, como mostra a equação 3.8. Por questões de simplicidade, utilizou-se o sinal $\leq$ para comparar os elementos $lista_i$ e $lista$; na prática, seria necessário usar uma outra função, de mesmo propósito, porém, capaz de suportar este tipo de elemento.

$$\begin{aligned} PegaItemPosLista(C, i) = c' : c' = y, (y, lista) \in C, i = |Min\_(y, lista)| \quad onde, \\ Min\_(y, lista) = \{(y_i, lista_i) \mid (y_i, lista_i) \in C, y_i \leq y, lista_i \leq lista\}. \end{aligned} \tag{3.8}$$

Para concluir a análise é preciso conhecer o tempo final de consenso, o qual depende dos atrasos fim-a-fim na comunicação entre os processos durante a execução do protocolo. Sendo assim, o tempo final do consenso é dado em função da definição do oráculo de latência. A seguir será definido o tempo final de consenso para um processo p_i ($t_{finalC}^{p_i,l_c}$) usando o oráculo de latência especificado através da Equação 3.1, além disso, apresenta-se a definição do conjunto $FINALC_{p_i,l_c}$ o qual reúne todos os valores possíveis de $t_{finalC}^{p_i,l_c}$.

A definição de $t_{finalC}^{p_i,l_c}$ depende do tempo de ocorrência de determinados eventos relacionados às execuções dos protocolos de consenso e de difusão confiável. Seja $p_c(r)$ a identificação do coordenador da rodada r, $p_c(r) \in \Pi$, a lista dos eventos e seus respectivos tempos de ocorrência são descritos abaixo:

e1=propostaC(r) $p_c(r)$ define sua proposta de valor consensual após receber as estimativas dos demais participantes — $t_{e1}^{p_c(r),l_c} = t_{propostaC(r)}^{p_c(r),l_c}$;

e2=valorC(r) $p_c(r)$ decide o valor consensual após receber confirmações para o valor proposto pelo mesmo — $t_{e2}^{p_c(r),l_c} = t_{valorC(r)}^{p_c(r),l_c}$;

e3=inicialDC(r) $p_c(r)$ inicia difusão confiável (DC) do valor consensual decidido para os demais participantes do consenso — $t_{e3}^{p_c(r),l_c} = t_{inicialDC(r)}^{p_c(r),l_c}$;

e4=finalDC(r) a difusão confiável iniciada por $p_c(r)$ é recebida, pela primeira vez, pelos participantes do protocolo p_i, $p_i \in \Pi$ — $t_{e4}^{p_i,l_c} = t_{finalDC(r)}^{p_i,l_c}$.

Os eventos **e3** e **e4** representam para cada p_i o início e término do protocolo de difusão confiável iniciado por $p_c(r)$. Seja $t_{inicialDC(r)}^{p_c(r),l_c}$ e $t_{finalDC(r)}^{p_i,l_c}$ os respectivos tempos de ocorrência destes eventos, então o tempo de execução de uma difusão confiável corresponde à duração do intervalo de tempo $[t_{inicialDC(r)}^{p_c(r),l_c}, t_{finalDC(r)}^{p_i,l_c}]$ e é representada por $T_{inicialDC(r),finalDC(r)}^{p_i,l_c}$. Durante a execução do consenso, o início de uma difusão confiável está condicionado à ocorrência do evento **e2**, de fato, o coordenador de uma rodada inicia uma difusão confiável imediatamente depois do evento **e2** acontecer, então pode-se considerar $t_{inicialDC(r)}^{p_c(r),l_c} = t_{valorC(r)}^{p_c(r),l_c}$. Por outro lado, os eventos **e1** e **e2** ocorrem durante a execução do protocolo de consenso e estão condicionados ao recebimento de um número majoritário de mensagens de um determinado tipo. A ocorrência do evento **e1**, no tempo $t_{propostaC(r)}^{p_c(r),l_c}$, está associada ao recebimento de estimativas do valor consensual (**Fase1** do *consenso-CT*), sendo estas provenientes dos participantes do protocolo com destino ao coordenador da rodada **r**, que as utiliza para compor sua proposta de valor consensual. Já a ocorrência do evento **e2**, no tempo $t_{valorC(r)}^{p_c(r),l_c}$, está associada ao recebimento de confirmações para a proposta de valor consensual enviada pelo coordenador da rodada **r** (**Fase4** do *consenso-CT*).

Como foi discutido acima, a ocorrência dos eventos **e1** e **e2** (por conseguinte, **e3**) em $p_c(r)$, na rodada r, depende do recebimento de um número majoritário de mensagens as quais serão enviadas por processos executando a rodada r. Para tal, faz-se necessária a existência de, pelo menos, $\lceil (N+1)/2 \rceil$ processos, $N = |\Pi|$, participando da rodada r. A razão é que, em cenários sem falhas ou falsas suspeições, o coordenador de uma rodada fica "travado" na mesma até decidir com difusão confiável iniciada pelo mesmo (situação (1)) ou pelos respectivos coordenadores de outras rodadas (situação (2)). Então, um processo p_i pode participar de todas as rodadas r', $1 \leq r' \leq r$, sendo r a rodada coordenada por p_i e $r \leq N$. Nesse caso, $r = 1$ engloba N participantes, $r = 2$ engloba $N-1$ participantes e, assim, sucessivamente até $r = N$ com, apenas, 1 participante que corresponde ao coordenador desta rodada. Vale ressaltar que $r = \lfloor (N+1)/2 \rfloor$ engloba $\lceil (N+1)/2 \rceil$ participantes. Dessa forma, os eventos **e1**, **e2** e **e3** ocorrem em qualquer rodada r, onde r tenha, pelo menos, $\lceil (N+1)/2 \rceil$ participantes, ou seja, $1 \leq r \leq \lfloor (N+1)/2 \rfloor$.

A Figura 3.1 ilustra a ocorrência dos eventos **e1**, **e2**, **e3** e **e4** até a decisão do consenso, envolvendo dois processos p e q. No caso, a relação de precedência na linha do tempo entre os eventos é dada por: $t^{p_c(r),l_c}_{propostaC(r)} \rightarrow t^{p_c(r),l_c}_{valorC(r)} \rightarrow t^{p_c(r),l_c}_{inicialDC(r)} \rightarrow t^{p_i,l_c}_{finalDC(r)}$, onde, $t^{p_c(r),l_c}_{valorC(r)} = t^{p_c(r),l_c}_{inicialDC(r)}$. Note que o consenso termina para um processo quando este recebe um valor consensual por difusão confiável. Portanto, o tempo final de consenso para qualquer $p_i \in \Pi$ equivale ao tempo final da difusão confiável iniciada pelo coordenador de alguma rodada do protocolo, ou seja, $t^{p_i,l_c}_{finalC} = t^{p_i,l_c}_{finalDC(r)}$.

Seja $DC_{p_i,r}$ o conjunto dos valores para o tempo no qual p_i recebe, pela primeira vez, uma difusão confiável iniciada pelo coordenador da rodada r, $1 \leq r \leq \lfloor (N+1)/2 \rfloor$, então o conjunto $FINALC_{p_i,l_c}$ com os tempos finais de consenso para p_i é definido pela Equação 3.9.

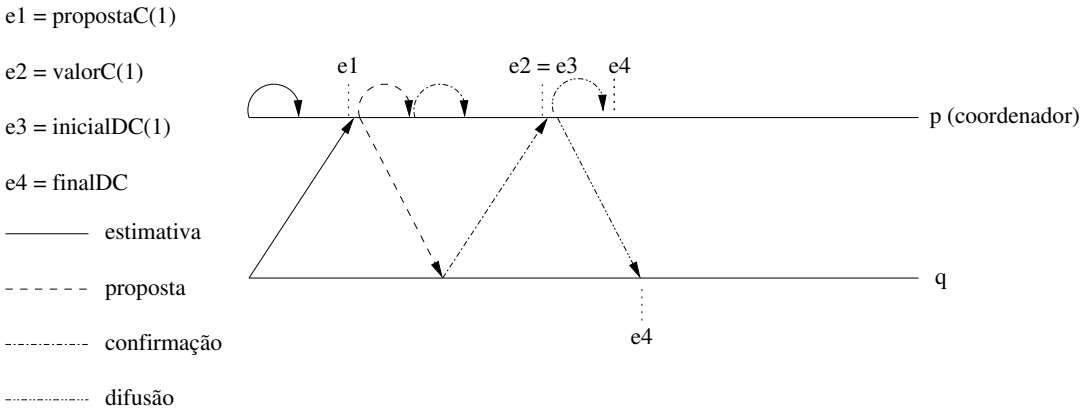


Figura 3.1: Ordem de ocorrência dos eventos associados ao protocolo de consenso e difusão confiável

$$FINALC_{p_i,l_c} = \{(t^{p_i,l_c}_{finalC}, r_{finalC}) \mid (t^{p_i,l_c}_{finalC}, r_{finalC}) \in \mathbb{R}_+ \times \mathbb{N}, \\ t^{p_i,l_c}_{finalC} = PegaItemPos(DC_{p_i,r}, 1), r_{finalC} = r, \\ \forall r, 1 \leq r \leq \lfloor (N+1)/2 \rfloor\}. \tag{3.9}$$

Neste estudo analítico sobre o desempenho do *consenso-CT* considera-se o protocolo de difusão confiável proposto em (CHANDRA; TOUEG, 1996). De acordo com este protocolo, uma mensagem é disseminada a todos os participantes, os quais, ao receberem uma mensagem pela primeira vez, devem retransmití-la aos destinatários da mesma. Por exemplo, considere $\Pi_{e2} = \{p_1, p_2, p_3, p_4, p_5\}$ o conjunto de participantes do consenso e da difusão confiável. A difusão de uma mensagem m_1 iniciada por p_1 é repetida pelos outros participantes quando estes recebem m_1 pela primeira vez, assim, m_1 é difundida 5 vezes. De modo geral, cada mensagem será difundida N vezes, onde $N = |\Pi_{e2}|$. Dessa forma, existem $N-1$ possibilidades para um processo p_i receber uma difusão confiável pela primeira vez, exceto se p_i tiver iniciado a difusão. Isto pode ser entendido como se a difusão confiável tivesse nl, $1 \leq nl \leq N-1$, níveis onde,

nl=1 identifica os processos pn_1 os quais recebem a difusão, pela primeira vez, diretamente do processo que a iniciou;

nl=2 identifica os processos pn_2 os quais recebem a difusão, pela primeira vez, de pn_1;

...

nl=N-1 identifica os processos pn_{N-1} os quais recebem a difusão, pela primeira vez, de pn_{N-2}.

Seja o exemplo acima, no qual 5 processos participam do consenso e da difusão confiável, assumindo que p_1 inicia a difusão. As possibilidades de p_2 receber, pela primeira vez, a mensagem difundida são representadas através da Figura 3.2; o mesmo procedimento se aplica aos demais participantes. Note que p_2 pode receber a mensagem difundida em qualquer um dos 4 níveis identificados, onde, em cada nível, a mensagem percorre um determinado caminho até alcançar seu destinatário. De fato, podem existir uma ou mais opções de caminhos por nível, pois a representação pn_{nl} serve para todos os processos que recebem a difusão, pela primeira vez, no nível nl. A Figura 3.3 ilustra tais caminhos para o exemplo em questão. Nesse caso, para cada nível, um caminho está associado ao tempo real no qual a difusão pode ser recebida, como será discutido a seguir.

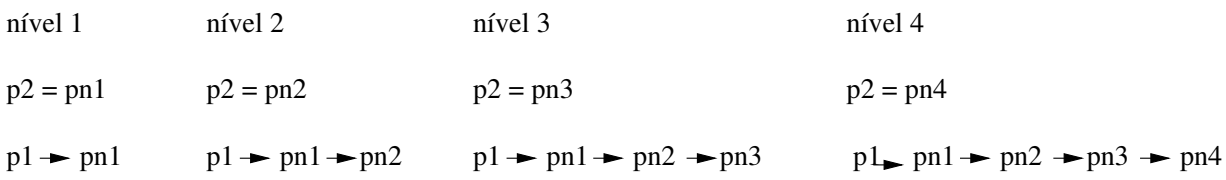


Figura 3.2: Conjunto de possibilidades para o processo p_2 receber, pela primeira vez, a difusão confiável iniciada por outro processo p_1

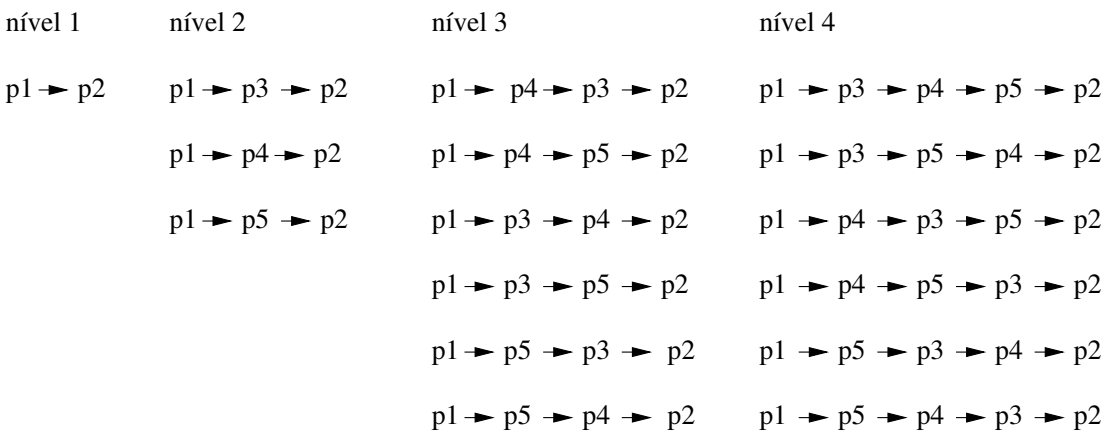


Figura 3.3: Detalhamento do conjunto de possibilidades para o processo p_2 receber, pela primeira vez, a difusão confiável iniciada por outro processo p_1

O tempo final de uma difusão confiável, iniciada na rodada r, para p_i ($t^{p_i,l_c}_{finalDC(r)}$) é o menor valor do conjunto $DC_{p_i,r}$, que contém todas as possibilidades de difusão confiável (caminhos) em cada um dos nl níveis citados anteriormente, como expresso pela Equação 3.10.

$$\begin{aligned} t^{p_i,l_c}_{finalDC(r)} &= PegaItemPos(DC_{p_i,r}, 1) \quad onde, \\ DC_{p_i,r} &= \bigcup DC_{p_i,r,nl}, \forall nl, 1 \leq nl \leq N-1. \end{aligned} \tag{3.10}$$

O conjunto $DC_{p_i,r,nl}$ representa os valores possíveis para o tempo de ocorrência do evento de recebimento de uma difusão confiável, pela primeira vez, no nível nl, em p_i, o qual é denominado de $t^{p_i,l_c}_{finalDC(r,nl)}$. No primeiro nível ($nl = 1$), existe um único valor para $t^{p_i,l_c}_{finalDC(r,nl)}$ o qual refere-se ao tempo de receber a difusão diretamente do processo que a iniciou. Nos níveis subsequentes, o tempo de ocorrência do evento $finalDC(r,x)$ em p_i é dado em função do tempo de ocorrência do evento $finalDC(r,x-1)$ em algum processo no nível $x-1$ e, assim, para todos os níveis anteriores. É importante ressaltar que, se um processo p_i inicia a difusão, então p_i pertence ao primeiro nível ($p_i = pn_1$), caso contrário, p_i pode pertencer a qualquer nível da difusão ($p_i = pn_{nl}$, $1 \leq nl \leq N-1$)[5]. Seja Π_{nl} o conjunto dos processos que pertencem ao nível nl. A definição do conjunto $DC_{p_i,r,nl}$ para um processo p_i e considerando nl níveis de difusão confiável, $1 \leq nl \leq N-1$, é dada pela Equação 3.11.

$$\begin{aligned} nl = 1, \quad DC_{p_i,r,1} &= \{t^{p_i,l_c}_{finalDC(r,1)} \mid t^{p_i,l_c}_{finalDC(r,1)} \in \mathbb{R}_+, \\ t^{p_i,l_c}_{finalDC(r,1)} &= t^{p_c(r),l_c}_{valorC(r)} + Lat(p_c(r), p_i, t^{p_c(r),l_c}_{valorC})\} \\ nl > 1, \quad DC_{p_i,r,nl} &= \{t^{p_i,l_c}_{finalDC(r,nl)} \mid t^{p_i,l_c}_{finalDC(r,nl)} \in \mathbb{R}_+, \\ t^{p_i,l_c}_{finalDC(r,nl)} &= t^{pn_{nl-1},l_c}_{finalDC(r,nl-1)} + Lat(pn_{nl-1}, p_i, t^{pn_{nl-1},l_c}_{finalDC(r,nl-1)}), \\ \forall pn_z \in \Pi_z \wedge \forall t^{pn_z,l_c}_{finalDC(r,z)} &\in DC_{pn_z,r,z}, z = nl-1\}. \end{aligned} \tag{3.11}$$

As equações 3.10 e 3.11 representam o tempo de ocorrência do evento **e4** ($finalDC(r)$). Entretanto, o estudo analítico sobre o tempo de execução do *consenso-CT* requer, ainda, a definição dos tempos de ocorrência dos eventos **e1** ($propostaC(r)$) e **e2** ($valorC(r)$). Como foi apresentado anteriormente, tais eventos ocorrem no coordenador de uma rodada

[5]Note que a difusão iniciada por um processo p_i é disseminada para todos os participantes da difusão inclusive para o próprio p_i. Então, o processo que inicia a difusão recebe a mensagem difundida, pela primeira vez, dele mesmo.

r, $1 \leq r \leq \lfloor (N+1)/2 \rfloor$, após o recebimento de um número majoritário de mensagens enviadas pelos demais participantes do protocolo. Considere os conjuntos $PROPOSTAC_{p_c(r)}$ e $VALORC_{p_c(r)}$ para representar todos os tempos nos quais foram recebidas, respectivamente, mensagens do tipo estimativas de valor consensual (associadas a **e1**) e confirmações de valor consensual (associadas a **e2**). Note que o número de participantes em cada rodada diminui ao longo da execução do protocolo devido ao "travamento" dos processos nas rodadas coordenadas pelos mesmos. Consequentemente, o tamanho dos conjuntos $PROPOSTAC_{p_c(r)}$ e $VALORC_{p_c(r)}$ também diminui. Além disso, para $r > \lfloor (N+1)/2 \rfloor$, o tamanho do conjunto $VALORC_{p_c(r)}$ é vazio. O tempo de ocorrência dos eventos **e1** e **e2**, como também, a definição dos conjuntos $PROPOSTAC_{p_c(r)}$ e $VALORC_{p_c(r)}$ são dados, respectivamente, pelas equações 3.12 e 3.13. Em ambas as equações, *pid* representa o identificador do processo origem das mensagens enviadas ao coordenador $p_c(r)$, além disso, função $PegaItemPosId$ é definida pela Equação 3.6.

$$t^{p_c(r),l_c}_{propostaC(r)} = max(t^{p_c(r),l_c}_{inicialR(r)}, PegaItemPosId(PROPOSTAC_{p_c(r)}, \lceil (N+1)/2 \rceil)), \quad onde,$$
$$|PROPOSTAC_{p_c(r)}| \geq \lceil (N+1)/2 \rceil;$$

$$PROPOSTAC_{p_c(r)} = \{(t^{p_c(r),l_c}_{est(r)}, pid) \mid (t^{p_c(r),l_c}_{est(r)}, pid) \in \mathbb{R}_+ \times \mathbb{N},$$
$$t^{p_c(r),l_c}_{est(r)} = t^{p_i,l_c}_{inicialR(r)} + Lat(p_i, p_c(r), t^{p_i,l_c}_{inicialR(r)}),$$
$$\forall p_i \in \Pi\}. \tag{3.12}$$

$$t^{p_c(r),l_c}_{valorC(r)} = PegaItemPosId(VALORC_{p_c(r)}, \lceil (N+1)/2 \rceil), \quad onde,$$
$$|VALORC_{p_c(r)}| \geq \lceil (N+1)/2 \rceil;$$

$$VALORC_{p_c(r)} = \{(t^{p_c(r),l_c}_{confirma(r)}, pid) \mid (t^{p_c(r),l_c}_{confirma(r)}, pid) \in \mathbb{R}_+ \times \mathbb{N}, \tag{3.13}$$
$$t^{p_c(r),l_c}_{confirma(r)} = t^{p_c(r),l_c}_{propostaC(r)} + Lat(p_c(r), p_i, t^{p_c(r),l_c}_{propostaC(r)}) +$$
$$Lat(p_i, p_c(r), t^{p_c(r),l_c}_{propostaC(r)} + Lat(p_c(r), p_i, t^{p_c(r),l_c}_{propostaC(r)}))\}.$$

O evento de inicialização de uma rodada r, denominado de $inicialR(r)$, precede ou ocorre no mesmo instante de qualquer outro evento associado à execução do *consenso-CT*. Além disso, o evento $valorC(r)$ sempre ocorrerá após o evento $propostaC(r)$, então $t^{p_c(r),l_c}_{valorC(r)} > t^{p_c(r)}_{proposta} \geq t^{p_c(r)}_{inicialR(r)}$. Isto justifica a definição de $t^{p_c(r),l_c}_{propostaC(r)}$ na Equação 3.12.

A primeira rodada do consenso representa o início da execução do protocolo, nesse caso, $t^{p_i,l_c}_{inicialR(1)} = t^{p_i,l_c}_{inicialC(1)}$, $\forall p_i \in \Pi$. A inicialização das demais rodadas por cada p_i está condicionada ao recebimento de mensagens na rodada anterior. Mais especificamente, se r é a rodada da qual p_i participa, $1 \leq r \leq N-1$, $p_i \neq p_c(r)$, então p_i iniciará a rodada $r+1$ após o recebimento de uma proposta de valor consensual enviada pelo coordenador da rodada corrente $p_c(r)$. Dessa forma, $t^{p_i,l_c}_{inicialR(r)}$ para $r \neq 1$ é dado pela equação abaixo.

$$t^{p_i,l_c}_{inicialR(r)} = t^{p_c(r-1),l_c}_{proposalC(r-1)} + Lat(p_c(r-1), p_i, t^{p_c(r-1),l_c}_{proposalC(r-1)}). \tag{3.14}$$

O desempenho do *consenso-CT* adaptativo foi representado através de uma série de definições as quais dependem da definição de um oráculo de latência (ver Seção 3.2). A partir das definições apresentadas é possível representar uma estimativa do tempo de execução do protocolo para todas as configurações possíveis de lista ordenada de processos. Nesse caso, o mecanismo de adaptação baseado no oráculo de latência poderá escolher uma configuração eficiente para toda execução do protocolo. Consequentemente, haverá ganhos de desempenho para o *consenso-CT* adaptativo, demonstrando a eficiência do mecanismo de adaptação.

Em suma o estudo analítico sobre o desempenho do *consenso-CT* consiste das seguintes etapas:

1. Definir o conjunto de participantes do protocolo Π;
2. Identificar o parâmetro de comportamento cujos valores serão alterados pelo mecanismo de adaptação. No caso do *consenso-CT* este parâmetro é a lista ordenada de processos utilizada em cada execução do protocolo;
3. Definir o conjunto $PE(n)$ para Π cujos elementos representam todos os possíveis valores para a lista ordenada de processos;
4. Definir o tempo inicial de uma execução do consenso usando a lista $l_c \in PE(n)$;
5. Definir, para cada $p_i \in \Pi$, o conjunto dos tempos finais de uma execução do consenso usando a lista l_c — $FINALC_{p_i,l_c}$;
6. Definir, para cada p_i, o conjunto das durações de consenso em uma execução usando a lista l_c — $EXEC\_LOCAL_{p_i,l_c}$;
7. Definir o conjunto das menores durações de consenso de cada p_i em uma execução usando a lista l_c — $EXEC\_GLOBAL_{\Pi,l_c}$;

8. Definir a k-ésima melhor duração de consenso em uma execução usando a lista l_c — $T^{\Pi,l_c}_{inicialC(1),finalC}$;

9. Repetir os passos (4), (5), (6), (7) e (8) $\forall l_c \in PE(n)$ e definir o conjunto com as k-ésimas melhores durações de consenso para todas as listas ordenadas de processos — $EXEC\_GLOBAL_{\Pi,PE(n)}$.

Ao final das etapas enumeradas acima é possível estimar a configuração de lista ordenada de processos mais eficiente para uma execução do *consenso-CT*, que será aquela cuja duração de consenso é dada por $PegaItemPosLista(EXEC\_GLOBAL_{\Pi,PE(n)},1)$. Conhecer a priori uma estimativa de desempenho do *consenso-CT* para todas as possíveis configurações de lista ordenada de processos garante a escolha por configurações eficientes em toda execução do protocolo. Usando este procedimento para tornar o *consenso-CT* adaptativo comprova-se a eficiência do mecanismo de adaptação.

3.5.1 Instanciando o estudo analítico para 3 processos

Na seção anterior apresentou-se um estudo analítico sobre o desempenho do protocolo *consenso-CT* adaptativo, através do qual foi possível confirmar a eficiência do mecanismo de adaptação utilizado. O estudo consta de 9 passos e resulta em um modelo de desempenho genérico. A seguir, discute-se a aplicação do modelo para 3 processos.

Inicialmente (**PASSO 1**), deve-se definir o conjunto de processos participantes do consenso, $\Pi_{e3} = \{p_1, p_2, p_3\}, N = |\Pi_{e3}| = 3$. Note que, $Lat(p_i, p_i, t) = 0$, $\forall t \in \mathbb{R}_+$ e $\forall p_i \in \Pi_{e3}$.

O parâmetro de comportamento do *consenso-CT* corresponde à lista ordenada de processos (**PASSO 2**) cujos valores são representados pelo conjunto permutação $PE(3)$ descrito abaixo (**PASSO 3**):

$$PE(3) = \{[p_1, p_2, p_3], [p_1, p_3, p_2], [p_2, p_1, p_3], [p_2, p_3, p_1], [p_3, p_1, p_2], [p_3, p_2, p_1]\}$$

Um estudo completo sobre o desempenho do *consenso-CT* com 3 processos envolveria a repetição dos passos 4 a 8 para todos os elementos do conjunto $PE(3)$ e, então, a realização do passo 9. Note que, a aplicação do modelo pode ser demonstrada usando, apenas, uma configuração do protocolo, ou seja, um valor para a lista ordenada de processos. Considere a lista $l_c = [p_1, p_2, p_3]$, o tempo inicial de uma execução do consenso usando tal lista pode ser escolhido aleatoriamente e será o mesmo para todos os participantes do protocolo, então assume-se $t^{p_i,[p_1,p_2,p_3]}_{inicialC(1)} = 0$, $\forall p_i \in \Pi_{e3}$ (**PASSO 4**).

O próximo passo envolve a definição do conjunto $FINALC_{p_i,[p_1,p_2,p_3]}$ para todos os participantes do protocolo. Por questões de simplicidade, na definição do modelo utilizou-se uma abordagem "top-down" para realizar esse passo. Por outro lado, na aplicação do modelo é mais interessante usar uma abordagem "bottom-up". As ações associadas ao **PASSO 5** incluem: i) definir o tempo de ocorrência do evento $propostaC(r)$ e o conjunto $PROPOSTAC_{p_c(r)}$, ii) definir o tempo de ocorrência do evento $valorC(r)$ e o conjunto $VALORC_{p_c(r)}$ e, iii) definir o tempo inicial de cada rodada executada por $p_i \in \Pi_{e3}$. Os resultados destas ações estão organizados por rodada (r), onde $1 \leq r \leq 3$.

PASSO 5, $r = 1$**.** A rodada $r = 1$ é coordenada pelo processo p_1 e tem como participantes todos os elementos do conjunto Π_{e3}, ou seja, p_1, p_2 e p_3. O tempo inicial desta rodada para seus participantes é:

$$\begin{aligned} t^{p_1(1),[p_1,p_2,p_3]}_{inicialR(1)} = t^{p_1(1),[p_1,p_2,p_3]}_{inicialC(1)} = 0 \\ t^{p_2,[p_1,p_2,p_3]}_{inicialR(1)} = t^{p_2,[p_1,p_2,p_3]}_{inicialC(1)} = 0 \\ t^{p_3,[p_1,p_2,p_3]}_{inicialR(1)} = t^{p_3,[p_1,p_2,p_3]}_{inicialC(1)} = 0 \end{aligned}$$

O tempo de ocorrência do evento $propostaC(1)$ para o coordenador da rodada, p_1, e o conjunto $PROPOSTAC_{p_1(1)}$ são definidos a seguir:

$$\begin{aligned} PROPOSTAC_{p_1(1)} = \{&(t^{p_1(1),[p_1,p_2,p_3]}_{inicialR(1)} + Lat(p_1(1), p_1(1), t^{p_1(1),[p_1,p_2,p_3]}_{inicialR(1)}), p_1), \\ &(t^{p_2,[p_1,p_2,p_3]}_{inicialR(1)} + Lat(p_2, p_1(1), t^{p_2,[p_1,p_2,p_3]}_{inicialR(1)}), p_2), \\ &(t^{p_3,[p_1,p_2,p_3]}_{inicialR(1)} + Lat(p_3, p_1(1), t^{p_3,[p_1,p_2,p_3]}_{inicialR(1)}), p_3)\} \end{aligned}$$

$$t^{p_1(1),[p_1,p_2,p_3]}_{propostaC(1)} = max(t^{p_1(1),[p_1,p_2,p_3]}_{inicialR(1)}, PegaItemPosId(PROPOSTAC_{p_1(1)}, 2))$$

O tempo de ocorrência do evento $valorC$ para o coordenador da rodada, $p_1(1)$, e o conjunto $VALORC_{p_1(1)}$ são definidos a seguir:

$$\begin{aligned} VALORC_{p_1(1)} = \{&(t^{p_1(1),[p_1,p_2,p_3]}_{propostaC(r)} + Lat(p_1(1), p_1(1), t^{p_1(1),[p_1,p_2,p_3]}_{propostaC(r)}) + \\ &Lat(p_1(1), p_1(1), t^{p_1(1),[p_1,p_2,p_3]}_{propostaC(r)} + Lat(p_1(1), p_1(1), t^{p_1(1),[p_1,p_2,p_3]}_{propostaC(r)})), p_1), \\ &(t^{p_1(1),[p_1,p_2,p_3]}_{propostaC(r)} + Lat(p_1(1), p_2, t^{p_1(1),[p_1,p_2,p_3]}_{propostaC(r)}) + \\ &Lat(p_2, p_1(1), t^{p_1(1),[p_1,p_2,p_3]}_{propostaC(r)} + Lat(p_1(1), p_2, t^{p_1(1),[p_1,p_2,p_3]}_{propostaC(r)})), p_2), \\ &(t^{p_1(1),[p_1,p_2,p_3]}_{propostaC(r)} + Lat(p_1(1), p_3, t^{p_1(1),[p_1,p_2,p_3]}_{propostaC(r)}) + \\ &Lat(p_3, p_1(1), t^{p_1(1),[p_1,p_2,p_3]}_{propostaC(r)} + Lat(p_1(1), p_3, t^{p_1(1),[p_1,p_2,p_3]}_{propostaC(r)})), p_3)\} \end{aligned}$$

$$t_{valorC(1)}^{p_1(1),[p_1,p_2,p_3]} = PegaItemPosId(VALORC_{p_1(1)}, 2)$$

A partir das definições dos conjuntos $PROPOSALC_{p_1(1)}$ e $VALORC_{p_1(1)}$, como também, dos tempos de ocorrência dos eventos $propostaC(1)$ e $valorC(1)$ é possível definir o tempo final da difusão confiável para cada participante de $r = 1$. Inicialmente, calcula-se os tempos de ocorrência do evento $finalDC(1, nl)$, onde $1 \leq nl \leq 2$. Note que apenas o coordenador de cada rodada pode iniciar uma difusão confiável. Então, em $r = 1$ a difusão confiável será iniciada por p_1 e este recebe o valor difundido no nível $nl = 1$; os demais processos, p_2 e p_3 podem receber o valor difundido nos níveis $nl = 1$ ou $nl = 2$.

Montagem do conjunto $DC_{p_i,1,1}$ o qual contém os tempos de ocorrência do evento $finalDC(1, 1)$ para todo $p_i \in \Pi_{e3}$:

$$\begin{aligned}
DC_{p_1(1),1,1} &= \{t_{valorC(1)}^{p_1(1),[p_1,p_2,p_3]} + Lat(p_1(1), p_1(1), t_{valorC(1)}^{p_1(1),[p_1,p_2,p_3]})\} \\
DC_{p_2,1,1} &= \{t_{valorC(1)}^{p_1(1),[p_1,p_2,p_3]} + Lat(p_1(1), p_2, t_{valorC(1)}^{p_1(1),[p_1,p_2,p_3]})\} \\
DC_{p_3,1,1} &= \{t_{valorC(1)}^{p_1(1),[p_1,p_2,p_3]} + Lat(p_1(1), p_3, t_{valorC(1)}^{p_1(1),[p_1,p_2,p_3]})\}
\end{aligned}$$

Montagem do conjunto $DC_{p_i,1,2}$ o qual contém os tempos de ocorrência do evento $finalDC(1, 2)$ para os processos p_2 e p_3:

$$\begin{aligned}
DC_{p_2,1,2} &= \{t_{finalDC(1,1)}^{pn_1,[p_1,p_2,p_3]} + Lat(pn_1, p_2, t_{finalDC(1,1)}^{pn_1,[p_1,p_2,p_3]})\} \\
&= \{t_{finalDC(1,1)}^{p_3,[p_1,p_2,p_3]} + Lat(p_3, p_2, t_{finalDC(1,1)}^{p_3,[p_1,p_2,p_3]})\} \\
DC_{p_3,1,2} &= \{t_{finalDC(1,1)}^{pn_1,[p_1,p_2,p_3]} + Lat(pn_1, p_3, t_{finalDC(1,1)}^{pn_1,[p_1,p_2,p_3]})\} \\
&= \{t_{finalDC(1,1)}^{p_2,[p_1,p_2,p_3]} + Lat(p_2, p_3, t_{finalDC(1,1)}^{p_2,[p_1,p_2,p_3]})\}
\end{aligned}$$

O conjunto $DC_{p_i,1}$ agrupa os valores possíveis para o tempo de ocorrência do evento $finalDC(1, nl)$ considerando todos os níveis no intervalo $[1, 2]$, *i.e.*, $DC_{p_i,1} = \bigcup DC_{p_i,1,nl}$, $1 \leq nl \leq 2$. A definição de tais conjuntos para cada $p_i \in \Pi_{e3}$ é apresentada abaixo.

$$\begin{aligned}
DC_{p_1(1),1} &= DC_{p_1(1),1,1} \\
DC_{p_2,1} &= DC_{p_2,1,1} \cup DC_{p_1(1),1,2} \\
DC_{p_3,1} &= DC_{p_3,1,1} \cup DC_{p_3,1,2}
\end{aligned}$$

O tempo final do consenso para um processo p_i é o menor tempo no qual p_i recebe uma difusão confiável, sendo representado pela variável $t_{finalC}^{p_i,l_c}$. Considerando a difusão iniciada pelo coordenador da rodada $r = 1$ $(p_1(1))$, o valor de $t_{finalC}^{p_i,l_c}$ é o seguinte:

$$t_{finalC}^{p_1(1),[p_1,p_2,p_3]} = PegaItemPos(DC_{p_1(1),1}, 1)$$
$$t_{finalC}^{p_2,[p_1,p_2,p_3]} = PegaItemPos(DC_{p_2,1}, 1)$$
$$t_{finalC}^{p_3,[p_1,p_2,p_3]} = PegaItemPos(DC_{p_3,1}, 1)$$

PASSO 5, $r = 2$**.** A rodada $r = 2$ é coordenada pelo processo p_2 e tem como participantes, além de p_2, o processo p_3. Como os coordenadores de cada rodada ficam "travados" nas mesmas até alcançarem um valor consensual, o coordenador de $r = 1$, p_1, não participa da rodada $r = 2$. A seguir, o tempo inicial da rodada $r = 2$ para p_2 e p_3.

$$t_{inicialR(2)}^{p_2(2),[p_1,p_2,p_3]} = t_{propostaC(1)}^{p_1(1),[p_1,p_2,p_3]} + Lat(p_1(1), p_2(2), t_{propostaC(1)}^{p_1(1),[p_1,p_2,p_3]})$$
$$t_{inicialR(2)}^{p_3,[p_1,p_2,p_3]} = t_{propostaC(1)}^{p_1(1),[p_1,p_2,p_3]} + Lat(p_1(1), p_3, t_{propostaC(1)}^{p_1(1),[p_1,p_2,p_3]})$$

O tempo de ocorrência do evento *propostaC* para o coordenador da rodada, $p_2(2)$, e o conjunto $PROPOSTAC_{p_2(2)}$ são definidos a seguir:

$$PROPOSTAC_{p_2(2)} = \{(t_{inicialR(2)}^{p_2(2),[p_1,p_2,p_3]} + Lat(p_2(2), p_2(2), t_{inicialR(2)}^{p_2(2),[p_1,p_2,p_3]}), p_2),$$
$$(t_{inicialR(2)}^{p_3,[p_1,p_2,p_3]} + Lat(p_3, p_2(2), t_{inicialR(2)}^{p_3,[p_1,p_2,p_3]}), p_3)\}$$

$$t_{propostaC(2)}^{p_2(2),[p_1,p_2,p_3]} = max(t_{inicialR(2)}^{p_2(2),[p_1,p_2,p_3]}, PegaItemPosId(PROPOSTAC_{p_2(2)}, 2))$$

O tempo de ocorrência do evento *valorC* para o coordenador da rodada, $p_2(2)$, e o conjunto $VALORC_{p_2(2)}$ são definidos a seguir:

$$\begin{aligned} VALORC_{p_2(2)} = \{&(t_{propostaC(2)}^{p_2(2),[p_1,p_2,p_3]} + Lat(p_2(2), p_2(2), t_{propostaC(2)}^{p_2(2),[p_1,p_2,p_3]}) + \\ &Lat(p_2(2), p_2(2), t_{propostaC(2)}^{p_2(2),[p_1,p_2,p_3]} + Lat(p_2(2), p_2(2), t_{propostaC(2)}^{p_2(2),[p_1,p_2,p_3]})), p_2), \\ &(t_{propostaC(2)}^{p_2(2),[p_1,p_2,p_3]} + Lat(p_2(2), p_3, t_{propostaC(2)}^{p_2(2),[p_1,p_2,p_3]}) + \\ &Lat(p_3, p_2(2), t_{propostaC(2)}^{p_2(2),[p_1,p_2,p_3]} + Lat(p_2(2), p_3, t_{propostaC(2)}^{p_2(2),[p_1,p_2,p_3]})), p_3)\} \end{aligned}$$

$$t_{valorC(2)}^{p_2(2),[p_1,p_2,p_3]} = PegaItemPosId(VALORC_{p_2(2)}, 2)$$

Em $r = 2$ a difusão confiável será iniciada por $p_2(2)$ e este recebe o valor difundido no nível $nl = 1$; os demais processos, p_1 e p_3 podem receber o valor difundido nos níveis $nl = 1$

ou $nl = 2$. O conjunto $DC_{p_i,2,1}$ contendo os tempos de ocorrência do evento $finalDC(2,1)$ para todo $p_i \in \Pi_{e3}$ é definido a seguir:

$$
\begin{aligned}
DC_{p_1,2,1} &= \{t_{valorC(2)}^{p_2(2),[p_1,p_2,p_3]} + Lat(p_2(2), p_1, t_{valorC(2)}^{p_2(2),[p_1,p_2,p_3]})\} \\
DC_{p_2(2),2,1} &= \{t_{valorC(2)}^{p_2(2),[p_1,p_2,p_3]} + Lat(p_2(2), p_2(2), t_{valorC(2)}^{p_2(2),[p_1,p_2,p_3]})\} \\
DC_{p_3,2,1} &= \{t_{valorC(2)}^{p_2(2),[p_1,p_2,p_3]} + Lat(p_2(2), p_3, t_{valorC(2)}^{p_2(2),[p_1,p_2,p_3]})\}
\end{aligned}
$$

Montagem do conjunto $DC_{p_i,2,2}$ o qual contém os tempos de ocorrência do evento $finalDC(2,2)$ para os processos p_1 e p_3:

$$
\begin{aligned}
DC_{p_1,2,2} &= \{t_{finalDC(2,1)}^{pn_1,[p_1,p_2,p_3]} + Lat(pn_1, p_1, t_{finalDC(2,1)}^{pn_1,[p_1,p_2,p_3]})\} \\
&= \{t_{finalDC(2,1)}^{p_3,[p_1,p_2,p_3]} + Lat(p_3, p_1, t_{finalDC(2,1)}^{p_3,[p_1,p_2,p_3]})\} \\
DC_{p_3,2,2} &= \{t_{finalDC(2,1)}^{pn_1,[p_1,p_2,p_3]} + Lat(pn_1, p_3, t_{finalDC(2,1)}^{pn_1,[p_1,p_2,p_3]})\} \\
&= \{t_{finalDC(2,1)}^{p_1,[p_1,p_2,p_3]} + Lat(p_1, p_3, t_{finalDC(2,1)}^{p_1,[p_1,p_2,p_3]})\}
\end{aligned}
$$

A definição do conjunto $DC_{p_i,2}$ para cada $p_i \in \Pi_{e3}$ é apresentada abaixo.

$$
\begin{aligned}
DC_{p_1,2} &= DC_{p_1,2,1} \cup DC_{p_1,2,2} \\
DC_{p_2(2),2} &= DC_{p_2(2),2,1} \\
DC_{p_3,2} &= DC_{p_3,2,1} \cup DC_{p_3,2,2}
\end{aligned}
\tag{3.15}
$$

Dessa forma o tempo final do consenso para uma difusão confiável iniciada na rodada $r = 2$, por p_2, é dado por:

$$
\begin{aligned}
t_{finalC}^{p_1,[p_1,p_2,p_3]} &= PegaItemPos(DC_{p_1,2}, 1) \\
t_{finalC}^{p_2(2),[p_1,p_2,p_3]} &= PegaItemPos(DC_{p_3,2}, 1) \\
t_{finalC}^{p_3,[p_1,p_2,p_3]} &= PegaItemPos(DC_{p_3,2}, 1)
\end{aligned}
$$

PASSO 5, $r = 3$**.** A rodada $r = 3$ é coordenada pelo processo p_3 que constitui o único participante desta rodada. Nesse caso, os processos p_1 e p_2 ficaram "travados", respectivamente, nas rodadas $r = 1$ e $r = 2$ as quais foram coordenadas pelos mesmos. O tempo inicial da rodada $r = 3$ para o processo p_3 é dado abaixo.

$$
t_{inicialR(3)}^{p_3(3),[p_1,p_2,p_3]} = t_{propostaC(2)}^{p_2(2),[p_1,p_2,p_3]} + Lat(p_2(2), p_3(3), t_{propostaC(2)}^{p_2(2),[p_1,p_2,p_3]})
$$

Como a quantidade de participantes desta rodada é inferior a $\lceil N + 1/2 \rceil = 2$, então não é possível calcular o tempo de ocorrência dos eventos $propostaC(3)$ e $valorC(3)$. Isto significa que os respectivos conjuntos, $PROPOSTAC_{p_3(3)}$ e $VALORC_{p_3(3)}$, não possuem a quantidade mínima de mensagens requerida para o cálculo de $t^{p_3(3),[p_1,p_2,p_3]}_{propostaC(3)}$ e $t^{p_3(3),[p_1,p_2,p_3]}_{valorC(3)}$. Os conjuntos e os tempos de ocorrência dos eventos citados são definidos abaixo.

$$PROPOSTAC_{p_3(3)} = \{(t^{p_3(3),[p_1,p_2,p_3]}_{inicialR(3)} + Lat(p_3(3), p_3(3), t^{p_3(3),[p_1,p_2,p_3]}_{inicialR(3)}), p_3)\}$$

$$\nexists t^{p_3(3),[p_1,p_2,p_3]}_{propostaC(3)} \quad pois \quad |PROPOSTAC_{p_3(3)}| < 2$$

$$VALORC_{p_3(3)} = \emptyset$$

$$\nexists t^{p_3(3),[p_1,p_2,p_3]}_{valorC(3)} \quad pois \quad |VALORC_{p_3(3)}| < 2$$

Haja vista a quantidade de participantes de $r = 3$ ser inferior a 2, o coordenador desta rodada não pode iniciar uma difusão confiável. Sendo assim $\nexists t^{p_i,[p_1,p_2,p_3]}_{finalC}$, $\forall p_i \in \Pi_{e3}$.

PASSO 5, definição do conjunto $FINALC_{p_i,[p_1,p_2,p_3]}$**.** O conjunto $FINALC_{p_i,[p_1,p_2,p_3]}$ reúne as possibilidades para o tempo de ocorrência do evento $finalC$ que representa o término do consenso para um processo p_i após o recebimento de uma difusão confiável, a qual pode ser iniciada nas rodadas $r = 1$ ou $r = 2$ como foi visto acima. A definição dos conjuntos $FINALC_{p_i,[p_1,p_2,p_3]}$ para cada $p_i \in \Pi_{e3}$ é apresentada abaixo:

$$FINALC_{p_1,[p_1,p_2,p_3]} = \{(PegaItemPos(DC_{p_1(1),1}, 1), 1), (PegaItemPos(DC_{p_1,2}, 1), 2)\}$$

$$FINALC_{p_2,[p_1,p_2,p_3]} = \{(PegaItemPos(DC_{p_2,1}, 1), 1), (PegaItemPos(DC_{p_2(2),2}, 1), 2)\}$$

$$FINALC_{p_3,[p_1,p_2,p_3]} = \{(PegaItemPos(DC_{p_3,1}, 1), 1), (PegaItemPos(DC_{p_3,2}, 1), 2)\}$$

O **PASSO 6** envolve a definição do conjunto de durações do consenso com a lista ordenada $[p_1, p_2, p_3]$ para todo processo $p_i \in \Pi_{e3}$, $EXEC\_LOCAL_{p_i,[p_1,p_2,p_3]}$. A duração de um consenso é dada pelo tempo transcorrido no intervalo $[t^{p_i,[p_1,p_2,p_3]}_{inicialC(1)}, t^{p_i,[p_1,p_2,p_3]}_{finalC}]$, $\forall t^{p_i,[p_1,p_2,p_3]}_{finalC} \in FINALC_{p_i,[p_1,p_2,p_3]}$. O conjunto $EXEC\_LOCAL_{p_i,[p_1,p_2,p_3]}$ é definido abaixo.

$$EXEC\_LOCAL_{p_1,[p_1,p_2,p_3]} = \{PegaItemPos(DC_{p_1(1),1}, 1) - 0, PegaItemPos(DC_{p_1,2}, 1) - 0\}$$
$$= \{PegaItemPos(DC_{p_1(1),1}, 1), PegaItemPos(DC_{p_1,2}, 1)\}$$

$$EXEC\_LOCAL_{p_2,[p_1,p_2,p_3]} = \{PegaItemPos(DC_{p_2,1}, 1) - 0, PegaItemPos(DC_{p_2(2),2}, 1) - 0\}$$
$$= \{PegaItemPos(DC_{p_2,1}, 1), PegaItemPos(DC_{p_2(2),2}, 1)\}$$

$$EXEC\_LOCAL_{p_3,[p_1,p_2,p_3]} = \{PegaItemPos(DC_{p_3,1}, 1) - 0, PegaItemPos(DC_{p_3,2}, 1) - 0\}$$
$$= \{PegaItemPos(DC_{p_3,1}, 1), PegaItemPos(DC_{p_3,2}, 1)\}$$

No **(PASSO 7)** define-se o conjunto $EXEC\_GLOBAL_{\{p_1,p_2,p_3\},[p_1,p_2,p_3]}$ que representa a menor duração de consenso para cada processo p_i, como apresentado a seguir. Tal informação é obtida a partir dos valores no conjunto $EXEC\_LOCAL_{p_i,[p_1,p_2,p_3]}$.

$$EXEC\_GLOBAL_{\{p_1,p_2,p_3\},[p_1,p_2,p_3]} = \{(PegaItemPos(EXEC\_LOCAL_{p_1,[p_1,p_2,p_3]}, 1), p_1),$$
$$(PegaItemPos(EXEC\_LOCAL_{p_2,[p_1,p_2,p_3]}, 1), p_2),$$
$$(PegaItemPos(EXEC\_LOCAL_{p_3,[p_1,p_2,p_3]}, 1), p_3)\}$$

Conhecendo o conjunto $EXEC\_GLOBAL_{\Pi_{e3},[p_1,p_2,p_3]}$ é possível obter a k-ésima duração do consenso, $1 \leq k \leq 3$ para uma execução usando a lista $[p_1, p_2, p_3]$ e, assim, representar o desempenho do consenso em termos do tempo de execução **(PASSO 8)**.

$$k = 1 \quad T^{\{p_1,p_2,p_3\},[p_1,p_2,p_3]}_{inicialC(1),finalC} = PegaItemPosId(EXEC\_GLOBAL_{\{p_1,p_2,p_3\},[p_1,p_2,p_3]}, 1)$$

$$k = 2 \quad T^{\{p_1,p_2,p_3\},[p_1,p_2,p_3]}_{inicialC(1),finalC} = PegaItemPosId(EXEC\_GLOBAL_{\{p_1,p_2,p_3\},[p_1,p_2,p_3]}, 2)$$

$$k = 3 \quad T^{\{p_1,p_2,p_3\},[p_1,p_2,p_3]}_{inicialC(1),finalC} = PegaItemPosId(EXEC\_GLOBAL_{\{p_1,p_2,p_3\},[p_1,p_2,p_3]}, 3)$$

O estudo descrito acima serviu para demonstrar a aplicação do modelo de desempenho do *consenso-CT* adaptativo, definido na Seção 3.5, para uma configuração de lista ordenada de processos, $l_c = [p_1, p_2, p_3]$. O procedimento para aplicar o modelo às demais configurações de lista ordenada de processos do conjunto $PE(3)$ é o mesmo e, por essa razão, não será apresentado nesta seção.

3.6 Conclusões parciais

Neste capítulo foi dado ênfase aos aspectos teóricos associados a oráculos de latência. As duas principais contribuições são a proposição de uma definição formal para oráculos de latência e sua aplicação no sentido de estudar o desempenho de protocolos adaptativos.

De acordo com a definição proposta, um oráculo de latência é uma função que informa uma estimativa de latência entre quaisquer dois processos do sistema (monitorados pelo oráculo) em qualquer instante de tempo. Isto implica que tal oráculo sabe informar sobre a situação de carga do ambiente todo o tempo, mesmo que tal informação tenha um erro associado (para mais ou menos). Além disso, o oráculo provê informações de latência globais, ou seja, diferentes consultas sobre a latência entre os mesmos dois processos em um determinado tempo resultará na mesma resposta.

É ainda importante ressaltar que o oráculo definido provê estimativas de latência no presente, passado ou futuro. Tais estimativas possuem um erro associado, nesse caso, quanto menor o erro maior a precisão das mesmas e, por conseguinte, maior a eficiência do mecanismo de adaptação baseado no oráculo de latência. Portanto, a garantia de erros pequenos deve ser uma restrição quando do projeto e implementação de oráculos de latência. De fato, é possível garantir a inexistência de erros, resultando em um oráculo de latência perfeito (*e.g.*, implementações para ambientes síncronos). Entretanto, para a maioria dos sistemas distribuídos reais, os quais são assíncronos, não é possível construir oráculos de latência perfeitos, mas aproximações com erros muito pequenos.

Usando a definição formal do oráculo é possível avaliar, analiticamente, a eficiência de uma solução adaptativa. Para demonstrar este fato, realizou-se um estudo analítico sobre o desempenho de um protocolo de consenso simétrico pelo texto (protocolo *consenso-CT* (CHANDRA; TOUEG, 1996)) equipado com uma solução adaptativa baseada em oráculos de latência. Protocolos simétricos pelo texto podem ser configurados quanto aos processos que assumem determinados papéis ao longo da execução dos mesmos, nesse caso, adaptação pode ser usada para definir tais configurações de acordo com a situação de carga do ambiente de execução. A solução adaptativa proposta para o *consenso-CT* usa as informações providas por um oráculo de latência para construir uma lista ordenada de processos a partir da qual escolhem-se os processos que desempenham o papel de coordenador de cada rodada do protocolo. O estudo analítico sobre o *consenso-CT* adaptativo consistiu da análise de um modelo de desempenho para o protocolo, o qual representa o fluxo de mensagens entre os seus participantes (padrão de comunicação) ao longo de uma execução sem falhas ou falsas suspeições; os atrasos na transmissão das mensagens são descritos em função da

definição do oráculo de latência. Através da análise do modelo demonstrou-se que, dado um oráculo de latência, é possível estimar o desempenho do protocolo para todas as configurações de lista ordenada de processos e, assim, garantir que a solução adaptativa sempre escolherá uma configuração eficiente. Em outras palavras, foi demonstrada a eficiência de uma solução adaptativa baseada em oráculos de latência.

O modelo de desempenho especificado para o *consenso-CT* foi instanciado para uma execução do protocolo com 3 processos participantes. Uma desvantagem de usar tal modelo é que a escolha da lista ordenada de processos mais eficiente depende da análise de desempenho do protocolo para cada configuração possível da lista. Como a quantidade de configurações da lista é dada pela permutação entre os participantes do protocolo, então a análise de desempenho do protocolo torna-se mais complexa à medida que cresce o número de participantes do mesmo.

Apesar do estudo analítico ter sido realizado para validar um mecanismo de adaptação para um domínio de problema específico (ordenação de processos), é possível generalizar as conclusões alcançadas até certo ponto. Note que a estratégia do modelo de desempenho, em função da definição do oráculo de latência, para avaliar a eficiência do uso de adaptação é aplicável a diferentes soluções adaptativas baseadas em oráculos de latência e, por conseguinte, pode contemplar outros protocolos simétricos pelo texto além daqueles que seguem o paradigma do coordenador rotativo, como estudado. Em princípio, a única restrição está no fato do protocolo simétrico pelo texto ter seu desempenho dependente do padrão de comunicação seguido pelo mesmo.

Capítulo 4

Aplicando oráculos de latência a um sistema baseado em consenso para tolerância a intrusões na Internet

O consenso é considerado um bloco básico para a implementação de soluções para problemas fundamentais de sistemas distribuídos tolerantes a faltas, tais como confirmação atômica, difusão atômica, filiação a grupo e replicação (CHANDRA; TOUEG, 1996; TUREK; SHASHA, 1992; GALLENI; POWELL, 1996; GREVE; NARZUL, 2005; DESWARTE; POWELL, 2006). Estes problemas são classificados como problemas de acordo.

Normalmente, o consenso é implementado como um módulo independente, com interface bem definida, o qual é usado pelos serviços distribuídos de acordo. Modularização é um dos atrativos das soluções baseadas em consenso (GREVE; NARZUL, 2004). Outro ponto positivo refere-se à facilidade de validar (provar correção de) serviços baseados em consenso quando comparados àqueles não baseados em consenso. Note que, o consenso é uma abstração simples e bem definida, a qual encapsula o cerne de um problema de acordo. Além disso, problemas de acordo (por exemplo, confirmação atômica) são mais complexos do que o consenso. Sendo assim, uma forma fácil de validar soluções de acordo é começar pela validação do módulo de consenso (TUREK; SHASHA, 1992). Nesse caso, se o módulo de consenso estiver incorreto, este resultado também se estende à solução de acordo. Em outras palavras, os resultados obtidos para o consenso também serão aplicados aos serviços de acordo baseados no mesmo.

Muitos protocolos de consenso disponíveis na literatura são simétricos pelo texto, mas não usam adaptação para configurar automaticamente os papéis assumidos por seus participantes. Ao contrário, tal parâmetro é configurado a priori podendo causar degradação

no desempenho do protocolo em ambientes sujeitos à carga heterogênea e dinâmica. Questões de desempenho do consenso têm sido objeto de estudo de vários pesquisadores. Os principais resultados já publicados avaliam os protocolos seguindo uma abordagem mais teórica, através de métricas e técnicas pouco realistas (ou limitadas) (HURFIN et al., 2001; GUERRAOUI; RAYNAL, 2004; DELPORTE-GALLET; FAUCONNIER, 2002); os trabalhos que usam abordagens mais práticas propõem-se a investigar o impacto de elementos externos ao protocolo no seu desempenho e não das características do protocolo em si (SERGENT; DéFAGO; SCHIPER, 2001; COCCOLI et al., 2002; URBáN et al., 2004). Além disso, não se considera o desempenho de tais protocolos em ambientes sujeitos à carga heterogênea e dinâmica que constitui um cenário interessante, dado a existência de aplicações para o consenso (com requisitos de tolerância a faltas). Nesse contexto, por exemplo, aplicações que envolvem replicação de dados (DESWARTE; POWELL, 2006; CACHIN; PORITZ, 2002; CACHIN; SAMAR, 2004; RODRIGUES et al., 2002; VERíSSIMO et al., 2006; AMIR et al., 2006).

Como foi discutido no Capítulo 3, protocolos de consenso simétricos pelo texto podem tornar-se adaptativos pela introdução de uma solução para ordenação de processos adaptativa baseada em oráculos de latência. Nas próximas seções serão descritos o projeto e a implementação de um sistema adaptativo baseado em consenso para tolerância a intrusões na Internet (DESWARTE; POWELL, 2004, 2006). De fato, foi dado ênfase ao subsistema de consenso adaptativo. O mecanismo de adaptação utilizado usa um oráculo de latência que informa sobre a latência entre os processos participantes do subsistema. É bom lembrar que o modelo de sistema adotado foi descrito no Capítulo 3.

O restante deste capítulo está organizado da seguinte forma. A Seção 4.1 descreve uma aplicação baseada em consenso para tolerância a intrusões na Internet; a aplicação representa um sistema de arquivos distribuído que usa fragmentação, redundância e dissemінação para garantir requisitos de segurança. O subsistema de consenso, objeto de estudo deste capítulo, é também descrito nesta seção. Na Seção 4.2 apresenta-se uma versão adaptativa para o subsistema de consenso em estudo, cujo mecanismo de adaptação é baseado na solução genérica para ordenação de processos proposta no Capítulo 3. O subsistema de consenso é composto de vários componentes cujos projetos são discutidos na Seção 4.3. Enquanto a implementação destes componentes é descrita na Seção 4.4. Finalmente, os principais resultados deste capítulo são resumidos e comentados na Seção 4.5.

4.1 Um sistema baseado em consenso para tolerância a intrusões na Internet através de fragmentação e disseminação de dados

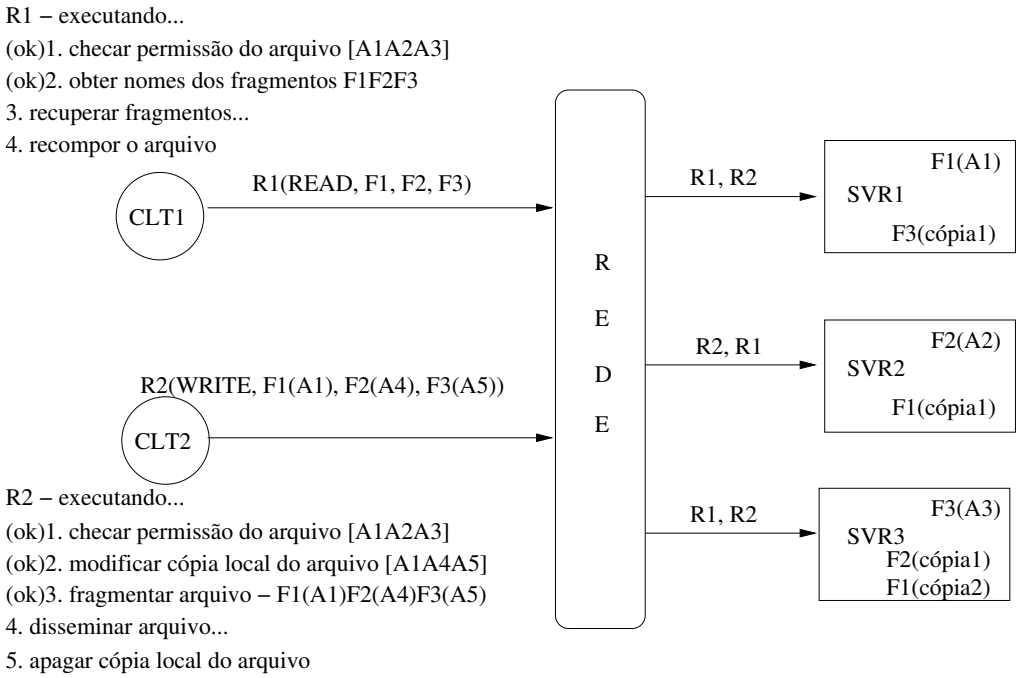
A Internet é um sistema aberto que reúne inúmeros usuários e diversas aplicações. Consequentemente, existem diferentes razões para usar a rede (tais como comercial, cultural, política, social e recreativa) e, assim, diferentes requisitos e políticas de segurança. O nível de segurança implantado em um servidor para aplicação bancária não é o mesmo de um servidor que hospeda páginas pessoais. Isto torna a Internet cada vez mais vulnerável a ataques e sujeita a intrusões.

Um sistema seguro deve garantir 3 propriedades: **confidencialidade**, proteção a informações pessoais ou jurídicas cuja descoberta causam perda de privacidade; **disponibilidade**, proteção contra interrupção do serviço oferecido; **integridade**, proteção contra ações que causem danos a informações pessoais ou jurídicas divulgadas na rede (destruição, modificação ou falsificação). Mecanismos de segurança tradicionais, baseadas em controle de acesso (*e.g.*, autenticação e autorização), são insuficientes para o contexto da Internet. Uma alternativa é usar conceitos e técnicas da área de confiança no funcionamento, o que originou a abordagem de tolerância a intrusões (DESWARTE; POWELL, 2004, 2006). Esta abordagem considera que o sistema apresenta vulnerabilidades as quais podem ser exploradas por ataques e gerar intrusões, mas tais intrusões não devem impedir o sistema de prover o serviço esperado. Em outras palavras, um sistema tolerante a intrusões garante as propriedades de segurança mesmo na presença de ataques.

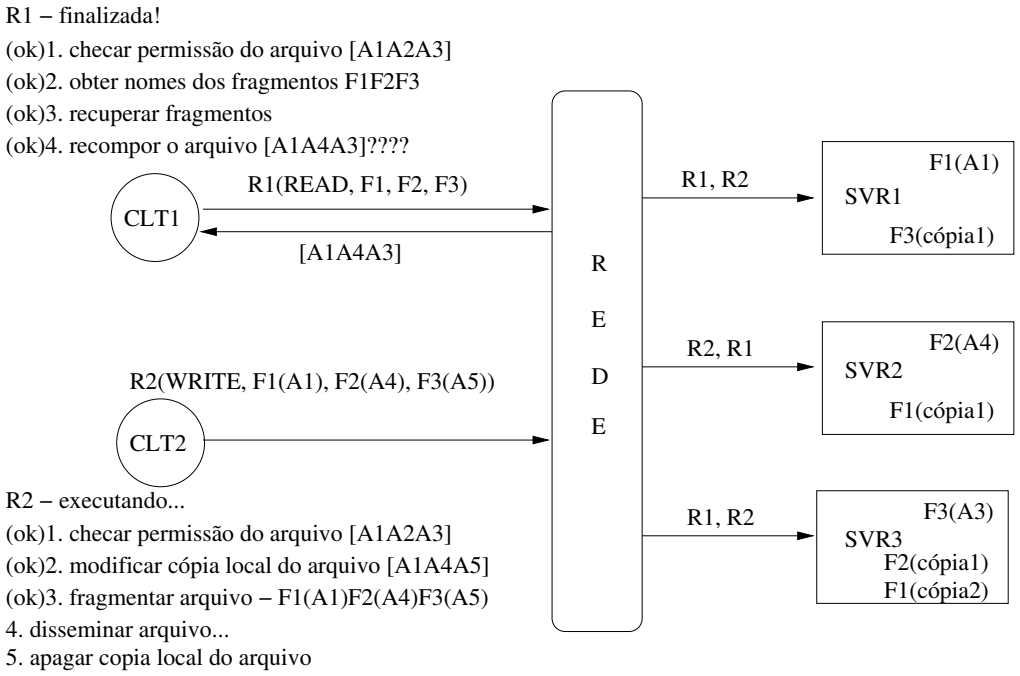
Redundância é a chave para tolerar faltas no contexto de confiança no funcionamento. Ao aplicar esta técnica para tolerar intrusões na Internet é preciso considerar que redundância implica em aumentar a quantidade de "alvos" no sistema susceptíveis a ataques, além disso, o mesmo ataque pode comprometer vários alvos. Por outro lado, ao combinar redundância e distribuição é possível dividir e espalhar a informação (ou processamento) do sistema de modo que, intrusões em uma parte do sistema não comprometem a confidencialidade, integridade e disponibilidade do sistema como um todo. Nesse sentido, foi proposto o mecanismo de fragmentação-redundância-disseminação (DESWARTE et al., 1988; FRAGA; POWELL, 1985). A fragmentação consiste em dividir a informação em pequenos fragmentos, os quais, isoladamente, não permitem recompor a informação. Ou seja, um fragmento isolado não representa informação útil; isto garante confidencialidade. Os fragmentos são replicados, assim, a modificação ou destruição de uma cópia (ou réplica) não impede a

reconstituição do dado por usuários legítimos. Nesse caso, redundância garante disponibilidade e integridade. A disseminação está relacionada com a forma pela qual os fragmentos são isolados uns dos outros e tem o objetivo de dificultar o acesso a toda a informação por parte dos intrusos. Isto é alcançado através da disseminação em diferentes lugares (repositórios de dados distribuídos geograficamente) e por diferentes canais de comunicação, ou ainda, transmitindo os fragmentos em ordem aleatória e com várias frequências (para aplicações com comunicação via rádio). Dessa forma, para obter toda a informação, o intruso precisa atacar o sistema várias vezes, combinando lugares, frequências e momentos de forma consistente; disseminação garante integridade e confidencialidade.

Alguns serviços baseados no mecanismo de fragmentação-redundância-disseminação foram desenvolvidos, a saber: armazenamento de arquivos, gerenciamento de segurança e processamento de dados (DESWARTE; POWELL, 2006; DESWARTE et al., 1988; FABRE; DESWARTE; RANDELL, 1994; FRAGA; POWELL, 1985). Em particular, o serviço de armazenamento de arquivos funciona da seguinte forma (DESWARTE; POWELL, 2006; DESWARTE et al., 1988). Cada arquivo é particionado em páginas de tamanhos iguais, estas são criptografadas e, então, fragmentadas. Todo fragmento possui um identificador único, o qual é obtido através de funções determinísticas de criptografia. O uso de criptografia tem o objetivo de aumentar o nível de segurança. A disseminação dos fragmentos de uma página pela rede segue uma ordem aleatória e, cada fragmento é enviado para um conjunto de servidores de armazenamento que decidem a localização das réplicas associadas ao mesmo. A fragmentação e disseminação ocorre no lado cliente, enquanto a redundância e armazenamento dos fragmentos é de responsabilidade do lado servidor. Operações comuns sobre arquivos são permitidas (tais como, criação, remoção, leitura e escrita). A operação de leitura engloba as seguintes atividades: 1) solicitação de abertura de arquivo para leitura (cliente), 2) verificação de permissão de leitura (cliente), 3) obtenção dos identificadores dos fragmentos do arquivo (cliente), 4) solicitação dos fragmentos do arquivo (cliente), 5) envio dos fragmentos do arquivo ao cliente (servidor) e 6) composição do arquivo a partir dos fragmentos recebidos (cliente). Já a operação de escrita é realizada através dos seguintes passos: 1) solicitação de abertura de arquivo para escrita (cliente), 2) verificação de permissão de escrita (cliente), 3) modificação da cópia local do arquivo, 4) fragmentação do arquivo modificado (cliente), 5) disseminação e armazenamento do arquivo modificado (servidor), 6) remoção da cópia local do arquivo (cliente). Note que podem existir vários clientes acessando o mesmo arquivo, concorrentemente, através de operações de leitura e escrita. Faz-se necessário sincronizar tais operações para evitar inconsistências (DESWARTE et al., 1988).



(a) Operação de leitura e escrita concorrentes



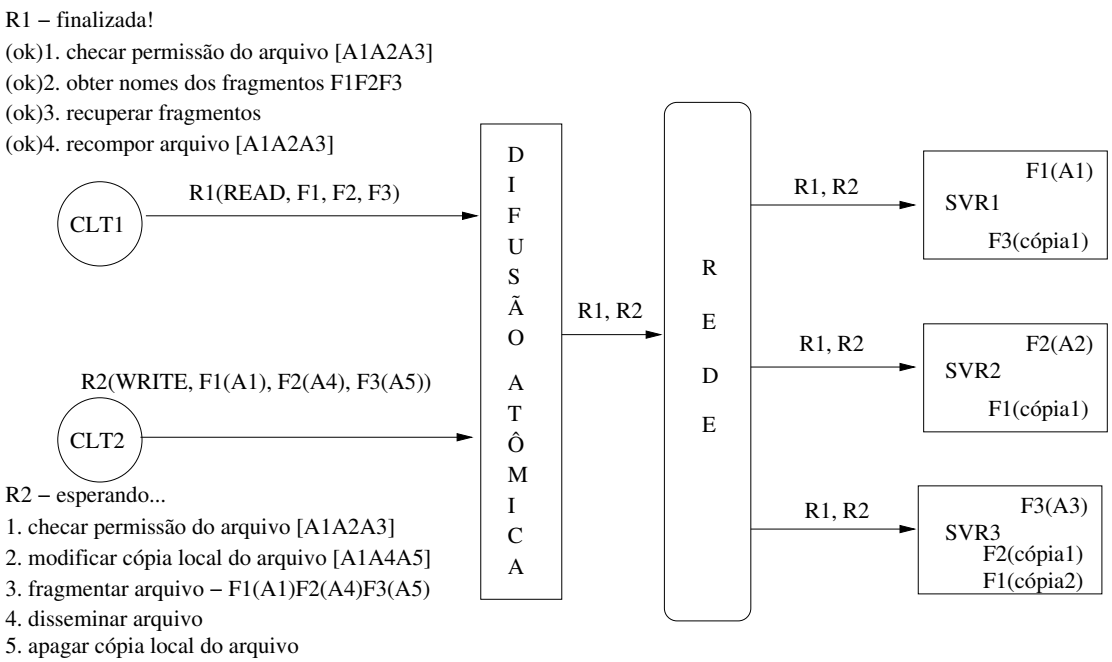
(b) Operação de leitura finalizada com valor inconsistente

Figura 4.1: Inconsistência na operação de leitura devido à falta de sincronização com outras operações concorrentes

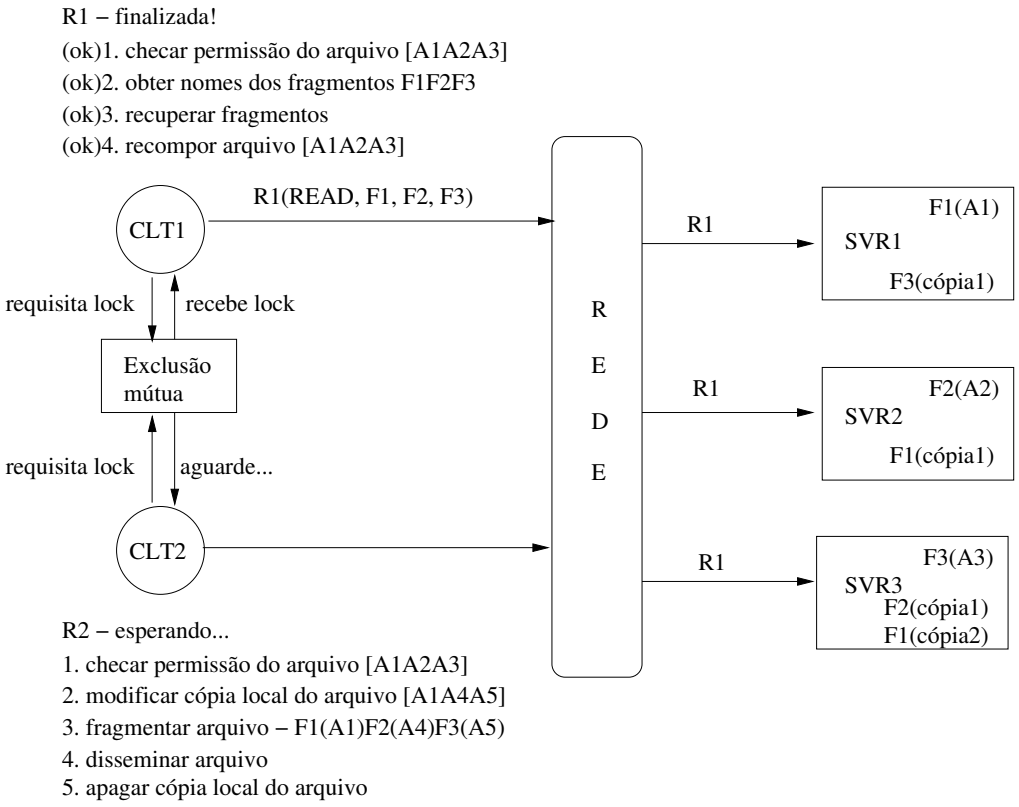
A Figura 4.1 ilustra uma situação de inconsistência gerada pela falta de sincronismo nas operações de leitura e escrita sobre arquivos em um sistema baseado em fragmentação-redundância-disseminação. Seja o arquivo (ou página) representado através dos fragmentos $[F_1(A_1), F_2(A_2), F_3(A_3)]$, onde F_i é o identificador do fragmento e A_i o seu conteúdo. Os fragmentos estão armazenados nos servidores $SVR1$, $SVR2$, $SVR3$. O cliente $CLT1$ realiza uma operação de leitura sobre este arquivo, ao mesmo tempo em que o cliente $CLT2$ realiza uma operação de escrita. Durante a operação de escrita, os fragmentos $F2$ e $F3$ serão modificados resultando na seguinte representação para o arquivo: $[F_1(A_1), F_2(A_4), F_3(A_5)]$. O cliente $CLT1$ pode ler o arquivo antes do término da operação de escrita, recuperando informação sem sentido, tal como $[F_1(A_1), F_2(A_4), F_3(A_3)]$ ilustrado na Figura 4.1(b). A sincronização das operações sobre o arquivo pode ser obtida através de mecanismos no lado cliente ou servidor. No primeiro caso, utiliza-se a idéia de exclusão mútua para acessar o arquivo (região crítica); os clientes decidem entre si quem ganhará acesso exclusivo ao arquivo a cada momento. Note que essa solução pode gerar uma sobrecarga significativa para uma grande quantidade de clientes, sendo mais adequado a redes locais. A segunda opção é usar difusão atômica para garantir ordenação total no processamento das requisições de operações sobre um arquivo no lado servidor. Nesse caso, as requisições dos clientes são repassadas aos servidores através de difusão atômica. Ao contrário da sincronização no lado cliente, a solução baseada em difusão atômica gera menor sobrecarga quando se tem muitos clientes, como na Internet. A Figura 4.2 ilustra ambas as estratégias.

Difusão atômica é um problema de acordo equivalente ao consenso, ou seja, é possível construir soluções para difusão atômica em função de uma solução para o consenso (CHANDRA; TOUEG, 1996). Da mesma forma, resultados de impossibilidade comprovados para o problema do consenso também são aplicados à difusão atômica em sistemas assíncronos.

Nesta tese foi proposto o uso de adaptação em protocolos distribuídos simétricos pelo texto para fins de desempenho. Como estudo de caso, optou-se por um subsistema de consenso simétrico pelo texto inserido em um sistema de arquivos distribuído baseado em fragmentação-redundância-disseminação, onde a consistência das operações sobre arquivos é garantida por meio de ordenação total (difusão atômica) no lado servidor. Por questões de simplicidade, a aplicação (sistema de arquivos distribuído) será abstraída e funcionará como um gerador de carga para o subsistema de consenso. Dessa forma, o foco do estudo de caso é o subsistema de consenso adaptativo, incluindo: descrição, projeto e implementação. A seção seguinte descreve o subsistema de consenso a ser estudado neste capítulo, porém sem considerar ainda o mecanismo de adaptação.



(a) Sincronização no lado servidor



(b) Sincronização no lado cliente

Figura 4.2: Garantindo consistência sobre operações de leitura e escrita

4.1.1 O subsistema de consenso: um protocolo simétrico pelo texto para sistemas assíncronos - *consenso-GR*

O protocolo *consenso-GR*

Existem diversas soluções para o problema do consenso em sistemas assíncronos com detectores de falhas não confiáveis disponíveis na literatura (HURFIN et al., 2001; CHANDRA; TOUEG, 1996; GUERRAOUI; RAYNAL, 2004; SCHIPER, 1997; BRASILEIRO et al., 2001; MOSTEFAOUI; RAYNAL, 1999). A solução estudada neste capítulo foi proposta por Guerraoui e Raynal e corresponde ao arcabouço unificado para o projeto de protocolos de consenso indulgentes[1] (GUERRAOUI; RAYNAL, 2004). O arcabouço descreve um protocolo de consenso genérico para sistemas assíncronos equipados com detectores de falhas. No sentido de garantir a generalidade da solução introduziu-se uma nova abstração denominada de LAMBDA. Esta abstração encapsula o uso de qualquer oráculo (por exemplo, detectores de falhas não confiáveis) durante cada rodada de execução do consenso. Nesse caso, é possível implementar diferentes protocolos de consenso, a partir do mesmo algoritmo, usando implementações diferentes de LAMBDA.

O consenso genérico proposto por Guerraoui e Raynal (*consenso-GR* - versão não-adaptativa) procede em rodadas assíncronas, onde cada rodada é composta de duas fases. Na primeira fase, denominada de fase de *seleção*, os processos participantes do protocolo irão escolher qual valor será proposto em consenso. Enquanto na segunda fase do protocolo, denominada de fase de *confirmação*, cada processo difunde a sua proposição para os demais a fim de decidir sobre um valor consensual. A decisão do consenso será alcançada quando todos os processos difundirem o mesmo valor v, $v \neq \perp$, na fase de *confirmação*[2]. O Algoritmo 2 descreve o pseudo-código do *consenso-GR* para um processo p_i (GUERRAOUI; RAYNAL, 2004).

O Algoritmo 2 é divido em duas tarefas: uma delas é responsável por obter a decisão do consenso (Tarefa T1) e a outra é responsável por difundir, de forma confiável, o valor consensual decidido (Tarefa T2). A fase de seleção é implementada pela função LAMBDA(r,est) (interface para a abstração LAMBDA) que recebe como parâmetros o identificador da rodada na qual a função está sendo invocada (r) e uma estimativa de consenso (est), fornecendo como saída um valor para ser proposto em consenso, naquela rodada específica. A função

[1] Nesse caso, os protocolos são indulgentes em relação ao oráculo utilizado. Portanto, mesmo se o oráculo apresentar um mau funcionamento, as propriedades de segurança do consenso nunca será violada, *i.e.*, ao final de um consenso todos os processos envolvidos no protocolo decidem o mesmo valor.

[2] O valor $\perp$ é definido de modo que nenhum processo participante do protocolo de consenso pode propor tal valor.

Algoritmo 2 Um protocolo de consenso genérico

when propose(v_i)
  Tarefa T1:
  $est1_i = v_i; r_i = 0$
  while true **do**
    $r_i = r_i + 1$
    % Fase de seleção
    $est2_i$ = LAMBDA($r_i, est1_i$)
    % Fase de confirmação
    broadcast PHASE2($r_i, est2_i$) to all
    waituntil delivered $n - f$ PHASE2(r_i,-) messages
    case (all messages delivered carry the same value v) : R_broadcast DECIDE(v)
    case (all messages delivered carry $\perp$) : $est1_i = \perp$
    case (at least one message carries v and other messages carry $\perp$) : $est1_i = v$
  end while
  ||
  Tarefa T2:
  when R_delivered DECIDE(v)
    return(v)
  end when
end when

LAMBDA(-, -) segue a especificação abaixo (GUERRAOUI; RAYNAL, 2004):

- **Validade:** nenhum processo invoca LAMBDA(1, $\perp$); além disso, se p_i receber o valor $v \neq \perp$ como resposta de uma chamada LAMBDA(r, -), então algum processo p_j (incluindo p_i) invocou LAMBDA(r', v), com $r' \leq r$.

- **Quase-acordo:** sejam p_i e p_j dois processos quaisquer que invocam LAMBDA(r, -) e recebem como resposta, respectivamente, os valores v e v'; se ($v \neq \perp$) e ($v' \neq \perp$), então $v = v'$.

- **Ponto fixo:** para qualquer rodada r, se todos os processos invocam LAMBDA(r, $est1$) com o mesmo valor $est1 = v$, então v e $\perp$ são os únicos valores possíveis de retorno para toda chamada LAMBDA(r', -), com $r' \geq r$.

- **Terminação:** para qualquer rodada r, todos os processos corretos que invocarem LAMBDA(r, -) receberão respostas para tal invocação.

- **Convergência após um tempo:** se todos os processos corretos invocarem, repetidas vezes, LAMBDA(-, -) para valores crescentes de rodada, então existirá uma rodada r tal que LAMBDA(r, -) retorna o mesmo valor v, $v \neq \perp$, para todos os processos corretos.

A propriedade de **Convergência após algum tempo** garante que a primeira opção do comando 'case' da fase de confirmação do Algoritmo 2, será executada, em algum momento, permitindo a obtenção de uma decisão para o consenso. Por outro lado, a propriedade de **Quase-acordo** irá garantir a consistência do conjunto de mensagens na fase de confirmação, nesse caso, estas mensagens devem satisfazer uma das opções do comando 'case' do Algoritmo 2. Dessa forma, se um processo decide o consenso, na rodada r, com o valor v, então qualquer processo p_j que finalizar a rodada r sem alcançar um valor consensual irá atualizar sua estimativa com o valor v ($est1_j = v$). Consequentemente, o valor consensual v é "travado" e qualquer decisão ocorrida em uma rodada futura l, $l > r$, remeterá para o mesmo valor v. Uma validação detalhada da correção do protocolo de consenso descrito é apresentada em (GUERRAOUI; RAYNAL, 2004).

Para obter diferentes implementações de protocolos de consenso, a partir do Algoritmo 2, basta utilizar uma implementação específica de LAMBDA(-, -). Em (GUERRAOUI; RAYNAL, 2004), são propostas algumas implementações de LAMBDA(-, -) usando diferentes oráculos. Os protocolos de consenso resultantes são tão eficientes quanto qualquer outro protocolo projetado para os respectivos oráculos (GUERRAOUI; RAYNAL, 2004). Nesse caso, eficiência é medida em termos de métricas de complexidade, especificamente, quantidade de passos de comunicação. O Algoritmo 3 descreve o pseudo-código da implementação baseada em $\Diamond S$ (GUERRAOUI; RAYNAL, 2004).

O Algoritmo 3 satisfaz as propriedades de **Validade**, **Quase-acordo**, **Ponto fixo**, **Terminação** e **Convergência após um tempo**, especificadas para a abstração LAMBDA. **Validade** é garantida pelas próprias características do algoritmo, onde, o retorno para qualquer chamada da função LAMBDA(r,-) será sempre o valor passado como entrada para a mesma por algum dos invocadores (nesse caso, o coordenador da rodada r) ou $\perp$. Além disso, pelo Algoritmo 2, nunca haverá uma invocação do tipo LAMBDA(1,$\perp$). A propriedade de **Quase-acordo** é assegurada pela fato da variável de retorno assumir, apenas, dois valores: $\perp$ ou v (valor proposto pelo coordenador da rodada corrente; note que, existe um único coordenador para cada rodada). Se todos os processos invocarem lambda com o mesmo valor, LAMBDA(r,v), o retorno da função será v (valor proposto pelo coordenador da rodada r, o qual também foi proposto pelos demais participantes) ou $\perp$, para qualquer processo em qualquer rodada r. Isto garante a propriedade de **Ponto fixo**. A validação

Algoritmo 3 Uma implementação de LAMBDA(r_i, $est1_i$) baseada no detector de falhas $\Diamond S$

```
if est1_i = ⊥ then
    est1_i = prev_est1_i
else
    prev_est1_i = est1_i
end if
c = (r_i mod n) + 1
if (i = c) then
    broadcast PHASE1(r_i, est1_i) to all
end if
waituntil delivered PHASE1(r_i, v) from p_c or p_c ∈ Suspected_i
if received PHASE1(r_i, v) from p_c then
    return v
else
    return ⊥
end if
```

da propriedade de **Terminação** apóia-se na propriedade de abrangência forte do detector de falhas $\Diamond S$, a qual evita uma espera indefinida por valores do coordenador da rodada, garantindo respostas para qualquer invocação de LAMBDA(-,-). Por fim, a validação da propriedade de **Convergência após um tempo** também apóia-se nas propriedades do detector de falhas, nesse caso, a propriedade de exatidão forte após um tempo. De acordo com esta propriedade, existirá um tempo a partir do qual todos os processos corretos não irão suspeitar de algum processo correto que poderá ser escolhido como coordenador de uma rodada.

4.2 Introduzindo ordenação de processos adaptativa no subsistema *consenso-GR* baseado em $\Diamond S$

O protocolo *consenso-GR* equipado com uma implementação de LAMBDA(-, -) baseada no detector de falhas da classe $\Diamond S$, representa um protocolo simétrico pelo texto. Nesse caso, a diferença entre os papéis aparece na implementação de LAMBDA(-, -), onde existe o papel do coordenador de uma rodada, o qual é responsável por enviar aos demais a proposta de valor consensual. Esta proposta de valor consensual deve ser usada para atualizar as

estimativas dos outros participantes, caso não haja suspeitas sobre o coordenador.

Na implementação de LAMBDA(-, -) baseada no detector de falhas $\Diamond S$, a escolha do coordenador de cada rodada é feita a priori e armazenada em uma lista de processos ordenada a partir dos identificadores dos processos, usando um critério bem definido. O coordenador da rodada r é o *r-ésimo* elemento da lista. Como foi discutido anteriormente, o desempenho de protocolos simétricos pelo texto, dependentes de ordenação de processos, está associado ao desempenho dos processos que desempenham papéis "especiais" (*e.g.* coordenador) e, indiretamente, ao valor da lista ordenada de processos. Isto porque as configurações de lista ordenada de processos refletem sobre o padrão de comunicação entre os participantes do protocolo, podendo ser eficientes ou não de acordo com a carga do ambiente de execução. Em ambientes sujeitos à carga heterogênea podem existir configurações mais eficientes do que outras e, o uso de uma configuração ineficiente causa degradação de desempenho. Se além de heterogênea a carga também for dinâmica, torna-se impossível garantir uma configuração eficiente a priori para protocolos simétricos pelo texto sujeitos à tal carga. Na implementação de LAMBDA(-,-) baseada em $\Diamond S$, o padrão de comunicação é representado pela troca de mensagens do coordenador para os demais participantes. Então, se o coordenador for "lento" o custo de comunicação para obter uma proposta de valor consensual será alto, degradando o desempenho da respectiva execução de LAMBDA(-,-) e, por conseguinte, do protocolo *consenso-GR* como um todo.

Soluções adaptativas para o problema da ordenação de processos podem tornar os protocolos dependentes de tal mecanismo mais eficientes em ambientes sujeitos à carga heterogênea e dinâmica. No Capítulo 3 apresentou-se uma solução genérica para ordenação de processos adaptativa aplicada a protocolos simétricos pelo texto. Na ocasião foi descrito um estudo de caso sobre um protocolo de consenso e demonstrada, analiticamente, a eficiência da solução. Nesta seção, propõe-se um novo estudo de caso para a solução adaptativa em questão, mas, dessa vez, considerando o subsistema *consenso-GR*. O subsistema adaptativo resultante foi denominado de subsistema *consenso-GRA*. É importante ressaltar que o novo estudo de caso tem um foco diferente daquele apresentado no Capítulo 3, nesse caso, deseja-se investigar o desempenho de um sistema equipado com a solução adaptativa usando avaliação experimental. Para tal, é preciso discutir e resolver questões de engenharia associadas ao projeto e implementação do subsistema adaptativo (objetivo deste capítulo) e, então, realizar a avaliação de desempenho (objetivo do Capítulo 5).

Em linhas gerais, o subsistema *consenso-GRA* executa várias instâncias de forma sequencial, onde cada instância representa uma execução do respectivo protocolo de consenso. No início de cada execução do consenso, o mecanismo de ordenação de processos adapta-

tivo é consultado, fornecendo uma lista ordenada de processos constituída de acordo com a latência fim-a-fim entre os participantes do subsistema. O mecanismo adaptativo usa um oráculo de latência para obter informações sobre a situação de carga do ambiente de execução.

A seguir, discute-se algumas decisões de engenharia sobre a solução para ordenação de processos adaptativa, incluindo o oráculo de latência e como tais decisões impactam na engenharia do subsistema *consenso-GRA*.

4.2.1 Engenharia da solução para ordenação de processos adaptativa

A solução adaptativa proposta no Capítulo 3 pode ser representada por um serviço distribuído, constituído de módulos locais independentes aos quais se tem acesso através de uma interface bem definida; os módulos locais executam em cada um dos participantes da aplicação (*e.g.*, protocolo de consenso simétrico pelo texto). A implementação do serviço deve satisfazer os requisitos funcionais que caracterizam uma solução para ordenação de processos (ver descrição de tais requisitos no Capítulo 3), como também, garantir a realização de ações de adaptação. Este último depende do conhecimento de informações sobre a situação de carga do ambiente de execução, que no caso do serviço de ordenação de processos adaptativo é representada pela latência fim-a-fim entre os processos. Tais informações são providas por um oráculo de latência. Portanto, o oráculo de latência desempenha um papel fundamental na concepção da solução adaptativa.

Em se tratando da engenharia do oráculo de latência, este também pode ser implementado como um serviço distribuído (OL), onde cada um dos seus módulos locais (OL_i) está associado a um módulo local do serviço de ordenação de processos adaptativo. A implementação do oráculo de latência satisfaz uma versão simplificada da definição apresentada no Capítulo 3, isto porque não é possível garantir o requisito de informação global inerente a tal definição, em um ambiente assíncrono, usando uma solução distribuída. Relembrando a definição em questão, tem-se que um oráculo de latência é uma função Lat cujo retorno corresponde à estimativa de latência entre dois processos p_i e p_j no instante de tempo t, como mostra a equação abaixo:

$$\forall(p_i, p_j, t) \in \Pi \times \Pi \times \mathbb{R}_+, \exists L \in \mathbb{R}_+ : Lat(p_i, p_j, t) = L$$

onde, Π é o conjunto de processos monitorados pelo oráculo de latência.

De acordo com a definição apresentada, um oráculo de latência provê informação glo-

bal, *i.e.*, qualquer consulta ao oráculo sobre o mesmo par de processos e no mesmo instante de tempo deve retornar a mesma estimativa de latência. Seja $Lat(p_i, p_j, t_1) = L_1$ e $Lat(p_k, p_m, t_2) = L_2$, se $(p_i, p_j, t_1) = (p_k, p_m, t_2)$, então $L_1 = L_2$.

A inexistência de garantias de tempo em um ambiente assíncrono combinado ao uso de uma solução distribuída, são os fatores que inviabilizam o requisito de informação global da definição de oráculo de latência apresentada acima. Nesse caso, em uma implementação distribuída do oráculo cada um de seus módulos locais pode acumular informações de latência diferentes para o mesmo par de processos, de modo que, no instante de tempo real t, as estimativas de latência fornecidas pelos mesmos sejam diferentes. Para garantir que o oráculo fornece informação de latência global, uma alternativa é substituir a noção de tempo físico pela noção de tempo lógico, onde o tempo lógico pode ser representado pelo valor de um contador. O critério de incremento do tempo lógico (grão) deve ser o mesmo para todos os módulos locais do oráculo de latência, dessa forma, tem-se garantias sobre o valor e o significado do tempo lógico. Em outras palavras, a idéia é discretizar o tempo físico, obtendo um tempo lógico sobre o qual existem garantias.

Considerando o requisito supracitado, simplificou-se a definição original de oráculo de latência, originando a definição representada pela Equação 4.1.

$$\begin{aligned} &\forall (p_i, p_j, t) \in \Pi \times \Pi \times \mathbb{N}^*, \exists L \in \mathbb{R}_+ : Lat(p_i, p_j, t) = L \quad onde, \\ &\qquad L \text{ é uma estimativa de latência e } t \text{ é um tempo lógico.} \end{aligned} \tag{4.1}$$

É ainda importante ressaltar que o oráculo de latência definido (seja pela definição original ou simplificada) provê estimativas de latência as quais possuem um erro (para mais ou menos). Nesse caso, um requisito de qualquer implementação para tal definição é a necessidade de minimizar o erro associado às estimativas do oráculo. Isto porque, quanto menor o erro maior a precisão do oráculo e, por conseguinte, melhor a eficiência da solução adaptativa baseada no mesmo. Em particular, as características de um ambiente assíncrono podem dificultar a garantia do requisito citado, principalmente, quando se considera estimativas de latência no futuro. Estratégias bem conhecidas para gerar estimativas de latência no futuro correspondem ao uso de modelos de previsão (NUNES; JANSCH-PôRTO, 2002) ou informações sobre o passado recente. Em ambos os casos, as estimativas são construídas a partir de amostras de latência coletadas pelo oráculo, sendo assim, quanto maior a amostra menor o erro associado à estimativa de latência. Além disso, o conteúdo da amostra deve ser atualizado com uma certa frequência, já que o passado recente é um bom indicador do futuro próximo.

4.2.2 Engenharia do subsistema *consenso-GRA*

O subsistema *consenso-GRA* usa um serviço de ordenação de processos adaptativo equipado com um oráculo de latência distribuído que satisfaz a definição apresentada na Equação 4.1. Nesse caso, a implementação do oráculo de latência é baseada em tempo lógico, representado por um contador, cujo grão é uma instância de execução do subsistema de consenso. Isto significa que, quando cada participante do subsistema de consenso inicia uma nova execução, seu contador (tempo lógico) é incrementado e tal informação torna-se conhecida pelo módulo local do oráculo de latência associado ao mesmo. Portanto, durante cada execução do consenso a informação provida pelos módulos locais do oráculo de latência deve ser global.

O serviço de ordenação de processos adaptativo pode usar a informação global provida pelo oráculo para calcular o tempo de terminação do consenso para cada lista ordenada de processos, obtida a partir da permutação do conjunto de participantes do subsistema *consenso-GRA*, e assim, escolher a lista ordenada de processos mais eficiente, *i.e.*, que resulte no melhor tempo de terminação do consenso. Esta estratégia foi usada no estudo analítico sobre o desempenho do *consenso-CT* (CHANDRA; TOUEG, 1996), apresentado no Capítulo 3. Note que quanto maior o número de participantes do consenso, maior será a complexidade associada ao uso da estratégia em questão; o número de possibilidades para a lista ordenada de processos é $n!$, onde n é o número de participantes do consenso. Devido a esta razão, optou-se por usar uma estratégia mais simples, a qual consiste em aplicar uma função determinística para ordenar os processos (participantes do subsistema de consenso) de acordo com a informação global provida pelo oráculo. Em outras palavras, os processos são ordenados, por cada módulo local do serviço de ordenação, com base na carga do ambiente de execução e usando o mesmo critério de ordenação.

A estratégia escolhida para gerar a lista ordenada de processos pelo serviço de ordenação adaptativo garante "atomicidade" sobre o valor da lista; isto tem relação direta com o fato do oráculo de latência prover informação global. Os demais requisitos funcionais da ordenação de processos também são satisfeitos. Partindo do pressuposto de que o oráculo de latência sempre responde às consultas do serviço de ordenação, a garantia do requisito de "terminação" depende do término da função determinística que ordena os processos, enquanto o requisito de "validade" é garantido pela própria definição do oráculo de latência, onde este fornece informação sobre todos os participantes do sistema.

4.3 Questões de projeto do subsistema *consenso-GRA*

O subsistema *consenso-GRA* foi projetado seguindo a abordagem modular funcional proposta em (FRIEDMAN; RAYNAL, 2004) (ver Capítulo 2). Esta abordagem enfatiza o uso de separação de conceitos de forma estruturada. Nesse caso, a arquitetura do subsistema em questão pode ser representada pela Figura 4.3.

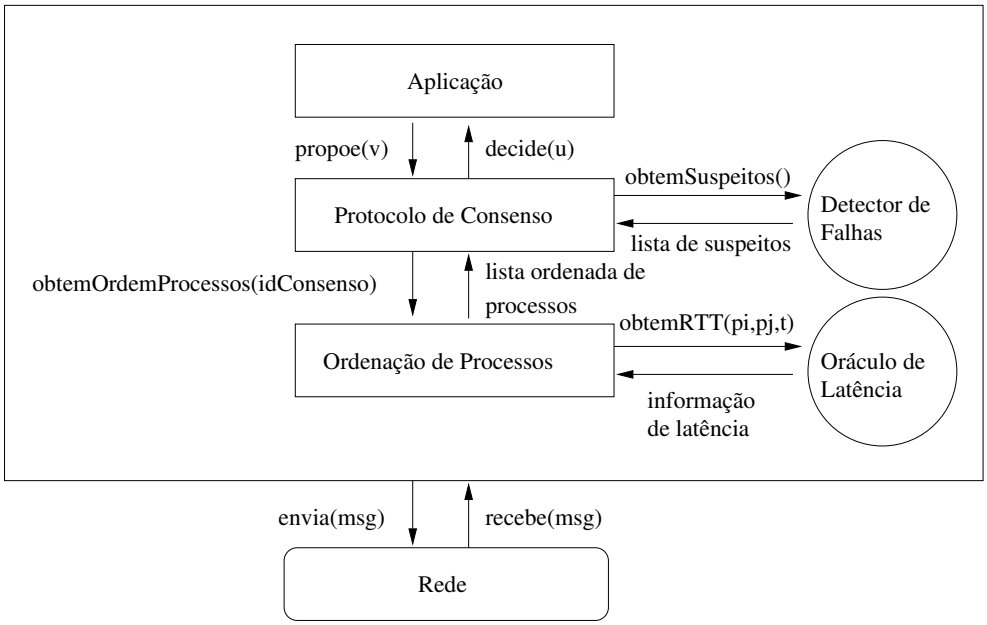


Figura 4.3: Arquitetura do subsistema *consenso-GRA*

O subsistema de consenso é composto por um conjunto de serviços distribuídos, onde cada um deles é constituído de módulos locais que executam em cada um dos participantes do subsistema e se comunicam para fornecer uma determinada funcionalidade. Os serviços possuem uma especificação bem definida e oferecem uma interface de acesso à funcionalidade provida. Como foi dito anteriormente, a aplicação (sistema de arquivos distribuído baseado em fragmentação-redundância-disseminação) é o "usuário" do subsistema de consenso, sendo responsável por iniciar novas execuções do protocolo *consenso-GRA*. Isto é feito através de uma chamada à função PROPOE(V), enquanto a decisão obtida em consenso é comunicada à aplicação por meio de chamada à função de retorno DECIDE(U).

O protocolo *consenso-GRA* tem acesso a um detector de falhas da classe $\Diamond S$ através de uma função OBTEMSUSPEITOS() que retorna uma lista de processos suspeitos. Além disso, o protocolo de consenso tem acesso ao serviço de ordenação de processos por meio de uma chamada à função OBTEMORDEMPROCESSOS(IDCONSENSO) a qual recebe como parâmetro o identificador da instância particular do consenso em execução (IDCONSENSO) e

retorna a lista de processos participantes do consenso ordenada segundo a situação de carga do ambiente. As informações sobre a situação de carga do ambiente são providas por um oráculo de latência através de chamadas à função OBTEMRTT(p_i, p_j, t) que retorna uma estimativa de latência de p_i em relação à p_j no instante de tempo lógico t =IDCONSENSO. A comunicação entre os componentes do sistema distribuído é dada por meio de troca de mensagens usando canais de comunicação confiáveis, nesse caso tem-se duas funções: ENVIA(MSG) e RECEBE(MSG), destinadas, respectivamente, ao envio e recebimento de mensagens. Note que qualquer componente pode ter acesso à rede através da interface citada, justificando a representação ilustrada na Figura 4.3.

Nas próximas seções serão detalhados os projetos dos serviços que compõem o subsistema *consenso-GRA* adaptativo destacando as ações e componentes associados a cada serviço.

4.3.1 O protocolo de consenso

A Figura 4.4 ilustra a arquitetura da solução para o protocolo *consenso-GRA*. A solução consta de dois componentes, **Seleção** e **Confirmação**, os quais implementam as respectivas fases nas quais o protocolo é dividido (ver Seção 4.1.1 para descrição detalhada do protocolo de consenso). Na etapa de seleção, os processos participantes do consenso irão escolher suas estimativas de valor consensual, enquanto na etapa de confirmação tais estimativas são difundidas e tenta-se decidir sobre uma delas. O valor decidido em consenso por cada participante é disseminado através de difusão confiável (CHANDRA; TOUEG, 1996).

O serviço de ordenação de processos adaptativo é consultado no início de cada execução do consenso, pelo componente **Seleção**, informando uma lista ordenada de processos. Esta lista será usada para escolher o coordenador de cada rodada da execução corrente do consenso. O serviço de detecção de falhas também se comunica com o componente **Seleção** a fim de fornecer informações sobre possíveis falhas no sistema; tais informações são usadas para decidir se o coordenador de uma rodada é considerado suspeito, fazendo o componente **Seleção** retornar um valor de estimativa nulo ($\perp$). De outra forma, o componente **Seleção** irá retornar a proposta consensual recebida do coordenador da rodada.

Serão consideradas execuções sequenciais do consenso, então, faz-se necessário a existência de um componente responsável por iniciar novas instâncias do protocolo de consenso e finalizá-las tão logo seja alcançado um valor consensual. Isto é realizado pelo componente **Sequenciador**.

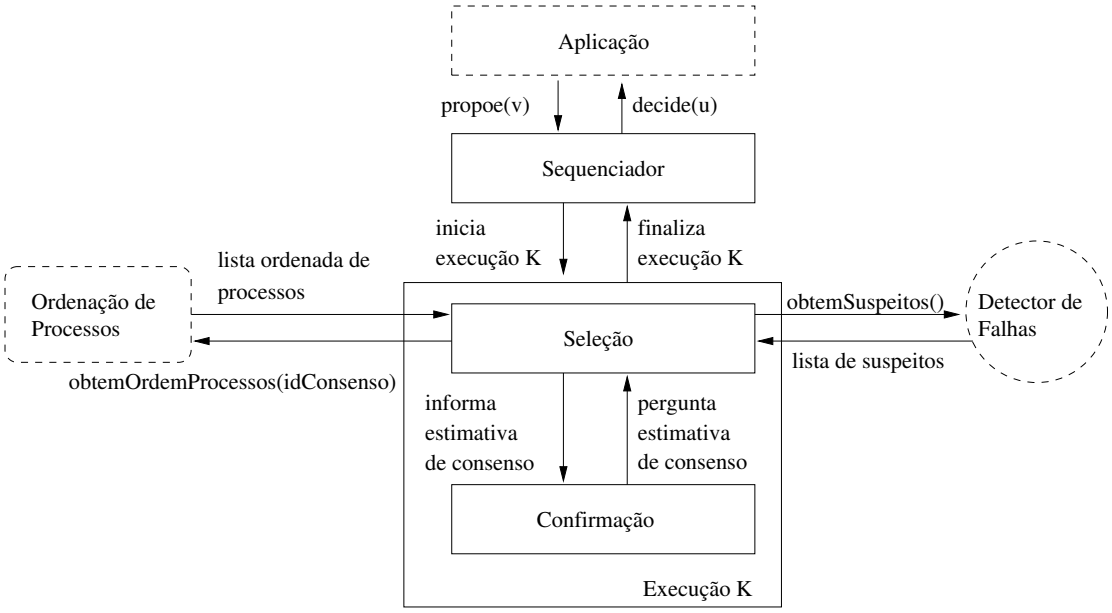


Figura 4.4: Arquitetura do serviço de consenso adaptativo que implementa o protocolo *consenso-GRA*

4.3.2 O serviço de detecção de falhas $\Diamond S$

Uma estratégia para implementar detectores de falhas não confiáveis em sistemas assíncronos é usando temporizadores. O protocolo de detecção de falhas segue o modelo cliente (monitores)/ servidor(monitorados), onde os clientes monitoram os servidores periodicamente e se não receberem respostas dos mesmos por um determinado período de tempo (valor do temporizador) caracteriza-se uma suspeita de falha. No subsistema *consenso-GRA* o monitoramento é de N-para-N, *i.e.*, os processos participantes do subsistema monitoram uns aos outros, comportando-se como clientes e servidores.

Foi adotado o modelo *pull* de monitoramento para detecção de falhas usando temporizadores (FELBER et al., 1999; SAMPAIO; BRITO; OLIVEIRA, 2003). Este modelo se baseia no envio periódico de mensagens do tipo "você está vivo?" para os processos monitorados. Logo após o envio dessas mensagens, é ativado um temporizador e se o detector não receber de volta uma mensagem do tipo "eu estou vivo!" daquele processo, antes do temporizador expirar, o processo é adicionado à lista de processos falhos. O serviço baseado no modelo *pull* possui três componentes, a saber: o **Emissor**, o **Temporizador** e o **DetectorPull**. Estes componentes são descritos abaixo, e a comunicação entre os mesmos é ilustrada na Figura 4.5.

- **Emissor**: envia, periodicamente, mensagens do tipo "você está vivo?" para o conjunto de processos monitorados.

- **Temporizador**: gerencia os temporizadores utilizados para esperar respostas provenientes dos processos monitorados, nesse caso, realiza ações de criar e remover temporizadores; se um temporizador expirar será caracterizado a ocorrência de uma suspeita de falha.

- **DetectorPull**: gerencia a lista de processos suspeitos, inserindo e removendo elementos; comunica-se com os componentes **Emissor** e **Temporizador**; é responsável por enviar mensagens do tipo "eu estou vivo!" como resposta às mensagens recebidas do tipo "você está vivo?".

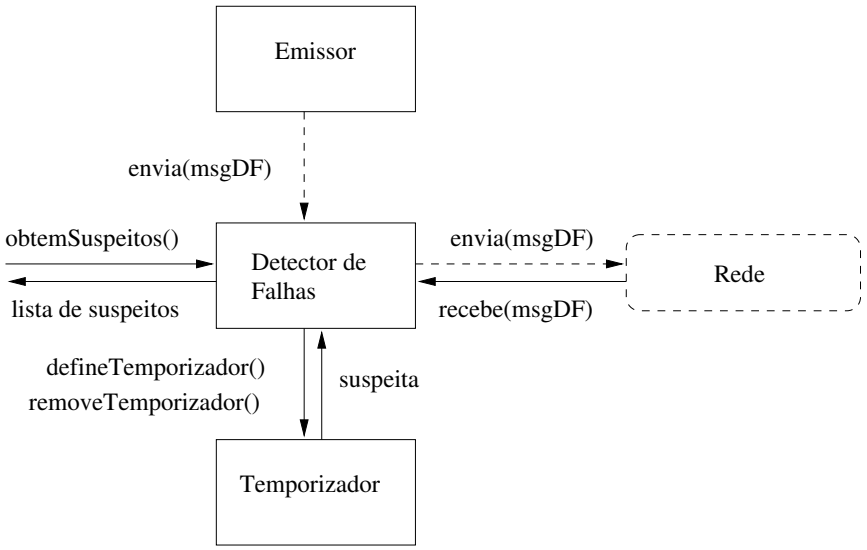


Figura 4.5: Arquitetura para o serviço de detecção de falhas $\Diamond S$

4.3.3 O oráculo de latência

O oráculo de latência é responsável por monitorar os participantes do subsistema *consenso-GRA*, coletando informação sobre a carga dos mesmos a qual será representada pelo tempo de ida e volta das mensagens trocadas por estes participantes (*round trip time - rtt*). Essas informações são coletadas por cada módulo local do oráculo, durante o ciclo de vida do subsistema *consenso-GRA*. Além disso, tais Informações são difundidas entre todos os módulos do oráculo de latência, de modo que cada módulo possa informar sobre a carga de qualquer participante do subsistema. Em outras palavras, o oráculo de latência provê informação sobre a carga do sistema como um todo. Entretanto, a informação provida representa uma visão local da carga do sistema. Para garantir uma visão global, o oráculo

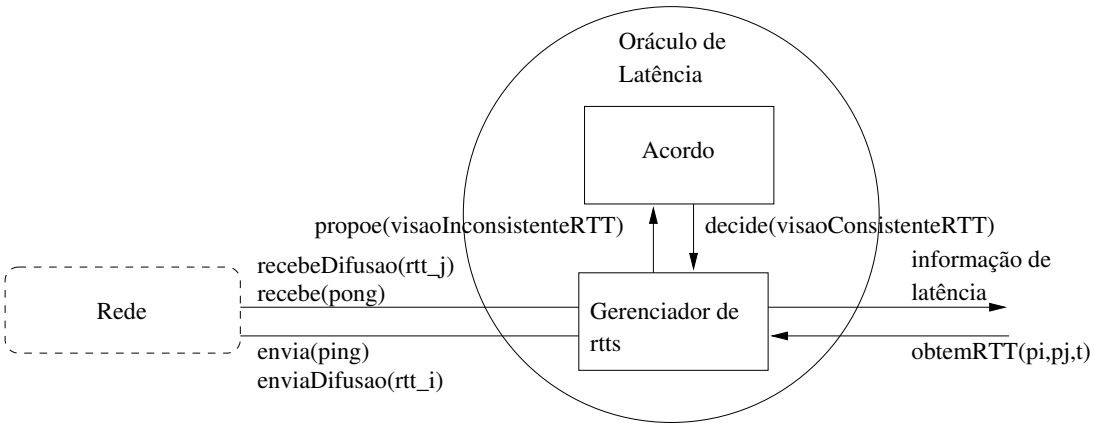


Figura 4.6: Arquitetura do oráculo de latência

de latência precisa usar uma solução de acordo. A Figura 4.6 descreve a arquitetura do oráculo de latência.

O oráculo de latência encapsula dois componentes, a saber: o **Gerenciador de rtts** e o **Acordo**. O componente **Gerenciador de rtts** é responsável por coletar e armazenar informações de latência localmente, para tal, implementa um esquema de *ping-pong*: envia, periodicamente, mensagens de controle (*ping*) e espera receber as respectivas respostas (*pong*) para calcular um *rtt*. Ao mesmo tempo, quando o **Gerenciador de rtts** recebe um *ping* deve responder imediatamente com o *pong*. Informações coletadas localmente são difundidas a todos os módulos locais do serviço do oráculo de latência através da função ENVIADIFUSAO(MSG), enquanto as mensagens difundidas são recebidas por meio da função RECEBEDIFUSAO(MSG).

Por outro lado, o componente **Acordo** é responsável por garantir a visão global da informação do oráculo de latência. Note que a solução de acordo empregada assume a existência de garantias de tempo, nesse caso, utiliza-se um tempo lógico. Para uma mesma unidade de tempo lógico, todos os módulos locais do oráculo de latência irão fornecer informação global sobre a carga do sistema. A solução de acordo pode ser representada por um protocolo de consenso, o qual é disparado com certa periodicidade, onde o valor proposto por cada um de seus participantes é a visão local da carga do sistema mantida por cada módulo local do oráculo de latência; a decisão do consenso representa a informação global do oráculo de latência (visão global da carga do sistema).

O oráculo é consultado através da função OBTEMRTT(p_i, p_j, t) que retorna informação de latência global, ou seja, tal informação é resultado da execução da solução de acordo discutida acima.

É interessante enfatizar que o custo[3] do oráculo de latência está associado, principalmente, às implementações de seus respectivos componentes. O **Gerenciador de rtts** gera mensagens extras para calcular os *rtts* entre os processos do sistema; já o componente de **Acordo** tem seu custo determinado pela sobrecarga da implementação da solução de acordo e quão frequentemente esta solução é executada. Este último fator influencia na exatidão do oráculo de latência, no sentido da informação de latência provida pelo mesmo tornar-se mais atualizada a cada nova execução da solução de acordo, porém, quanto maior a frequência de execuções da solução de acordo maior o custo associado ao oráculo de latência.

4.3.4 O serviço de ordenação de processos adaptativo

O serviço de ordenação de processos adaptativo foi descrito, em linhas gerais, na Seção 4.2. É bom lembrar que o início e término da execução do serviço acompanham o ciclo de vida do subsistema *consenso-GRA*, o qual consta da execução sequencial de várias instâncias do mesmo.

O serviço de ordenação de processos adaptativo encapsula um componente de **Adaptação** que tem por responsabilidade gerenciar a lista de processos participantes do *consenso-GRA*, modificando a ordem de seus elementos a partir das informações providas por um oráculo de latência. Tais ações de adaptação ocorrem no início de cada execução do consenso quando a lista de processos é requisitada por cada participante. É preciso garantir atomicidade sobre a lista ordenada de processos, assim, os participantes do consenso irão escolher o mesmo coordenador para cada rodada do protocolo. Isto é conseguido através da informação global provida pelo oráculo de latência. Figura 4.7 ilustra a comunicação entre o serviço de ordenação proposto e o oráculo de latência.

4.4 Questões de implementação do subsistema *consenso-GRA*

O subsistema *consenso-GRA* foi implementado em Java usando a versão 1.4.2_01 e seguindo os projetos descritos na Seção 4.3. De acordo com tais projetos, o subsistema *consenso-GRA* é constituído de uma série de serviços distribuídos. Apesar de cada serviço possuir uma especificação bem definida e uma interface de acesso conhecida, permitindo

[3]Define-se custo em função da sobrecarga de comunicação, sendo esta influenciada por 3 fatores: periodicidade no envio de mensagens, padrão de comunicação utilizado e quantidade de mensagens geradas.

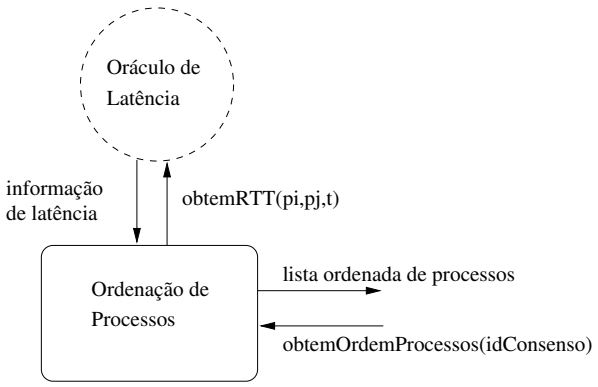


Figura 4.7: Arquitetura do serviço de ordenação de processos adaptativo

que os mesmos sejam utilizados por diferentes aplicações, optou-se por simplificar a implementação destes serviços para fins de análise de desempenho (a ser discutida no Capítulo 5). Nesse caso, a simplificação é no sentido da autonomia e generalização no uso dos serviços por parte de componentes do sistema no qual estão inseridos. Primeiramente, o início e término dos serviços acontece de maneira conjunta e, portanto, não possuem ciclo de vida independente. Além disso, alguns serviços mantém relações de dependência entre si, *e.g.*, o oráculo de latência e o serviço de detecção de falhas, onde este foi modificado para oferecer uma funcionalidade extra que permite essa relação. De fato, a implementação proposta para o subsistema *consenso-GRA* é destinada a uma aplicação específica (sistema de arquivos distribuídos baseado em fragmentação-redundância-disseminação).

Foi utilizada uma implementação de detector de falhas $\Diamond S$ baseada no modelo *pull* (*DF-S-Pull*). A implementação do oráculo de latência foi denominada de *OL*, sendo esta associada ao serviço de ordenação de processos adaptativo, denominado de *SOPA*.

O protocolo *consenso-GRA* usa o *SOPA* para ordenação de processos adaptativa, e o *DF-S-Pull* para detecção de falhas. A escolha por um detector de falhas do modelo *pull* é justificada por este tipo de detector ter a capacidade de calcular *rtts*, comportando-se como o componente **Gerenciador de rtts** para o serviço do oráculo de latência. Nesse caso, o número de mensagens geradas pela implementação do oráculo diminui e isto, possivelmente, causará um efeito positivo na sobrecarga (custo) introduzida pela mesma. O subsistema de consenso adaptativo usa um protocolo de difusão confiável para disseminar o valor consensual decidido. Foi implementado o protocolo de difusão confiável proposto em (CHANDRA; TOUEG, 1996).

O detector de falhas implementado não é adaptativo, mas usa temporizadores dinâmicos. O detector *DF-S-Pull* usa temporizadores, os quais são ajustados sempre que uma falsa suspeição for descoberta. De acordo com a estratégia de ajuste do temporizador, quando

uma resposta (mensagem do tipo "eu estou vivo!") é recebida de um processo p_j, o qual é considerado suspeito no momento, calcula-se o *rtt* da mensagem correspondente e este será usado como o novo temporizador para p_j. Outro aspecto importante da implementação é que, quando uma falsa suspeição para um processo p_j é detectada, este processo é removido da lista de suspeitos.

As implementações do serviço de ordenação adaptativo e seu respectivo oráculo são mais complexas e, por essa razão, serão descritas na próxima seção.

4.4.1 Implementações do serviço de ordenação de processos adaptativo e oráculos de latência

O serviço de ordenação de processos adaptativo *SOPA* foi instanciado com um oráculo de latência *OL* que satisfaz a definição dada pela Equação 4.1. Como foi discutido anteriormente, o oráculo de latência proposto possui um componente **Acordo** para garantir a informação global provida pelo mesmo. No caso do subsistema *consenso-GRA*, a implementação deste componente pode ser o próprio protocolo *consenso-GRA*. Nesse caso, uma execução do consenso decide valores para uma aplicação específica (*e.g.*, sistema de arquivos distribuídos baseado em fragmentação-redundância-disseminação) e para o serviço do oráculo de latência. O valor consensual retornado para o serviço do oráculo na *k-ésima* execução do consenso, corresponde à informação global a ser usada para construir a lista ordenada de processos requerida para a próxima execução do consenso, ou seja, $(k+1)$-*ésima* execução do consenso (para $k = 1$, utiliza-se uma lista ordenada de processos acordada a priori). Isto significa que, em cada execução do protocolo de consenso, o serviço de ordenação monta a lista de processos a partir de informação de latência globais sobre um passado recente (execução anterior do consenso). Além disso, a informação global do oráculo de latência é atualizada uma única vez a cada unidade de tempo lógico (uma instância de execução do subsistema *consenso-GRA*).

Cada participante p_i do subsistema *consenso-GRA* executa módulos locais dos serviços que compõem o subsistema. Informações sobre a situação de carga do sistema são armazenadas localmente, por cada módulo OL_i do serviço do oráculo de latência, em 3 matrizes de dimensão $n \times n$. Além disso, tais informações são representadas através do *rtt* entre cada par de processos. Sendo assim, o elemento $[i, j]$ de uma matriz representa o valor para o atraso de ida e volta entre os processos p_i e p_j. Todos os elementos de todas as matrizes têm valor inicial igual a zero.

A primeira matriz, chamada de $local_i$, é usada por um módulo OL_i para armazenar

sua visão local sobre a carga do sistema. A segunda matriz, chamada de $global_i$, representa a visão global corrente da carga do sistema acessível a todos os módulos do serviço OL. Ao final de cada execução do protocolo *consenso-GRA* é obtido um novo valor para $global_i$ a partir dos valores propostos por cada OL_i. Denomina-se tais propostas de estimativas correntes de visão global da carga do sistema, sendo estas armazenadas na matriz $estimativa\_corrente_i$. O valor inicial de $estimativa\_corrente_i$ é obtido da matriz $local_i$.

Periodicamente, todo módulo local OL_i calcula *rtts* entre p_i e os outros processos pertencentes a Π. À medida que novos *rtts* são calculados, OL_i atualiza o elemento $[i, j]$ correspondente na matriz $local_i$. Para tornar o oráculo de latência capaz de responder sobre a carga do sistema como um todo, *i.e.*, a latência entre todos os pares de processos em Π, todo módulo OL_i difunde $rtt_{i,j}$ para todos os elementos de Π. Então, quando um módulo OL_k recebe uma nova informação de latência, este atualiza o elemento $[i, j]$ da sua matriz $local_k$. Note que as informação da visão local de cada módulo OL_i podem ser difundidas usando as próprias mensagens trocadas pelos componentes do subsistema *consenso-GRA*. Esta estratégia evitaria a sobrecarga introduzida pelo envio de mensagens extras.

A matriz $global_i$ armazena informação global sobre a carga do sistema. Tal informação garante atomicidade sobre a lista ordenada de processos usada em cada execução do protocolo *consenso-GRA*. Nesse sentido, utiliza-se o próprio *consenso-GRA* da seguinte forma. O início de uma nova execução do protocolo é representada por uma chamada à função PROPOE(V) por parte de cada participante p_i, onde V encapsula dois valores: a estimativa da aplicação que iniciou o consenso (sistema de arquivos distribuídos baseado em fragmentação-redundância-disseminação) e a estimativa de visão global da carga do sistema de cada módulo OL_i (conteúdo da matriz $estimativa\_corrente_i$). Ao final do consenso, as decisões pertinentes à aplicação e ao serviço do oráculo de latência são comunicadas através da função DECIDE(U). O valor decidido em consenso será utilizado para atualizar a matriz $global_i$. É bom lembrar que a matriz $estimativa\_corrente_i$ é iniciada com o valor da matriz $local_i$ a cada nova execução do protocolo de consenso. A Figura 4.8 ilustra a dinâmica de atualização das matrizes $local_i$, $global_i$ e $estimativa\_corrente_i$ durante duas execuções do consenso ($k = 1, 2$). Nesse caso, as matrizes foram representadas de forma simplificada usando os valores 0 (valor inicial) e X_i.

O serviço *SOPA* é usado pelo protocolo *consenso-GRA* para gerar a lista de processos requerida em cada execução do mesmo. Cada participante p_i do consenso k tem acesso ao serviço de ordenação de processos adaptativo através do seu módulo local $SOPA_i$, usando a função OBTEMORDEMPROCESSOS(K). Quando p_i consulta $SOPA_i$ para saber a lista ordenada de processos da *k-ésima* execução do consenso, $k > 1$, este módulo pergunta ao

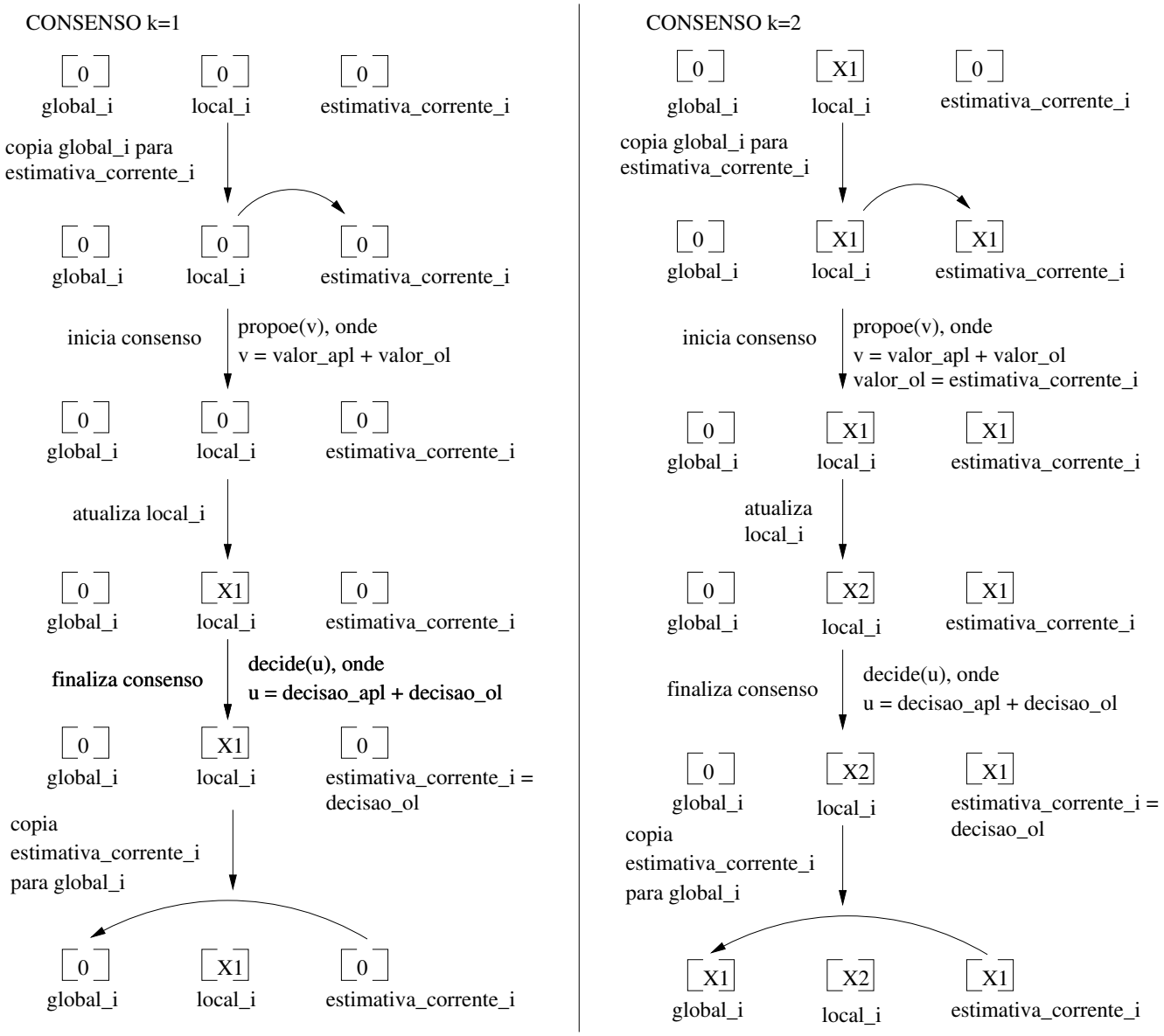


Figura 4.8: Dinâmica de atualização das matrizes manipuladas pelo oráculo de latência: $local_i$, $global_i$ e $estimativa\_corrente_i$

oráculo OL_i informações globais de latência para o tempo lógico corrente (instância k do consenso) e armazena as respostas, localmente, em uma estrutura idêntica à matriz $global_i$. Então, o serviço de ordenação $SOPA_i$ executa uma função determinística que recebe sua cópia local de $global_i$ como parâmetro e retorna a lista ordenada de processos de acordo com a situação de carga corrente do sistema. Na primeira execução do consenso ($k = 1$), o oráculo de latência ainda não possui informações sobre a carga do sistema. Nesse caso, a lista de processos será ordenada considerando o valor inicial da matriz $global_i$ de cada módulo OL_i.

No *consenso-GRA*, a obtenção de um valor consensual depende da existência de um coordenador correto que não seja suspeitado por, pelo menos, uma maioria de processos corretos ($n - f$ processos corretos, sendo f a hipótese de falhas e $n \geq 2f + 1$ o número de processos participantes do protocolo). Nesse contexto, uma função possível para ordenar processos de modo a favorecer coordenadores rápidos é a seguinte: seja $maioria\_rtt_i$ o $(n - f)$-ésimo menor rtt esperado para a comunicação entre p_i e qualquer outro processo (incluindo o próprio p_i), o qual é obtido a partir da matriz global consensual; cada módulo $SOPA_i$ calcula $maioria\_rtt_k$ para todo processo $p_k \in \Pi$, $1 \leq k \leq n$ e $n = |\Pi|$, gera como saída a lista de processos ordenada de forma crescente, de acordo com $maioria\_rtt$ (a ordem dos identificadores dos processos é usada como critério de desempate). A Figura 4.9 mostra um exemplo de aplicação da função descrita, considerando uma matriz $global_i$ de dimensão 5×5 e $\Pi_{e1} = \{p_1, p_2, p_3, p_4, p_5\}$.

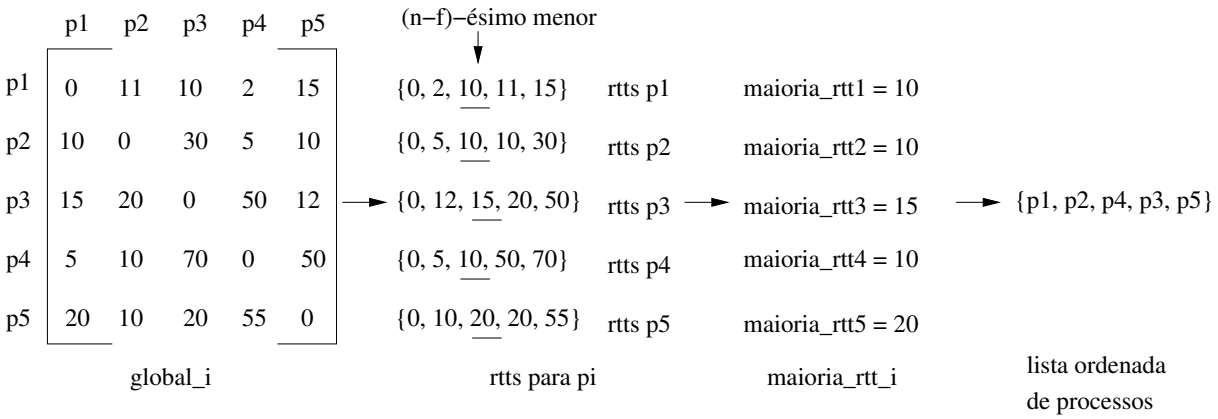


Figura 4.9: Exemplo de critério de ordenação baseado na mediana ($(n - f)$-ésimo menor) para 5 processos

O oráculo de latência OL informa o último rtt calculado para um dado par de processos em um tempo lógico específico. Esta estratégia pode ser inadequada, principalmente, quando a distribuição de probabilidade dos $rtts$ varia, significativamente, em um curto es-

paço de tempo. Alguns pesquisadores têm estudado este problema no contexto de projeto de oráculos de detecção de falhas (CHEN; TOUEG; AGUILERA, 2000; NUNES; JANSCH-PôRTO, 2004). Em particular, a abordagem baseada em preditores, proposta por Nunes e Jansch-Pôrto (NUNES; JANSCH-PôRTO, 2004), pode ser aplicada ao oráculo de latência em questão (SAMPAIO et al., 2005), resultando em uma versão otimizada do mesmo, a qual será denominada de *OL-P*. Seguindo tal abordagem, a arquitetura do oráculo *OL-P* constará de três componentes: o **Gerenciador de rtts**, o **Acordo** e o **Preditor** (ver Figura 4.10). O componente **Gerenciador de rtts** transmite os *rtts* calculados para o componente **Preditor** que aplica modelos de previsão específicos para tornar a informação sobre carga do sistema mais precisa. Nesse caso, a interface de comunicação entre o oráculo e o serviço de ordenação de processos é a mesma (chamadas à função OBTEMRTT(p_i, p_j, t)), porém, esta passa a ser oferecida pelo componente **Preditor**.

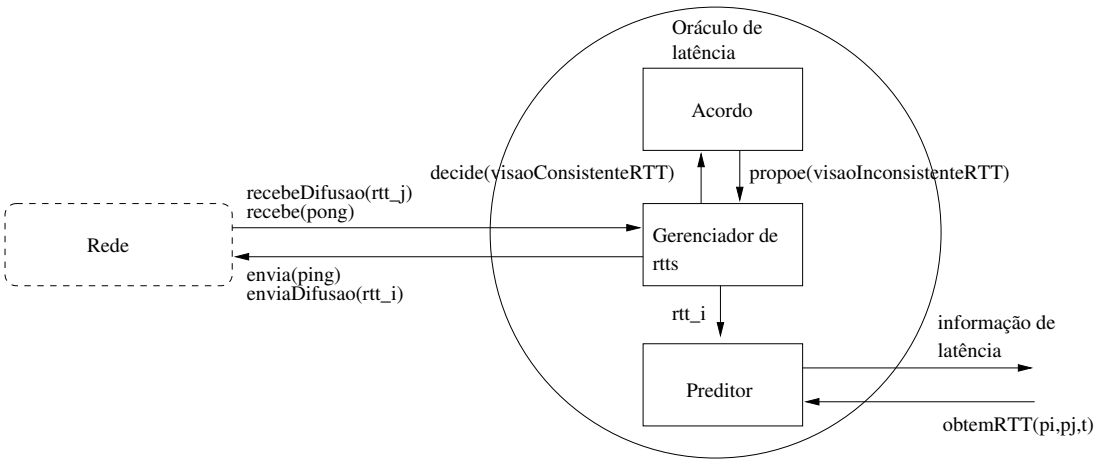


Figura 4.10: Arquitetura do oráculo de latência que usa preditores

Existem várias implementações possíveis de preditores (NUNES; JANSCH-PôRTO, 2004), dentre as quais foi utilizada uma implementação do preditor BROWN. Este preditor assume um modelo de crescimento linear e funciona como um filtro *double low-pass*. Por extrapolação, a previsão gerada pelo preditor BROWN segue $\widehat{rtt}_{t+1} = 2.rtt_t^{st} - rtt_t^{nd} + \frac{\alpha}{1-\alpha}.(rtt_t^{st} - rtt_t^{nd})$, onde $\alpha = 0.125$, rtt_t^{st} é uma variável de suavização exponencial simples, computada como $rtt_t^{st} = \alpha.rtt_t + (1-\alpha).rtt_{t-1}^{st}$, e rtt_t^{nd} é uma variável de suavização exponencial quadrado, computado como $rtt_t^{nd} = \alpha.rtt_t^{st} + (1-\alpha).rtt_{t-1}^{nd}$.

4.5 Conclusões parciais

Neste capítulo foram descritos o projeto e a implementação do subsistema *consenso-GRA* que está inserido em um sistema de arquivos distribuído para tolerância a intrusões na Internet (DESWARTE; POWELL, 2004, 2006). O subsistema encapsula um protocolo de consenso simétrico pelo texto, sobre o qual foi aplicada uma solução adaptativa baseada em oráculos de latência para ordenação de processos.

O problema da ordenação de processos caracteriza os protocolos simétricos pelo texto cuja atribuição de papéis aos respectivos processos é feita através de uma lista ordenada de processos. No caso do protocolo *consenso-GRA*, a lista ordenada de processos é usada para escolher a identidade dos processos que desempenham o papel de coordenador em cada rodada do protocolo. Podem ser encontrados na literatura exemplos de protocolos simétricos pelo texto dependentes de ordenação de processos em vários contextos (e não apenas no contexto do consenso) (CHANDRA; TOUEG, 1996; LARREA; FERNáNDEZ; ARéVALO, 2000; RICCIARDI; BIRMAN, 1991; AGUILERA et al., 2001; HURFIN et al., 1999; GUERRAOUI; LARREA; SCHIPER, 1995).

A solução para ordenação de processos adaptativa usa um oráculo de latência cuja definição é uma simplificação daquela proposta no Capítulo 3. A simplificação foi necessária para permitir a implementação do oráculo de latência em um sistema assíncrono por meio de uma solução distribuída. De acordo com a definição simplificada, o oráculo de latência provê estimativas de latência globais para qualquer instante de tempo lógico. Portanto, foi introduzida a idéia de "tempo lógico". Ainda sobre questões de engenharia da solução adaptativa, foi decidida a forma como a solução seria usada no subsistema *consenso-GRA*, considerando-se execuções sequenciais do subsistema, onde em cada execução seria consultada a solução para ordenação de processos. Além disso, enfatizou-se a necessidade de minimizar os erros associados às estimativas de latência do oráculo para garantir a eficiência da respectiva solução adaptativa.

Foi definido custo em função da sobrecarga de comunicação. Dessa forma, o custo de introduzir adaptação no subsistema *consenso-GRA* é influenciado pelas implementações da solução para ordenação de processos adaptativa e do oráculo de latência. De fato, o oráculo de latência é responsável pela maior sobrecarga que, por sua vez, está associada ao 1) mecanismo de monitoramento dos processos do sistema para coletar informações de latência, como também, à 2) solução de acordo usada para garantir informação global. Foram propostos artifícios para amenizar os custos causados por tais fatores. Em relação ao primeiro fator, o artifício foi usar mensagens de aplicações do ambiente de execução

para calcular a latência entre os processos monitorados, nesse caso, a aplicação escolhida foi um serviço de detecção de falhas baseado no modelo *pull*. Quanto ao segundo fator, o artifício foi usar o próprio protocolo *consenso-GRA* como solução de acordo do oráculo de latência, nesse caso, em cada execução do protocolo, a decisão alcançada reunia valores consensuais para uma aplicação específica (que iniciou o consenso) e para o oráculo de latência. Portanto, o oráculo implementado garante informação global para cada instância de execução do consenso o que corresponde à unidade de tempo lógico adotada.

Outra definição de custo diz respeito à complexidade. Vale ressaltar que o uso de adaptação aumenta a complexidade do sistema. Uma forma de amenizar tal complexidade é não deixar o usuário pagar pela capacidade do sistema se adaptar, mas pela adaptação de fato. Sendo assim, o custo da adaptação só será percebido quando da ocorrência de ações de adaptação. Isto não é verdade para o subsistema *consenso-GRA*, nesse caso algumas atividades que permitem a realização das ações de adaptação acontecem ao longo da execução do subsistema (*e.g.*, coleta de informações pelo oráculo de latência e uso do respectivo protocolo de consenso como solução de acordo do oráculo de latência). Por outro lado, o uso de princípios da engenharia, tais como separação de conceitos e modularização, no projeto e implementação do subsistema *consenso-GRA* também contribui para amenizar a complexidade introduzida pela adaptação. Em todo o caso, os benefícios alcançados pelo uso da adaptação superaram a sua complexidade (ver resultados apresentados no Capítulo 5), justificando sua aplicação no subsistema *consenso-GRA*.

A atividade de projetar e implementar um sistema contribui para a discussão sobre questões de engenharia que permitem fazer a transição entre a definição teórica e a concretização prática do sistema. Nesse contexto, pode ser necessário repensar a definição teórica para garantir a implementação do sistema nas condições desejadas, como foi observado para o subsistema *consenso-GRA*.

Capítulo 5

Avaliação de desempenho de um subsistema de consenso adaptativo baseado em oráculos de latência

Avaliação de desempenho é um processo que tem como finalidade entender e estimar (ou medir) o desempenho de sistemas computacionais usando métricas e modelos apropriados.

O uso de modelos e métricas realistas para avaliação de desempenho de protocolos distribuídos ainda é uma questão em aberto (KEIDAR, 2003; KEIDAR; RAJSBAUM, 2003), apesar da sua importância. Por outro lado, tem-se algumas contribuições nesse sentido. Isto envolve o uso de métricas temporais (tais como latência e vazão), considerações sobre contenção de recursos do sistema e escolha das técnicas para avaliação de desempenho, principalmente, o uso de técnicas combinadas (por exemplo, medição e simulação).

No contexto de protocolos de consenso para sistemas assíncronos com detectores de falhas não confiáveis, muitos trabalhos disponíveis na literatura usam, apenas, modelos analíticos e métricas pouco práticas, nesse caso, métricas de complexidade e métricas baseadas em tempo lógico (número de rodadas, número de passos de comunicação etc.), para analisar o desempenho dos protocolos (SCHIPER, 1997; HURFIN et al., 2001; DELPORTE-GALLET; FAUCONNIER, 2002; DUTTA; GUERRAOUI, 2002; GUERRAOUI; RAYNAL, 2004). A motivação para esta estratégia é a possibilidade de abstrair a noção de tempo quando se avalia soluções projetadas para o modelo de sistema em questão, pois, nesse caso, toda a noção de tempo e sincronismo estão encapsuladas no detector de falhas. Entretanto, este tipo de estratégia pode gerar interpretações equivocadas sobre o desempenho do sistema.

Dentre os trabalhos que consideram uma avaliação de desempenho fundamentada em suposições mais práticas, a maioria deles analisa como a qualidade de serviço dos detecto-

res de falhas (CHEN; TOUEG; AGUILERA, 2000) ou a carga extra de comunicação gerada pela implementação dos mesmos influencia o desempenho de protocolos de consenso (SERGENT; DéFAGO; SCHIPER, 2001; COCCOLI et al., 2002; URBáN et al., 2004)[1]. Ou seja, considera-se a influência de fatores externos sobre o desempenho dos protocolos. Por outro lado, a influência de características estruturais dos protocolos (fatores internos) sobre o seu desempenho tem sido pouco explorada, apesar dos resultados encontrados na literatura demonstrarem que esta influência é significativa (SAMPAIO et al., 2003, 2007). Além disso, muitos resultados de desempenho são obtidos a partir de experimentos realizados sobre um ambiente dedicado, onde apenas o serviço de detecção e o protocolo de consenso consomem recursos do sistema, *i.e.*, um ambiente onde não existe nenhuma outra fonte de contenção de recursos, seja de processamento ou comunicação. Nesse caso, a carga do ambiente de execução não pode ser considerada heterogênea e dinâmica.

Seguindo outros trabalhos já publicados na literatura, neste capítulo é descrita a avaliação de desempenho de protocolos de consenso. Porém, observa-se diferenças primordiais nos processos empregados. A primeira diferença refere-se ao objetivo da avaliação de desempenho, qual seja: avaliar o impacto de fatores internos sobre a eficiência de protocolos de consenso distribuídos simétricos pelo texto. Nesse caso, avaliou-se o impacto de um mecanismo de adaptação baseado em oráculos de latência sobre o desempenho do subsistema *consenso-GRA*, proposto no Capítulo 4. A segunda diferença refere-se à escolha dos modelos de avaliação e métricas de desempenho. Realizou-se uma análise quantitativa de desempenho, usando métricas temporais, fundamentada em experimentos de simulação e de medição em um ambiente real. A última diferença está associada ao ambiente de execução dos experimentos o qual está sujeito à carga (atrasos fim-a-fim na comunicação) heterogênea e dinâmica; este cenário tem sido pouco explorado, mas representa um cenário realista para avaliar o uso de adaptação em protocolos distribuídos. As decisões ao longo do processo de avaliação de desempenho foram conduzidas a partir de uma abordagem sistemática (JAIN, 1991) cuja finalidade foi facilitar a aquisição de informações sobre desempenho para entender melhor os sistemas estudados. É bom lembrar que o modelo de sistema adotado foi descrito no Capítulo 3.

O restante desse capítulo está organizado da seguinte forma. Na Seção 5.1 introduz-se o arcabouço NEKO, usado para dar suporte aos experimentos de simulação e medição

[1]De fato, pode existir uma influência do serviço de detecção sobre o desempenho dos protocolos que o utiliza, mas é preciso enfatizar que o detector de falhas deve ser implementado de modo a oferecer o melhor serviço de detecção possível, então, se isto influenciar no desempenho de outros serviços será uma consequência.

realizados. Em seguida, na Seção 5.2, descreve-se a preparação da avaliação de desempenho no NEKO, enfatizando alguns parâmetros importantes e suas configurações. As Seções 5.3 e 5.4 relatam os experimentos de simulação e medição, respectivamente, como também os resultados alcançados. A Seção 5.5 finaliza este capítulo trazendo algumas conclusões acerca dos resultados obtidos.

5.1 O arcabouço NEKO

A ferramenta de suporte para os experimentos realizados foi o **NEKO** (URBáN; DéFAGO; SCHIPER, 2001), um arcabouço baseado em Java que oferece uma plataforma de comunicação para avaliação de desempenho de protocolos distribuídos. Um dos atrativos do arcabouço é a possibilidade de avaliar um protocolo por meio de simulação e medições na rede real usando uma única implementação do mesmo. Além disso, o arcabouço foi projetado com o objetivo de ser simples, extensível e fácil de usar.

O arcabouço NEKO é constituído por três componentes, a saber: aplicação, *NekoProcess* e rede (ver Figura 5.1 (URBáN; DéFAGO; SCHIPER, 2001)). O componente aplicação descreve o protocolo distribuído a ser implementado, enquanto o componente rede representa a infra-estrutura de comunicação necessária para executar o protocolo distribuído. Já o *NekoProcess* é um componente de controle, criado pelo arcabouço, responsável por gerenciar a aplicação e interligá-la com a infra-estrutura de comunicação.

Uma aplicação é formada por um conjunto de N processos que se comunicam através da troca de mensagens sobre a infra-estrutura de comunicação, usando as primitivas SEND e DELIVER para envio e recebimento de mensagens, respectivamente. Cada processo está organizado, normalmente, como uma hierarquia de camadas, as quais representam módulos da aplicação. Em relação à infra-estrutura de comunicação, esta pode ser constituída por uma ou mais redes. Sendo assim, o NEKO permite que processos diferentes de uma aplicação usem redes diferentes em uma mesma execução. O usuário tem a flexibilidade para instanciar as redes a partir de modelos pré-definidos do NEKO, como também, outros modelos criados ou adicionados pelo próprio usuário. Além disso, não há restrições quanto à organização da aplicação seguindo estratégias diferentes daquela em camadas. O mesmo se aplica à definição das primitivas de comunicação.

NekoProcess é um objeto criado durante a inicialização do arcabouço e associado a todo processo da aplicação, podendo ser considerado a primeira camada de uma aplicação NEKO. Em outras palavras, o *NekoProcess* é uma entidade especial do arcabouço NEKO que desempenha algumas funções de controle. As funcionalidades atribuídas ao *NekoPro-*

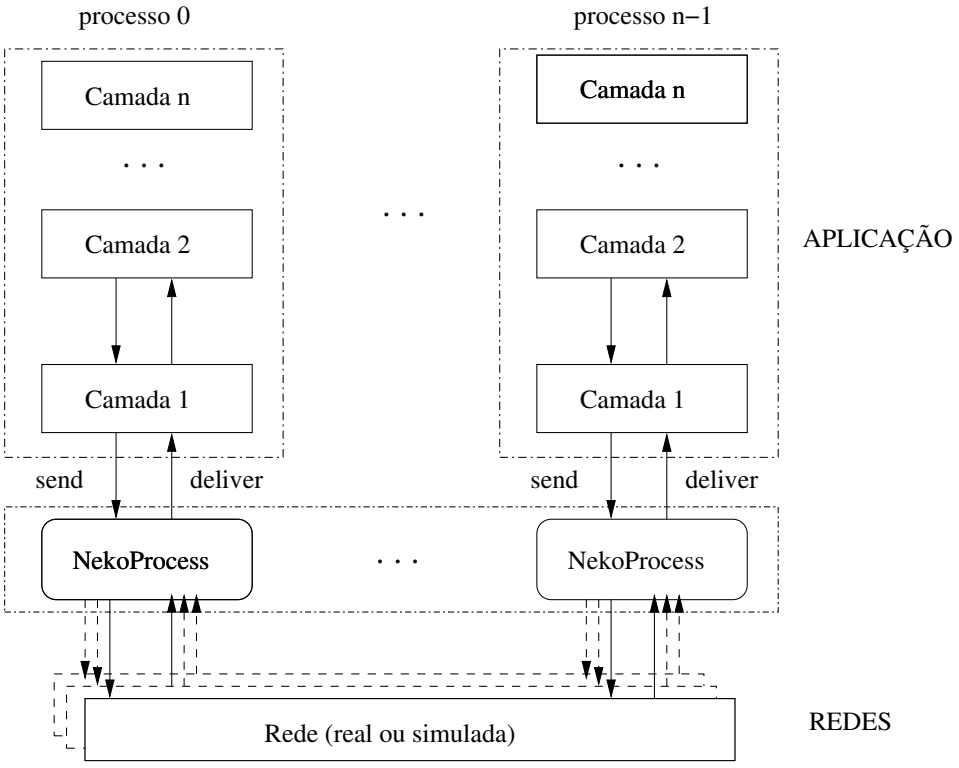


Figura 5.1: Arquitetura de uma aplicação no NEKO

cess incluem: armazenamento de algumas informações sobre o processo da aplicação (*e.g.*„ endereço e identificador); provimento de serviços úteis à aplicação (*e.g.*„ log de mensagens e terminação da aplicação); suporte à comunicação entre os processos da aplicação, principalmente, o que se refere ao gerenciamento de várias redes ativas em paralelo (*e.g.*„ envio e recebimento de mensagens para/de as redes).

A inicialização de uma aplicação NEKO é feita através de um único arquivo de configuração, passado como parâmetro de entrada para o arcabouço:

```
java        lse.neko.Main          arquivo_minha_aplicacao.config
     <programa principal do NEKO>   <arquivo de configuração>
```

As entradas de um arquivo de configuração são representadas por diretivas seguidas de seus respectivos valores (`diretiva = valor`). Existem diretivas pré-definidas do arcabouço comuns a ambos os tipos de execução (simulação e medição), como listado abaixo:

```
// primitivas pré-definidas, comuns tanto para simulação quanto medições
simulation = <value_boolean>
process.num = <value_int>
network = <value_string>
```

```
process.1.initializer = <value_string>
...
process.N.initializer = <value_string>

// primitivas pré-definidas, específicas para medições
master = <value1_string>
slave = <value_2_string>, <value_3_string>, ..., <value_N_string>
```

Através das diretivas do arcabouço NEKO é possível configurar o tipo de execução (`simulation =`), a quantidade de processos da aplicação (`process.num =`), o nome da entidade que implementa a rede (`network =`) e, o nome da(s) entidade(s) que implementa a inicialização de cada processo da aplicação (`process.i.initializer =`), o qual é identificado pela marca `i`. Outras diretivas podem ser definidas pelo usuário de acordo com os parâmetros da aplicação.

O NEKO permite que a mesma implementação de uma aplicação seja usada para execuções em ambientes simulados e reais. Note que existem algumas diferenças entre estes dois tipos de execuções. Por exemplo, a inicialização de uma execução na rede real é conduzida por um processo específico, denominado de "master", responsável por ler o arquivo de configuração e distribuir as informações de configuração para os demais processos, denominados de "slave". A denominação dos processos (hosts) que assumem os papéis de "master" e "slave", aparece no arquivo de configuração através das diretivas de mesmo nome (`diretiva = nome ou endereço IP do host`). Esta diferença entre os papéis é válida, apenas, durante a inicialização da aplicação. Outro aspecto relevante refere-se às entidades que controlam cada tipo de execução. A escolha por uma ou outra entidade é transparente ao usuário, ocorrendo em tempo de execução de acordo com informações no arquivo de configuração. Tais entidades são extensões de uma entidade genérica, denominada de *NekoSystem*, como ilustrado na Figura 5.2. A entidade que controla as simulações encapsula um simulador, nesse caso, o simulador padrão do NEKO é baseado no SimJava (`http://www.dcs.ed.ac.uk/home/hase/simjava`), mas é possível adicionar outros simuladores.

5.2 Preparação dos experimentos no NEKO

O objetivo da avaliação de desempenho realizada foi quantificar o impacto de usar adaptação em protocolos simétricos pelo texto, usando como estudo de caso o subsistema *consenso-GRA*, proposto no Capítulo 4. Para tal, optou-se por uma análise comparativa entre o

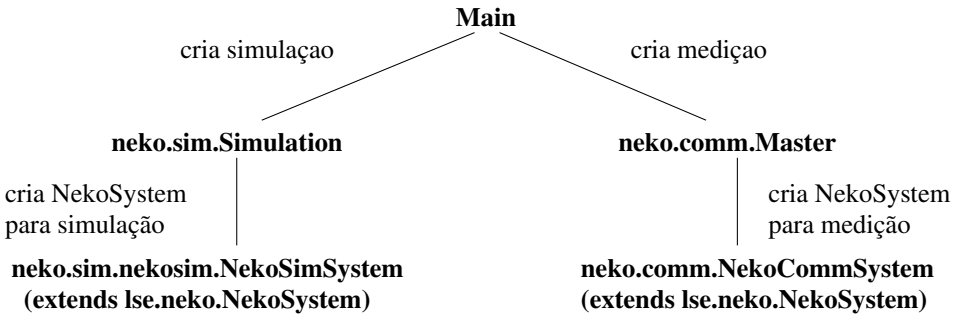


Figura 5.2: Hierarquia de inicialização do NEKO

subsistema *consenso-GRA* e seu correspondente não-adaptativo, denominado de subsistema *consenso-GR*. Duas diferenças caracterizam as versões adaptativa e não-adaptativa do subsistema de consenso em questão, quais sejam: mecanismo de ordenação de mensagens empregado e mecanismo de detecção de falhas. Enquanto o subsistema *consenso-GRA* usa uma lista de processos ordenada de acordo com informações providas por um oráculo de latência (ver descrição completa no Capítulo 4), o subsistema *consenso-GR* usa uma lista de processos definida a priori e passada como parâmetro para o protocolo de consenso. Esta lista é válida para uma sequência de execuções do protocolo. Nesse caso, não é requerida a existência de um oráculo de latência nem do serviço de ordenação de processos. De fato, este último é executado antes do protocolo, gerando parâmetros de entrada para o mesmo (valor da lista ordenada de processos).

Em relação ao serviço de detecção de falhas, tanto no subsistema *consenso-GRA* quanto no subsistema *consenso-GR* utiliza-se uma instanciação da classe $\Diamond S$ baseada em temporizadores, porém, a diferença reside nos modelos de detecção de falhas implementados, respectivamente, *pull* e *push*. O modelo *push* baseia-se no envio periódico de mensagens do tipo "eu estou vivo!" para os processos monitorados. O não recebimento de tais mensagens em um intervalo de tempo (temporizador) faz com que o processo seja incluído na lista de suspeitos. O projeto do detector de falhas *push* segue a mesma arquitetura especificada para o detector de falhas *pull*, descrito no Capítulo 4. Da mesma forma, a implementação deste detector de falhas, denominada de *DF-S-Push*, não é adaptativa, apesar, de usar temporizadores dinâmicos. O detector *DF-S-Push* usa temporizadores, os quais são incrementados de t_{adj} sempre que uma falsa suspeição for descoberta. Em ambos os detectores de falhas, quando uma falsa suspeição para o processo p_j é detectada, este processo é removido da lista de suspeitos.

Note que o modelo de detecção de falhas *pull* envolve o dobro de mensagens para monitoramento dos processos em comparação com o modelo *push*. Dessa forma, este gera

menor sobrecarga no ambiente de execução, sendo mais eficiente do que aquele (SERGENT; DéFAGO; SCHIPER, 2001). Isto justifica a escolha por um detector de falhas *push* para o subsistema *consenso-GR*. Por outro lado, a vantagem de um detector de falhas *pull* é a sua capacidade de contabilizar tempos de ida e volta entre os processos monitorados. Tal característica é relevante quando outros serviços do sistema necessitam de informação sobre tempos de ida e volta, assim, evita-se o envio de mensagens extras, diminuindo a sobrecarga introduzida pela implementação destes serviços. Este é o caso do subsistema *consenso-GRA* cujo mecanismo de adaptação usa um oráculo de latência que requer informação sobre tempos de ida e volta.

Portanto, os subsistemas de consenso foram construídos no NEKO (versão 0.8 - 24 Feb 2004) seguindo o modelo em camadas, como mostra a Figura 5.3. Nesse caso, a aplicação NEKO é representada pelo subsistema de consenso (*consenso-GR* ou *consenso-GRA*), a aplicação usuária do consenso (baseada no mecanismo de fragmentação-redundância-disseminação, frd (DESWARTE; POWELL, 2006)) e o teste de latência. Este último é composto por um conjunto de elementos (sub-camadas) responsáveis por gerenciar o início e término de cada execução dos respectivos protocolos de consenso, medindo latências de decisão e outras estatísticas de desempenho. O cálculo da latência de decisão para um participante do consenso p_k corresponde à duração do intervalo $[t_i, t_{f_k}]$, onde t_i é o tempo no qual a execução do consenso foi iniciada (após algum processo receber a requisição enviada pela aplicação) e, t_{f_k} é o tempo no qual p_k alcança o valor consensual. Note que t_i deve ser o mesmo para todos os participantes de uma execução do consenso, garantindo, assim, consistência no cálculo das latências. Quanto à representação de tempo no NEKO, configurou-se o arcabouço para usar precisão em milisegundos (ms).

A avaliação dos subsistemas de consenso consistiu de experimentos em ambientes simulados e reais, considerando cenários sem falhas ou falsas suspeições e um conjunto de $n = 5$ processos participantes[2], $\Pi_{e1} = \{p_1, p_2, p_3, p_4, p_5\}$, $n = |\Pi_{e1}|$. Nesse caso, p_1 corresponde ao "processo 0" do arcabouço NEKO e assim sucessivamente. Cada experimento envolveu 100 invocações consecutivas dos respectivos protocolos de consenso, sendo que o início de cada nova invocação era determinado pela aplicação, abstração de um sistema de arquivos distribuído baseado em fragmentação-redundância-disseminação, a qual se comporta como um gerador de requisições de consenso. A métrica de desempenho utilizada está associada à latência de decisão do consenso e equivale à média da menor latência de decisão para as 100 invocações de consenso realizadas em cada experimento.

[2]Os altos atrasos de comunicação observados em um sistema distribuído de larga escala, tal como a Internet, torna impraticável o desenvolvimento de serviços de consenso envolvendo muitos processos.

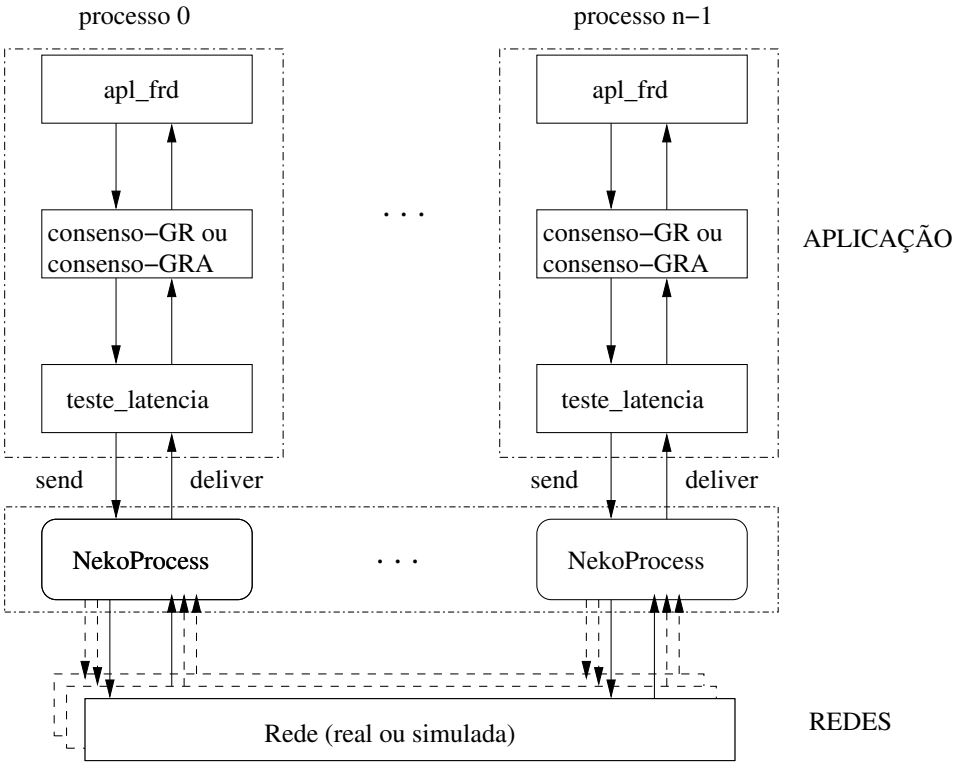


Figura 5.3: Arquitetura dos subsistemas *consenso-GR* e *consenso-GRA* usando o NEKO

Os experimentos foram configurados atribuindo valores para os parâmetros de entrada de cada subsistema de consenso, mais especificamente, dos componentes de tais subsistemas. Os parâmetros de entrada de alguns componentes dependem do método de avaliação de desempenho utilizado (simulação ou medições) ou do tipo de subsistema (adaptativo e não-adaptativo), o que será discutido em seções posteriores. Por outro lado, os detectores de falhas e a aplicação são parametrizados da mesma forma. Os detectores de falhas (*DF-S-Push* e *DF-S-Pull*) possuem dois parâmetros: periodicidade para envio de mensagens de controle do tipo "eu estou vivo!" e "você está vivo?", respectivamente, (η) e o valor do temporizador (δ); estes parâmetros assumem os valores $\eta = 1000ms$ e $\delta = 10^{12}ms$. A aplicação é parametrizada com um trace de carga de requisições, onde cada entrada representa o intervalo de tempo após o qual uma nova requisição de consenso deve ser emitida e, consequentemente, uma nova execução do respectivo protocolo de consenso deve ser disparada.

A aplicação representa um sistema de arquivos distribuído tolerante a intrusões, nesse caso, utiliza-se o mecanismo de fragmentação-redundância-disseminação para garantir segurança de dados na Internet (ver Capitulo 4). A consistência das operações sobre arquivos é garantida através de um protocolo de difusão atômica baseado em consenso, *i.e.*, orde-

nação total de mensagens (requisições de operações sobre arquivos). Assume-se que cada nova requisição dispara uma execução do protocolo de difusão atômica e, por conseguinte, uma execução do consenso. Para fins de análise de desempenho, a aplicação foi abstraída e implementada como um gerador de carga para os subsistemas de consenso em estudo. Esta carga corresponde aos intervalos de tempo entre execuções consecutivas do respectivo protocolo de consenso, sendo obtida a partir da carga de um sistema de arquivos distribuído real passada como parâmetro de entrada para a aplicação. Na próxima seção serão discutidos a origem e características desta carga, como também, a forma de utilização da mesma nos experimentos envolvendo os subsistemas de consenso em estudo.

5.2.1 Geração dos traces de carga da aplicação

Em (BODNARCHUK; BUNT, 1991) é apresentado um gerador de carga para um sistema de arquivos distribuído baseado no protocolo NFS (*Network File System*) para um ambiente UNIX. Inicialmente, foram realizados experimentos com o sistema de arquivos estudado para coleta de dados sobre a carga do mesmo. Nesse caso, o ambiente dos experimentos corresponde à rede de um laboratório de pesquisas acadêmicas, a qual engloba algumas dezenas de máquinas e cerca de 250 usuários. Os dados coletados foram analisados para construir o gerador de cargas.

Um tipo de carga gerada corresponde aos intervalos de chegada de requisições finais NFS para o sistema de arquivos. Por requisições finais NFS entenda-se operações sobre arquivos, tais como criação, remoção, leitura e escrita, como também, operações sobre diretórios, tais como criação, remoção e leitura. Em outras palavras requisições finais NFS são chamadas ao sistema que traduzem as ações requisitadas por um usuário. Vale salientar que operações de leitura e escrita sobre arquivos são as mais frequentes. A amostra coletada de intervalos de chegada de requisições possui média, mediana e variância de, respectivamente, $800ms$, $80ms$ e $6000ms$. O coeficiente quadrático da variância é de $2,98\%$, isto significa que tais dados seguem uma distribuição hiperexponencial. A distribuição hiperexponencial é representada por 3 distribuições exponenciais[3] associadas a uma probabilidade de ocor-

[3]Distribuições exponenciais são normalmente usadas para modelar variáveis aleatórias contínuas que representam o intervalo (de tempo ou comprimento) entre ocorrências de eventos sucessivos, tal como o tempo entre requisições a um servidor. Seja T o tempo entre as ocorrências de um evento (variável aleatória com distribuição exponencial) e λ a taxa média de ocorrências de um evento por unidade de tempo, então a função de distribuição de probabilidade de T é dada por $f(t) = \lambda e^{-\lambda t}$ (fdp) e, a função de distribuição acumulada de T é dada por $F(t) = 1 - e^{-\lambda t}$ (fda). Nesse caso, o valor de λ é dado em função da média ou variância de T: $E(T) = 1/\lambda$, $V(T) = 1/\lambda^2$ (BARBETTA; REIS; BORNIA, 2004).

rência, onde a distribuição com maior probabilidade usa $\lambda = 0,01$ (média de $90ms$) e, a de menor probabilidade usa $\lambda = 0,00004$ (média de $24.700ms$).

Os resultados obtidos em (BODNARCHUK; BUNT, 1991) serviram de base para definir os traces de carga da aplicação. Cada entrada do trace corresponde a um intervalo de chegada de requisições de operações sobre um sistema de arquivos distribuído. Nos experimentos com os subsistemas *consenso-GR* e *consenso-GRA*, intervalos de chegada de requisições representam intervalos entre execuções consecutivas do consenso, sendo assim, este valor é usado para calcular o tempo inicial de cada execução. Os traces são gerados a priori e passados como parâmetro para a aplicação. Os dados contidos nos traces seguem uma distribuição exponencial parametrizada com $\lambda = 0,0013$; o valor de λ foi calculado usando média de $800ms$ ($\lambda = 1/800$) que representa a média geral da amostra usada em (BODNARCHUK; BUNT, 1991).

A geração dos traces de carga da aplicação é descrita no Algoritmo 4. Foram gerados 100 traces de carga da aplicação, cada um deles contendo 100 entradas. Para tal, utilizou-se um programa Java (Exponential.java) que gera N valores de uma distribuição exponencial, recebendo como parâmetro o valor de λ especificado acima. Este programa usa a biblioteca de estatística do projeto Colt (`http://dsd.lbl.gov/~hoschek/colt/`).

Algoritmo 4 Geração dos traces de carga da aplicação

```
for all 1 ≤ i ≤ 100 do
    % trace_app_i é o nome do trace gerado
    java Exponential 0,0013 100 > trace_app_i
end for
```

5.3 Experimentos de simulação

5.3.1 Modelo da rede

Para os experimentos de simulação realizados foi desenvolvido um modelo de rede baseado em um modelo do arcabouço NEKO. Este último representa uma rede Ethernet simplificada com suporte à contenção de recursos de processamento e rede (URBáN; DéFAGO; SCHIPER, 2000).

O modelo de rede simulada representa uma rede comutada de larga escala (*WAN - Wide Area Network*) através da qual os processos se comunicam por troca de mensagens. Os processos estão conectados via canais de comunicação unidirecionais (ver Capítulo 4 contendo descrição do modelo do sistema), além disso, o modelo incorpora, apenas, contenção de

rede (canais de comunicação)[4]. Nesse caso, se um determinado canal de comunicação $c_{i,j}$ está ocupado quando p_i quer enviar uma mensagem m para p_j, então m permanece em uma fila de espera até que $c_{i,j}$ esteja disponível para transmitir esta mensagem. A Figura 5.4 ilustra o comportamento da rede simulada e seus componentes.

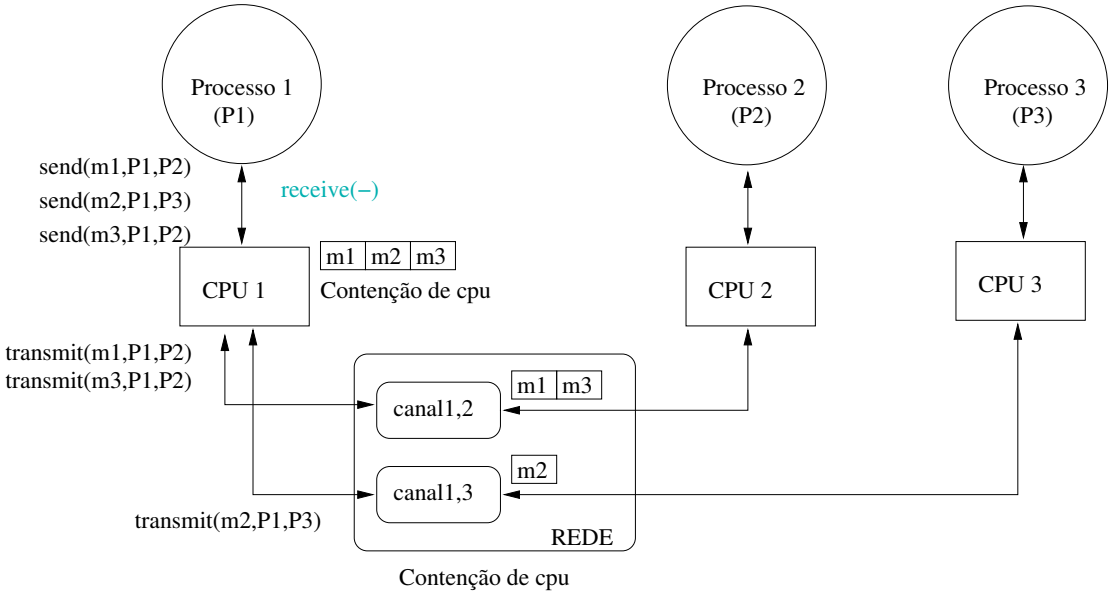


Figura 5.4: Arquitetura da rede de comunicação simulada no NEKO

Os atrasos na transmissão de mensagens através da rede simulada foram definidos a partir de traces de cargas reais, os quais descrevem o comportamento de uma rede *WAN*. A geração dos traces de carga da rede será descrita na próxima seção.

5.3.2 Geração dos traces de carga da rede

Os traces de carga da rede foram gerados a partir de dados de latência coletados no PlanetLab (PATERSON et al., 2003; SPRING et al., 2006), uma plataforma de testes distribuídos sobre a Internet (ver Capítulo 2 para obter mais informações). O primeiro passo foi configurar a "fatia" da rede destinada aos testes, sendo esta composta por 160 máquinas distribuídas por diferentes domínios. Em seguida, definiu-se a ferramenta para coletar os dados de latência, nesse caso, utilizou-se uma solução cliente/servidor, implementada em

[4]Um sistema distribuído real pode estar sujeito a cargas heterogêneas de processamento e comunicação. Sob o ponto de vista da aplicação (por exemplo, serviços de consenso), tal carga é percebida como o atraso fim-a-fim nas trocas de mensagens entre os processos do sistema. Para fins de simulação, pode-se anular a contenção de CPU e considerar, apenas, contenção de rede, pois não faz muita diferença se é a carga de processamento ou a carga de comunicação que exerce maior influência sobre o atraso fim-a-fim total.

C, que calcula latências entre duas máquinas quaisquer; as máquinas comunicam-se através de conexões TCP usando mensagens de 100 bytes. O cálculo das latências segue uma estratégia ping/pong, onde uma máquina clt_i (cliente) envia uma mensagem "ping" para outra máquina svr_i (servidor), a qual responde com uma mensagem "pong"; quando clt_i recebe a resposta de svr_i, este calcula o tempo de ida e volta (*round trip time* - *rtt*) do ping/pong que representa a latência entre clt_i e svr_i. Uma nova mensagem "ping" será enviada tão logo a resposta "pong" for recebida, sendo que, a cada 3 segundos realizam-se 3 pares "ping/pong" e grava-se os *rtts* calculados em arquivo.

A solução cliente/servidor original foi modificada no sentido de suportar novas configurações para o intervalo entre envio de mensagens "ping" e o intervalo de gravação de dados em arquivo. Nesse caso, a cada segundo realiza-se um par "ping/pong", gravando-se os *rtts* calculados a cada minuto. A decisão por tal configuração considerou os resultados de um teste usando a solução cliente/servidor para um grupo de 5 máquinas específicas no PlanetLab. O teste foi realizado com 4 configurações diferentes, quais sejam: 1) um par "ping/pong" a cada segundo e, gravação em arquivo a cada minuto; 2) um par "ping/pong" a cada 10 segundos e, gravação em arquivo a cada 2 minutos; 3) um par "ping/pong" a cada 30 segundos e, gravação em arquivo a cada 5 minutos; 4) um par "ping/pong" a cada 60 segundos e, gravação em arquivo a cada 10 minutos. Não foram observadas diferenças significativas entre os dados obtidos para cada configuração, escolhendo-se aquela com menor granularidade (1).

A coleta de dados no PlanetLab foi feita através de um conjunto de experimentos usando a solução cliente/servidor descrita acima. Os experimentos aconteceram no período de 10-02-2006 a 17-02-2006. Cada experimento consistiu da coleta de dados entre diferentes grupos de 5 máquinas escolhidas a partir de um conjunto de 160 máquinas. Para calcular as latências entre todos os pares de um grupo de 5 máquinas, $\{A, B, C, D, E\}$, seriam necessárias 20 combinações cliente/servidor como mostra a Tabela 5.1. Entretanto, considerando simetria na comunicação[5], o número de combinações diminuiu para 10 (ver Tabela 5.2). Sendo assim, a atribuição dos papéis de cliente/servidor para 5 máquinas obedece ao esquema descrito na Figura 5.5. A geração dos grupos de 5 máquinas para cada experimento segue o Algoritmo 5.

Realizaram-se 9 experimentos para coleta de dados de latência no PlanetLab, cada qual com 4 horas de duração e com uma quantidade de grupos variando de 10 a 32, variação esta decorrente do número de máquinas operacionais dentro do conjunto de 160 máquinas

[5]Note que simetria na comunicação é uma simplificação usada para facilitar a geração dos traces de carga na rede.

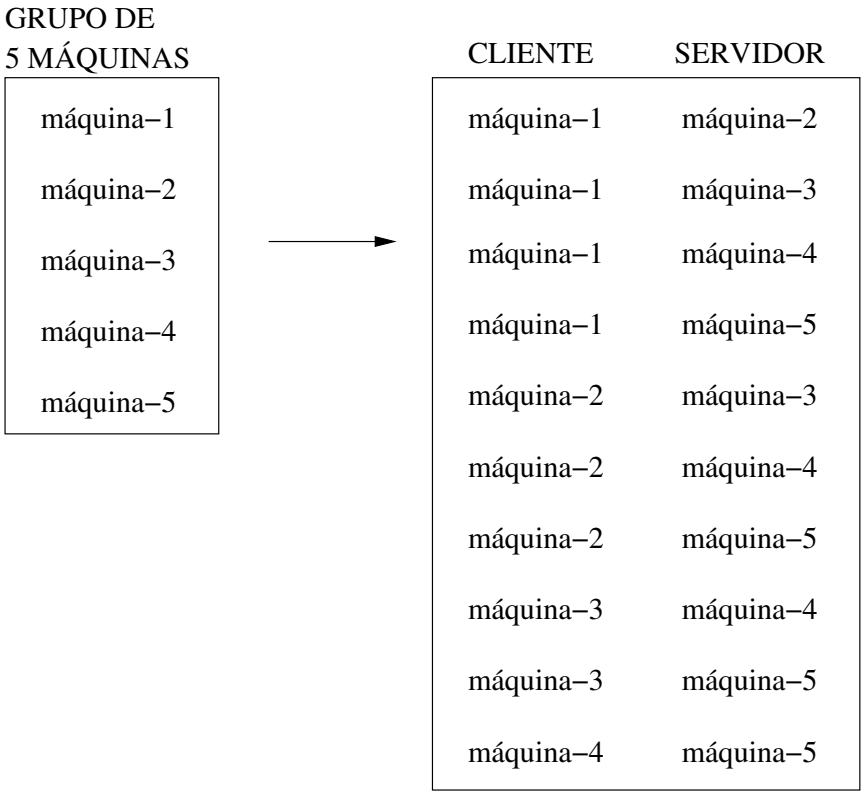


Figura 5.5: Atribuição dos papéis de cliente e servidor para um grupo de 5 máquinas

Algoritmo 5 Geração dos grupos de 5 máquinas

% para cada experimento:
for all $1 \leq i \leq quantidade\_grupos$ **do**
  for all $1 \leq j \leq 5$ **do**
    escolhe randomicamente uma máquina a partir de arquivo_160_máquinas (excludente)
    adiciona máquina ao grupo_i
  end for
  atribuição dos papéis de cliente/servidor para grupo_i; armazena em grupo_i_cs
end for

Servidor → / Cliente ↓	*A*	*B*	*C*	*D*	*E*
A	•	AB	AC	AD	AE
B	BA	•	BC	BD	BE
C	CA	CB	•	CD	CE
D	DA	DB	DC	•	DE
E	EA	EB	EC	ED	•

Tabela 5.1: Combinações cliente/servidor para um grupo de 5 máquinas

Servidor → / Cliente ↓	*A*	*B*	*C*	*D*	*E*
A	•	AB	AC	AD	AE
B	~~BA~~	•	BC	BD	BE
C	~~CA~~	~~CB~~	•	CD	CE
D	~~DA~~	~~DB~~	~~DC~~	•	DE
E	~~EA~~	~~EB~~	~~EC~~	~~ED~~	•

Tabela 5.2: Combinações cliente/servidor para um grupo de 5 máquinas usando simetria na comunicação

no momento de formação dos grupos. Ao todo, foram coletados dados completos para 81 grupos de 5 máquinas, isto porque, dos 212 grupos formados, alguns grupos falharam antes do término do experimento, resultando em dados incompletos que foram descartados.

A partir dos dados coletados no PlanetLab gerou-se 3 tipos de carga na rede: carga com picos, carga sem picos e carga de previsão. Para os três tipos de carga, os *rtts* foram divididos pela metade no sentido de representar a latência entre dois hosts na rede como o atraso de ida na transmissão de mensagens. Na carga com picos, nenhuma outra modificação foi imposta sobre os dados coletados, assim, tem-se 81 traces, onde cada um deles reúne os dados de latência para 1 grupo de 5 máquinas; tais dados estão organizados através de 10 arquivos de latência, um para cada combinação cliente/servidor estabelecida, os quais possuem 14.400 entradas (4 horas de experimento). Por outro lado a carga sem picos e a carga de previsão foram geradas a partir da aplicação de procedimentos específicos sobre os dados de latência em questão.

A carga sem picos foi obtida eliminando-se os valores extremos dos dados de latência. Valores extremos são causados por diferentes fatores, como por exemplo a ocorrência de eventos raros durante a coleta dos dados. A existência de valores extremos em uma amostra

de dados pode distorcer a interpretação dos mesmos, portanto, é importante identificar e saber lidar com os valores extremos. Uma estratégia para identificar valores extremos em uma amostra de dados é denominada de *BoxPlot*. Esta estratégia consiste em definir valores mínimo e máximo, de modo que, todo elemento da amostra de dados menor do que o parâmetro mínimo ou maior do que o parâmetro máximo será considerado um valor extremo. Os valores mínimo e máximo são obtidos a partir das expressões abaixo (DEVORE, 2000):

$$\begin{aligned} \text{valor mínimo} &= lower\_forth - (forth\_spread * 3) \\ \text{valor máximo} &= upper\_forth + (forth\_spread * 3) \end{aligned} \tag{5.1}$$

tal que,

$$lower\_forth = \begin{cases} \text{mediana dos n/2 menores elementos da amostra, se n é par} \\ \text{mediana dos (n+1)/2 menores elementos da amostra, se n é ímpar} \end{cases}$$

$$upper\_forth = \begin{cases} \text{mediana dos n/2 maiores elementos da amostra, se n é par} \\ \text{mediana dos (n+1)/2 maiores elementos da amostra, se n é ímpar} \end{cases}$$

$$forth\_spread = upper\_forth - lower\_forth$$

Aplicando a estratégia *BoxPlot* a uma amostra de dados representada pelo conjunto {5; 10; 82; 90,5; 52,1; 80; 110; 150; 470; 1.000}, tem-se:

$$\begin{aligned} lower\_forth &= 52,1 \\ upper\_forth &= 150 \\ forth\_spread &= 97,9 \\ \text{valor mínimo} &= -241,6 \\ \text{valor máximo} &= 443,7 \end{aligned}$$

qualquer valor fora do intervalo $[-241,6; 443,7]$ é um valor extremo, sendo assim, 470 e 1.000 são valores extremos dentro da amostra de dados considerada.

A estratégia *BoxPlot* foi usada para eliminar valores extremos dos dados de latência coletados no PlanetLab[6]. Com a eliminação dos valores extremos, o tamanho dos arquivos de latência que constituem cada um dos 81 traces de carga sem picos diminuiu, nesse caso, optou-se por uniformizar o tamanho dos arquivos para 10.800 entradas (3 horas de experimento).

Por sua vez, a carga de previsão foi obtida por meio do sistema de previsão da ferramenta NWS - *Network Weather Service* - versão 2.13 (WOLSKI; SPRING; HAYES, 1999; WOLSKI, 2003), o qual é acessível através de chamadas ao programa `add_forecast`. Tal sistema recebe como entrada um histórico de medidas para uma variável e produz como saída previsões futuras de medidas para esta variável. As previsões são geradas da seguinte forma: o histórico de medidas é submetido a um conjunto de modelos de previsão que fornecem previsões para cada medida considerando as medidas anteriores, então calcula-se o erro de previsão para cada modelo, escolhendo-se aquele com menor erro. O sistema de previsão é baseado em séries temporais, sendo assim, os dados manipulados pelo mesmo são descritos como um conjunto de medidas ordenadas no tempo. Os dados de latência coletados no PlanetLab foram submetidos ao sistema de previsão citado, produzindo traces de carga de previsão. Nesse caso, a organização dos traces é semelhante àquela apresentada para a carga com picos (81 traces com arquivos de 14.400 entradas), porém, a diferença está no conteúdo.

Os traces de carga gerados foram usados para configurar a rede de comunicação simulada. Cada experimento de simulação é parametrizado com um trace de carga, cujos arquivos de latência são numerados de forma crescente, de modo que a primeira entrada dos arquivos tenha índice zero ($ind = 0$), a segunda tenha índice um ($ind = 1$) e, assim sucessivamente. Os índices dos arquivos de latência são mapeados para o tempo de simulação corrente, obedecendo à equação descrita abaixo:

$$ind = \lceil \text{tempo\_de\_simulação\_corrente}/1000 \rceil + ind\_inicial \% \text{tamanho\_arquivo\_latência} \tag{5.2}$$

onde, $ind\_inicial$ é o índice a partir do qual o arquivo de latência deve ser pesquisado e tempo_de_simulação_corrente é o tempo lógico da simulação em ms. Seja o tempo inicial

[6]Note que eliminar picos de uma amostra de dados deve ser uma decisão bem fundamentada, pois, caso contrário, pode representar a exclusão de dados significativos, os quais devem ser entendidos. Sugere-se na literatura analisar o conjunto de dados com picos e sem picos, comparando os resultados. Esta sugestão foi aplicada neste trabalho, onde considera-se carga sem picos e carga com picos na geração dos traces de carga de rede.

de simulação igual a $0ms$, $ind\_inicial = 0$ e tamanho_arquivo_latência $= 14.400$, então, para tempo_de_simulação_corrente $= 2ms, 1.400ms, 10.000ms$ os respectivos índices nos arquivos de latência serão $ind = 0, 2, 10$. Isto significa que, $ind = k$ representa uma latência válida pelo $(k + 1)$-ésimo segundo do experimento (*e.g.*,, $ind = 0$ representa uma latência válida pelo 1^o segundo do experimento).

5.3.3 Definição dos cenários de simulação

Os subsistemas *consenso-GR* e *consenso-GRA* foram simulados em diferentes cenários. Cada cenário é a combinação de valores para as seguintes variáveis: 1) lista ordenada de processos, 2) tipos de traces de carga na rede, 3) quantidade de traces de carga da rede e, 4) quantidade de traces de carga da aplicação. Tais variáveis assumem mais de um valor.

A lista de processos possui 120 combinações possíveis, obtidas a partir da permutação dos 5 processos participantes dos subsistemas. No caso do subsistema *consenso-GR*, a lista ordenada de processos é configurada a priori, antes da execução do consenso. Enquanto no subsistema adaptativo, a lista de processos é configurada em tempo de execução, de acordo com a carga do ambiente, entretanto, a lista assume valores dentro do conjunto das 120 combinações. A rede de comunicação usa 3 tipos de carga (carga sem picos, carga com picos e carga de previsão), sendo que cada tipo de carga é composta de 81 traces. Já a aplicação, esta pode ser configurada com 100 traces diferentes. Portanto, o número de cenários de simulação possíveis para os subsistemas de consenso em estudo é descrito na Figura 5.6, ou seja, $3 * 120 * 81 * 100 = 2.916.000$ cenários para cada subsistema.

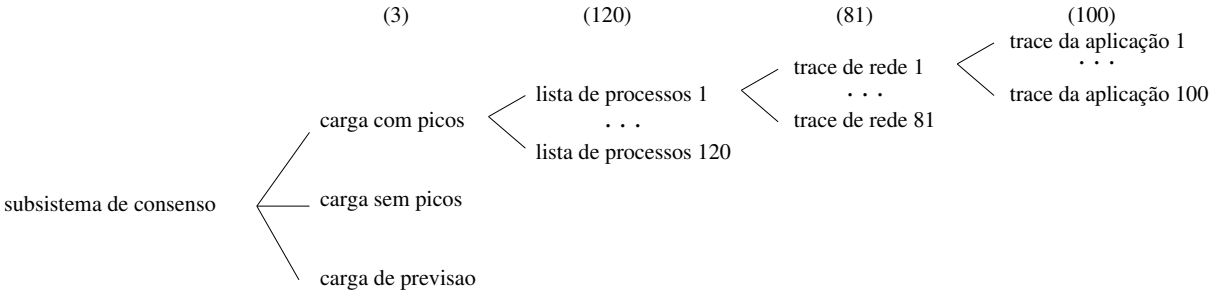


Figura 5.6: Cálculo do número de experimentos a serem realizados considerando os cenários de simulação especificados

Todo experimento de simulação envolve a execução de uma sequência de 100 consensos para um determinado cenário e, a métrica de desempenho é a média da menor latência de decisão para cada um dos 100 consensos. Para alcançar um certo nível de confiança sobre os resultados dos experimentos é preciso executá-los repetidas vezes, variando os

cenários. Nesta avaliação de desempenho, os cenários escolhidos devem conter os 3 tipos de carga de rede, variando, apenas, os valores da lista ordenada de processos, trace de carga da aplicação e trace de carga de rede. O cálculo do número de experimentos a serem realizados foi feito a partir dos resultados obtidos (em termos da média da menor latência de decisão do consenso) em uma amostra de $n\_amostra$ experimentos, usando a Equação 5.3.

$$
\begin{aligned}
n &= t_y^2 \frac{S^2}{E_o^2}, \\
S^2 &= \frac{1}{n\_amostra - 1} \sum_{i=1}^{n\_amostra} (x_i - \bar{x})^2
\end{aligned}
\tag{5.3}
$$

onde,

t_y = valor da variável aleatória T com probabilidade (nível de confiança) y;
T tem distribuição t de Student e, neste trabalho, representa o tempo de terminação do consenso.
E_o = erro amostral máximo tolerado, o qual é dado em função da unidade de medida da média amostral (ms,l, mg etc.).
$n\_amostra$ = tamanho da amostra.
S^2 = variância da amostra.

A amostra constou de $n\_amostra = 20$ experimentos para cada uma das 6 combinações subsistema de consenso/tipo de carga na rede. Isto significa que foram analisados os 2 subistemas de consenso para os 3 tipos de carga de rede, variando a lista ordenada de processos, trace de carga da aplicação e trace de carga de rede 20 vezes. O nível de confiança para os resultados dos experimentos foi de 95% com erro máximo de 5% do valor da média amostral. Para $n\_amostra = 20$ pode-se assumir que a média amostral segue uma distribuição de probabilidade t de Student[7], assim, $t_y = 2.093$. Os resultados do cálculo do número de experimentos para alcançar o nível de confiança desejado são descritos nas Tabelas 5.3 e 5.4. De acordo com os resultados, no pior caso (subsistema *consenso-GR* / carga de rede com picos) o número de experimentos necessários seria de $8.507 \approx 8.600$.

No sentido de acelerar a execução dos experimentos foi utilizado o sistema OurGrid, uma grade computacional cooperativa, com código aberto e livre acesso (CIRNE et al., 2006). O

[7]Para $n\_amostra > 30$ assume-se distribuição normal e, ao invés de usar a variável aleatória T deve-se usar a variável aleatória normal padrão Z.

Estatísticas	consenso-GR		
	carga_sem_picos	carga_picos	carga_previsão
n_amostra	20	20	20
n_total	7.826	8.507	2.551
latência de decisão (ms)	1.145,4	545,7	305,8

Tabela 5.3: Estatísticas sobre a simulação do subsistema *consenso-GR* com amostra de 20 experimentos

Estatísticas	consenso-GRA		
	carga_sem_picos	carga_picos	carga_previsão
n_amostra	20	20	20
n_total	4.062	6.529	1.469
latência de decisão (ms)	222,3	355,1	203,1

Tabela 5.4: Estatísticas sobre a simulação do subsistema *consenso-GRA* com amostra de 20 experimentos

OurGrid é constituído de laboratórios distribuídos por todo o mundo que doam seus recursos computacionais ociosos em troca de acesso aos recursos ociosos de outros laboratórios, quando necessário. Por questões de escalabilidade, o OurGrid é baseado em uma rede par-a-par onde cada laboratório de pesquisa é considerado um par da rede (também denominada comunidade OurGrid)[8]. O OurGrid foi projetado para rodar aplicações paralelas cujas tarefas sejam independentes (aplicações BoT - *Bag-of-Tasks*). Tais aplicações são simples, porém, podem ser usadas em diversas áreas, tal como simulações. Uma simulação no OurGrid é descrita através de um trabalho (*job*), que é composto por uma coleção de tarefas independentes (*tasks*). A geração dos *jobs* OurGrid para as simulações envolvendo os subsistemas *consenso-GR* e *consenso-GRA* é descrita no Algoritmo 6. Introduziu-se a idéia de tarefa agrupada (*grouped_task*) que equivale à composição de 6 *tasks* OurGrid, uma para cada combinação subsistema de consenso/tipo de carga de rede. Assim, um *job* OurGrid é composto de 86 *grouped_tasks*, sendo necessários um total de 100 *jobs* para cobrir o conjunto de 8.600 experimentos previstos.

5.3.4 Resultados

As simulações no OurGrid foram executadas entre 01-04-2006 e 23-05-2006 e constaram de mais experimentos do que o planejado. Ao todo foram executados 64.714 experimentos de

[8]Visite o site do sistema OurGrid para maiores informações sobre o mesmo (`http://www.ourgrid.org`).

Algoritmo 6 Geração dos jobs OurGrid

```
% para cada job:
for all 1 ≤ exp ≤ quantidade_experimentos do
  lista_processos = gera, aleatoriamente, lista ordenada de processos
  trace_aplicação = escolhe, aleatoriamente, trace de carga da aplicação
  num_trace = escolhe, aleatoriamente, número do trace a ser usado entre 1..81
  for all t ∈ {carga com picos, carga sem picos, carga de previsão} do
    % TASK-GR corresponde a um experimento com 100 execuções do subsistema-GR
    % usando lista_processos, trace_aplicação e o trace de tipo t e número num_trace
    gera TASK-GR
    adiciona TASK-GR a GROUPED_TASK
    % TASK-GRA corresponde a um experimento com 100 execuções do
    % subsistema-GRA usando trace_aplicação e o
    % trace de tipo t e número num_trace
    gera TASK-GRA
    adiciona TASK-GRA a GROUPED_TASK
  end for
  adiciona GROUPED_TASK a JOB
end for
```

100 consensos para as 6 combinações de subsistema de consenso/tipo de carga de rede, isto equivale à 64.714 *grouped_ tasks* em 750 *jobs*. A razão para tais valores refere-se à variação dos cenários que se tornou mais evidente ao longo da execução das simulações, exigindo a realização de um maior número de experimentos para garantir o nível de confiança especificado (95% com erro máximo de 5%). Os resultados como também o total de experimentos realizados são resumidos nas Tabelas 5.5 e 5.6.

Estatísticas	consenso-GR		
	carga_sem_picos	carga_picos	carga_previsão
n_amostra	64.714	64.714	64.714
n_total	58.091	46.108	43.738
latência de decisão (ms)	830,3 ± 39,3	944,3 ± 39,9	1.017,6 ± 41,8

Tabela 5.5: Estatísticas finais sobre a simulação do subsistema *consenso-GR*

Estatísticas	consenso-GRA		
	carga_sem_picos	carga_picos	carga_previsão
n_amostra	64.714	64.714	64.714
n_total	8.616	9.053	6.874
latência de decisão (ms)	221,0 ± 4,0	264,4 ± 5,0	242,9 ± 4,0

Tabela 5.6: Estatísticas finais sobre a simulação do subsistema *consenso-GRA*

Analisando o desempenho de cada subsistema de consenso usando a carga de rede sem picos (carga_picos), carga de rede com picos (carga_sem_picos) e carga de rede de previsão (carga_previsão), observa-se que o subsistema *consenso-GRA* apresenta desempenhos semelhantes para carga_sem_picos e carga_previsão; uma diferença de 9% (ver Figura 5.7)[9]. Estas cargas de rede são parecidas, pois, o modelo de previsão aplicado à carga_picos tinha o objetivo de suavizar os picos tornando o conjunto de dados mais uniforme. Como o mecanismo de adaptação usado no subsistema *consenso-GRA* toma decisões a partir das condições de carga do ambiente de execução, os cenários de carga_sem_picos e carga_previsão devem gerar ações de adaptação semelhantes. Isto demonstra a robustez do mecanismo de adaptação. Por outro lado, os cenários com carga_picos resultaram em desempenhos um pouco piores, diferença em relação à carga_sem_picos de $16,4\%$.

[9]Os valores percentuais calculados neste capítulo representam os ganhos ou perdas de desempenho, comparando os valores máximo e mínimo envolvidos no cálculo. Então, os ganhos/perdas percentuais entre os valores *max* e *min* é dado por: % = (max - min) * 100 / max.

Note que a existência de picos na carga de rede pode indicar variações abruptas no comportamento de máquinas lentas (rápidas) tornando-as mais lentas ou mais rápidas. Tais variações podem diminuir a eficiência do mecanismo de adaptação utilizado, causando decisões erradas por parte do mesmo (mudanças ineficientes na lista ordenada de processos). Em contra-partida, o desempenho do subsistema *consenso-GR* também difere para os três tipos de carga de rede, porém, as diferenças não estão associadas às características de cada carga, mas à configuração de lista ordenada de processos utilizada.

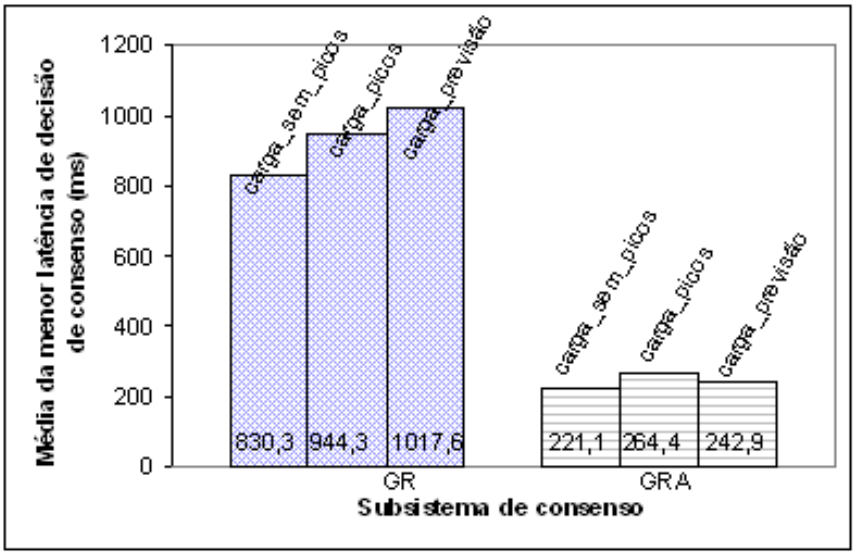


Figura 5.7: Impacto do tipo da carga de rede sobre o desempenho dos subsistemas *consenso-GR* e *consenso-GRA*

A Figura 5.8 ilustra o desempenho dos subsistemas *consenso-GR* e *consenso-GRA* considerando os respectivos intervalos de confiança para a média da menor latência de consenso. Nesse caso, o desempenho do subsistema adaptativo é superior àquele apresentado pelo seu correspondente não-adaptativo nos três tipos de carga de rede, alcançando ganhos de até $76,1\%$ (carga_previsão). Estes ganhos de desempenho significativos comprovam a eficiência do mecanismo de adaptação aplicado. É importante ressaltar que, o intervalo de confiança para os resultados do *consenso-GRA* é muito pequeno, dessa forma, o erro sobre a média amostral da menor latência de consenso para este protocolo é muito pequeno.

5.4 Experimentos na rede real

Na seção anterior foram descritos os resultados dos experimentos de simulação para os subsistemas *consenso-GR* e *consenso-GRA*. Os resultados demonstram a eficiência do sub-

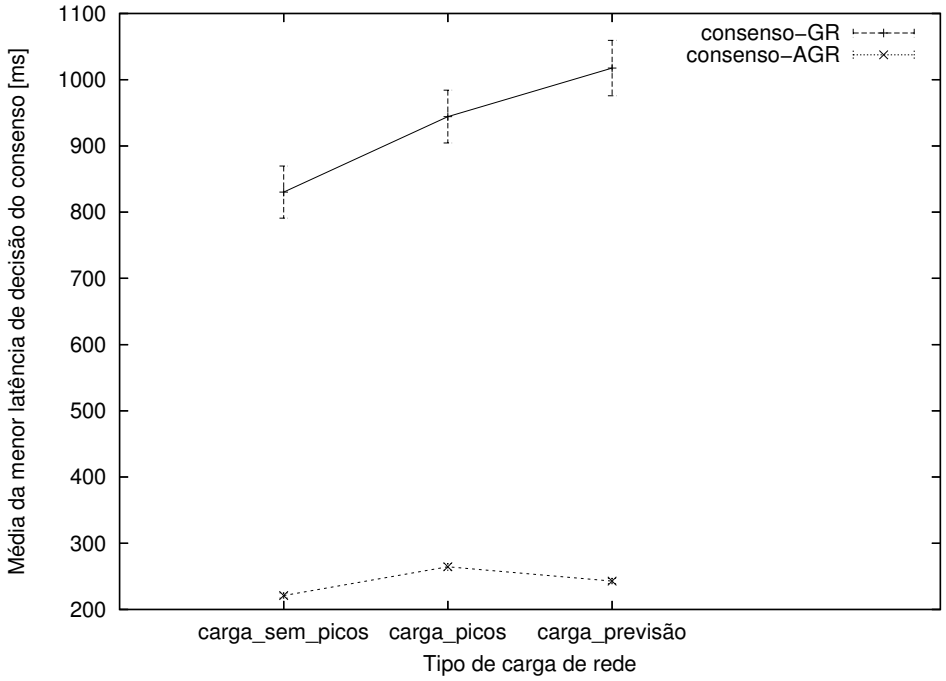


Figura 5.8: Comparação do desempenho dos subsistemas *consenso-GR* e *consenso-GRA* usando simulação

sistema adaptativo. No sentido de validar os experimentos de simulação, confirmando, assim, a eficiência do subsistema adaptativo, foram realizados experimentos na rede real (ou medições).

A estratégia para conduzir os experimentos na rede real consistiu em reproduzir um cenário de simulação específico e comparar os resultados obtidos no ambiente real com aqueles no ambiente simulado. Como já é conhecido, um cenário de simulação consta de valores para o trace de carga da aplicação, trace de carga de rede, tipo de carga de rede, e lista ordenada de processos. Na rede real o trace de carga de rede foi usado para identificar o grupo de 5 máquinas onde as medições seriam realizadas, além disso, o tipo de carga de rede foi descartado pois tal parâmetro não é configurável.

O ambiente de execução das medições sobre o desempenho dos subsistemas de consenso foi o PlanetLab (PATERSON et al., 2003; SPRING et al., 2006). Nesse caso, utilizou-se o arcabouço NEKO, configurado para execuções na rede real. Nas próximas seções enfatiza-se alguns aspectos de configuração do NEKO na rede real, além da definição dos cenários de medição e resultados obtidos com as medições no PlanetLab.

5.4.1 Utilizando o arcabouço NEKO na rede real

Como foi discutido anteriormente (ver Seção 5.1), a inicialização de uma aplicação NEKO difere para simulações ou medições. Outra diferença refere-se à configuração da rede de comunicação. Existem modelos para rede simulada e rede real, neste último caso, o arcabouço provê implementações de redes TCP/IP e UDP/IP. Nos experimentos realizados com os subsistemas *consenso-GR* e *consenso-GRA* utilizou-se a implementação de rede TCP/IP.

Uma característica específica de execuções na rede real usando o NEKO corresponde à necessidade de sincronização de relógios. O sistema de sincronização de relógios é implementado por dois objetos denominados de *ClockSynchronizerSlave* e *ClockSynchronizer*, os quais são associados a uma aplicação NEKO como mostra a Figura 5.9 para os subsistemas de consenso em estudo. O objeto *ClockSynchronizer* está associado ao processo p_0 (processo mestre), enquanto o objeto *ClockSynchronizerSlave* está associado a cada um dos demais processos (processos escravos).

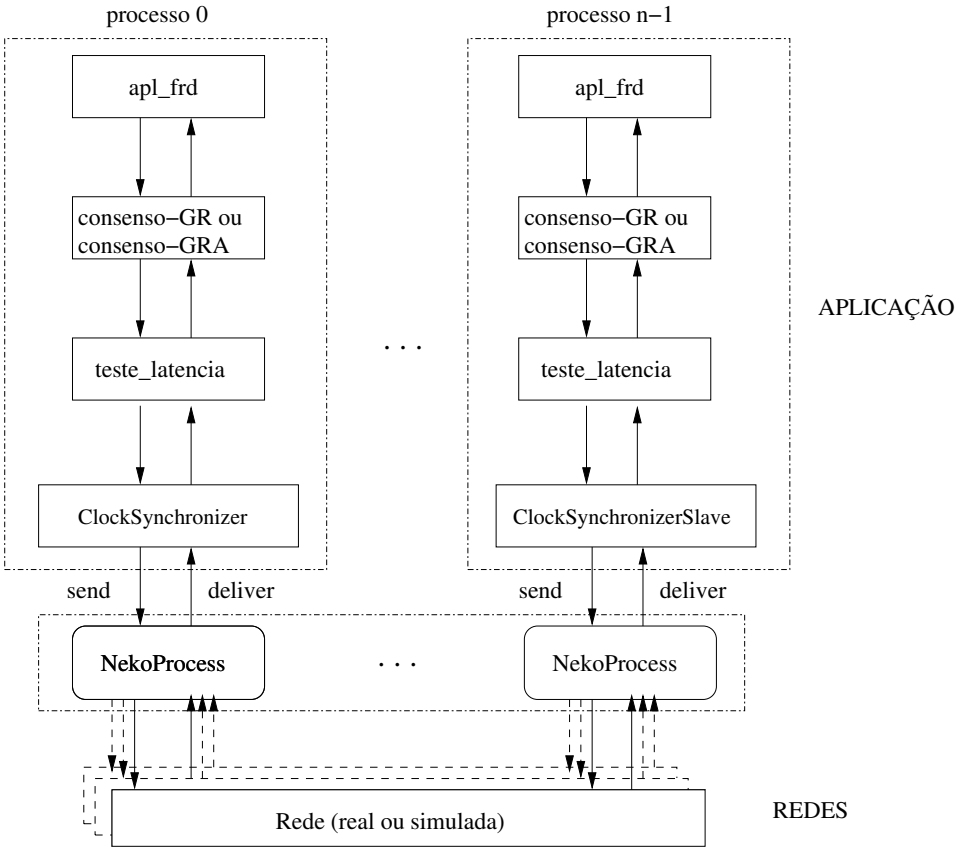


Figura 5.9: Arquitetura dos subsistemas *consenso-GR* e *consenso-GRA* para execuções na rede real usando o NEKO

O mecanismo de sincronização de relógios consta de rodadas as quais são repetidas por

um número finito de vezes ($total\_rodadas$), suficiente para que todos os processos participantes do sistema sejam considerados sincronizados em relação ao processo mestre. Isto ocorre quando a diferença entre os valores dos relógios físicos do processo mestre e cada um dos demais processos satisfizer uma determinada precisão ($prec$). O Algoritmo 7 descreve o mecanismo de sincronização de relógios, onde, em cada rodada o processo mestre troca mensagens com os processos escravos para partilhar o valor de seus respectivos relógios. Então, calcula-se a diferença entre os relógios, ajustando o relógio dos processos escravos. Se o ajuste de relógio para um processo p_i satisfizer uma determinada precisão por um número mínimo de rodadas ($min\_rodadas$), p_i está sincronizado em relação ao processo mestre p_0. Note que, se o número máximo de rodadas for extrapolado o algoritmo será finalizado e não se garante a sincronização entre todos os processos. Portanto, é preciso escolher o valor de $total\_rodadas$ para evitar tal inconsistência.

Nos experimentos envolvendo os subsistemas de consenso no PlanetLab, o mecanismo de sincronização de relógios foi configurado com precisão de $30ms$, além disso, a configuração do número de rodadas do mecanismo de sincronização segue valores padrões (sugeridos no código-fonte do NEKO): $min\_rodadas = 3$ e $total\_rodadas = 30$. O objetivo de usar tal mecanismo foi garantir, com certa precisão, que os participantes dos subsistemas iniciassem as execuções dos consensos ao mesmo tempo. Nesse caso, a sincronização de relógios era a primeira atividade realizada após a inicialização da aplicação NEKO.

5.4.2 Definição dos cenários de medição

O cenário de simulação escolhido para configurar os experimentos no PlanetLab é composto de um trace de carga de rede do tipo carga com picos. Este tipo de carga descreve mais fielmente o comportamento do ambiente de execução real, entretanto, o tipo de carga serve apenas para os experimentos de simulação. O trace de carga de rede engloba um grupo de 5 máquinas distribuídas em diferentes localidades, a saber: Canadá (ca), Japão (jp), Estados Unidos (usa) e Suécia (se), sendo uma máquina em cada localidade, com exceção dos Estados Unidos com duas máquinas. O grupo de 5 máquinas é identificado pela configuração **ca-jp-usa1-se-usa2**, tal que, se $\Pi_{e1} = \{p_1, p_2, p_3, p_4, p_5\}$ é o conjunto de participantes de cada subsistema de consenso, então p_1 executa em **ca**, p_2 em **jp** e assim, sucessivamente, nessa ordem. O nome completo das máquinas é apresentado a seguir.

```
planet2.calgary.canet4.nodes.planet-lab.org (ca)
planetlab2.koganei.wide.ad.jp (jp)
planetlab2.cs.virginia.edu (usa1)
```

Algoritmo 7 Sincronização de relógios no NEKO

```
% Para o processo mestre:
while ainda existem processos não sincronizados e total_rodadas não extrapolou do
  envia mensagem SINC para processos escravos
  registra tempo de envio T_ENVIO
  waituntil receber resposta para mensagem SINC de todos os processos escravos
  registra tempo de recebimento de cada resposta T_RESPOSTA
  requisita de todos os processos escravos informação sobre relógio local T_LOCAL
  waituntil receber T_LOCAL de todos os processos escravos
  for all processo escravo ainda não sincronizado do
    calcula diferença entre os relógios do processo mestre e do processo escravo
    DIF = (T_ENVIO + T_RESPOSTA) / 2 - T_LOCAL
    envia ajuste do relógio DIF para o processo escravo
    if DIF ≤ prec para min_rodadas then
      processo escravo está sincronizado
    end if
  end for
end while

% Para os processos escravos:
loop
  % PASSO 1
  recebe mensagem SINC do processo mestre
  registra tempo de recebimento T_LOCAL
  envia resposta para mensagem SINC
  % PASSO 2
  envia T_LOCAL para processo mestre
  % PASSO 3
  recebe do processo mestre ajuste para relógio local DIF
  ajusta tempo inicial do experimento
  T_INICIAL = T_INICIAL - DIF
end loop
```

```
planetlab-1.it.uu.se (se)
planetlab11.millennium.berkeley.edu (usa2)
```

A lista ordenada de processos tem valor $[p_4, p_3, p_1, p_2, p_5]$, sendo usada para configurar o subsistema de consenso não-adaptativo. No sentido de permitir uma análise mais completa sobre o impacto da lista ordenada de processos sobre o desempenho do consenso, foram geradas outras configurações para a mesma a partir da original, como mostra a Figura 5.10. Tais configurações diferem da original pelo fato do primeiro elemento de cada lista de processos executar em uma máquina diferente. Este elemento representa o coordenador da primeira rodada de cada execução do consenso e, possivelmente, o primeiro processo a alcançar um valor consensual já que se considera cenários sem falhas ou falsas suspeições. Assim, totalizaram-se 5 valores para a lista ordenada de processos.

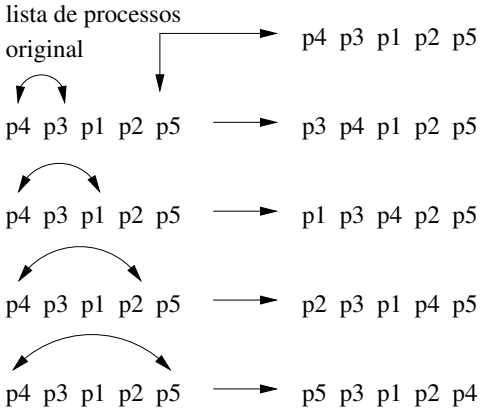


Figura 5.10: Composição das listas ordenadas de processos para os experimentos de medição e simulação (parte II)

O último parâmetro do cenário de simulação escolhido é o trace de carga da aplicação cujo valor foi utilizado nas medições tal como nas simulações.

Portanto, foram definidos 5 cenários para as medições no PlanetLab, os quais utilizam o mesmo valor para o grupo de 5 máquinas e trace de carga da aplicação, porém variam a lista ordenada de processos. Cada experimento consistiu de uma execução de 100 consensos do subsistema *consenso-GR* e do subsistema *consenso-GRA*, considerando um dos cenários especificados. Como foi enfatizado na simulação, as configurações de lista ordenada de processos são parâmetros de entrada para o subsistema não-adaptativo, entretanto, o subsistema adaptativo também pode usar listas de processos diferentes em cada experimento o que dependerá da situação de carga do ambiente de execução. Note que, nas simulações os subsistemas foram avaliados considerando os mesmos traces de carga de rede. Esta condição não pode ser garantida nos experimentos na rede real. No sentido de proporcionar

cargas de rede parecidas para as execuções de cada subsistema, optou-se por configurar uma nova "fatia" de rede no PlanetLab e executar os subsistemas de consenso em paralelo. Nesse caso, a primeira "fatia" de rede (fatia01) executou o subsistema *consenso-GR* e a segunda (fatia02), o subsistema *consenso-GRA*. Ambas as "fatias" de rede são compostas pelas mesmas máquinas, sendo utilizado o mesmo grupo de 5 máquinas para as execuções dos dois subsistemas.

5.4.3 Resultados

Os experimentos no PlanetLab foram repetidos a fim de alcançar 95% de nível de confiança na média da menor latência de consenso (métrica de desempenho) com erro máximo de 5% da média; as repetições de alguns experimentos ocorreram de maneira intercalada ao longo de 1 a 3 dias. Isto é necessário para cobrir as possíveis variações de carga na rede que constitui a variável aleatória destes experimentos. Por outro lado, a fim de comparar resultados de simulação e medição fez-se necessário executar simulações dos subsistemas de consenso usando as mesmas configurações de lista ordenada de processos do experimento na rede real. É importante lembrar que os experimentos de simulação usam o mesmo cenário dos experimentos na rede real, onde o trace de carga de rede é do tipo carga com picos. Além disso, as simulações e medições do subsistema *consenso-GRA* usaram a lista ordenada de processos $[p_1, p_3, p_4, p_2, p_5]$ como valor inicial. Os resultados obtidos são descritos na Tabela 5.7, onde estão representados cada lista ordenada de processos.

	consenso-GR (ms)		*consenso-GRA (ms)*		ganhos/perdas (%)	
	sim	real	sim	real	sim	real
$[p_1, p_3, p_4, p_2, p_5]$	144,6	241,6	197,4	305,3	-26,8	-20,9
$[p_2, p_3, p_1, p_4, p_5]$	356,8	376,8	197,4	324,0	44,7	14,0
$[p_3, p_4, p_1, p_2, p_5]$	209,0	347,0	197,4	367,4	5,6	-5,6
$[p_4, p_3, p_1, p_2, p_5]$	489,0	463,6	197,4	344,0	59,6	25,8
$[p_5, p_3, p_1, p_2, p_4]$	183,4	371,4	197,4	334,5	-7,1	9,9

Tabela 5.7: Resultados preliminares de simulação e medição para os subsistemas *consenso-GR* e *consenso-GRA*

Tanto nas medições quanto nas simulações, para cada execução do subsistema não-adaptativo, usando uma das 5 configurações de lista ordenada de processos, foi executado em paralelo o subsistema adaptativo. Note que os resultados de simulação do subsistema *consenso-GRA* são idênticos em todas as execuções, pois os cenários de simulação são os mesmos. Porém, os resultados de medição são diferentes devido às variações na carga

do ambiente de execução (PlanetLab). Nesse caso, o fato de executar em paralelo os subsistemas *consenso-GR* e *consenso-GRA* permitiu observar o comportamento do subsistema adaptativo em diferentes momentos de carga do ambiente de execução. Analisando o desempenho de cada subsistema de consenso em tal contexto observa-se a estabilidade do subsistema adaptativo ao longo do experimento, apresentando índices de desempenho entre $305,3ms$ e $367,4ms$ (diferença de $16,9\%$). O mesmo não ocorre com o subsistema não-adaptativo cujo desempenho varia entre $241,6ms$ e $463,6ms$ (diferença de $47,9\%$) para diferentes listas ordenadas de processos.

Note que, nas medições no PlanetLab, para cada execução do subsistema não-adaptativo usando uma das 5 configurações de lista ordenada de processos, foi executado em paralelo o subsistema adaptativo. Isto permitiu observar o comportamento do subsistema adaptativo em diferentes momentos de carga do ambiente de execução. Analisando o desempenho de cada subsistema de consenso em tal contexto observa-se a estabilidade do subsistema adaptativo ao longo do experimento, apresentando índices de desempenho entre $305,3ms$ e $367,4ms$ (diferença de $16,9\%$). O mesmo não ocorre com o subsistema não-adaptativo cujo desempenho varia entre $241,6ms$ e $463,6ms$ (diferença de $47,9\%$) para diferentes listas ordenadas de processos.

A comparação dos resultados de desempenho dos subsistemas de consenso usando simulação e medição se baseia na lentidão do primeiro coordenador em uma execução (ou o primeiro elemento da lista ordenada de processos). Lentidão é uma característica associada aos atrasos fim-a-fim na transmissão de mensagens pela rede; uma máquina será considerada lenta dentro de um grupo se mensagens trocadas entre ela e demais máquinas do grupo sofrerem atrasos altos. De acordo com os resultados descritos na Tabela 5.7, o subsistema não-adaptativo apresentou pior desempenho quando o primeiro coordenador executava em **jp** e **se**. Isto sugere que as máquinas **jp** e **se** sejam lentas, enquanto as demais **ca**, **usa1** e **usa2** sejam rápidas. Sendo assim, o desempenho dos subsistemas de consenso é entendido da seguinte forma.

No caso do primeiro coordenador ser rápido (**ca**, **usa1** e **usa2**), a justificativa para o desempenho do subsistema adaptativo ter índices piores pode ser porque os ganhos pelo uso de adaptação são inferiores ao custo do mecanismo de adaptação, assim, como o subsistema não-adaptativo está livre destes custos, seu desempenho tende a ser melhor. Este custo de adaptação é percebido tanto nas simulações quanto nas medições e refere-se à sobrecarga de execução do mecanismo de adaptação (serviço de ordenação de processos): processamento, comunicação e escolhas erradas. Quando o primeiro coordenador é lento (**jp** e **se**), o subsistema adaptativo apresenta melhor desempenho do que o não-adaptativo tanto nas

simulações quanto nas medições, apesar dos ganhos percentuais terem grandezas diferentes. Uma justificativa para os menores ganhos do subsistema adaptativo seria a existência de sobrecarga de escalonamento nas medições, o que pode influenciar ambos os subsistemas de formas e proporções diferentes. Além disso, as possíveis escolhas erradas por parte do mecanismo de adaptação podem ser supervalorizadas nas medições devido à sobrecarga de escalonamento. Note que existem duas configurações onde os resultados de simulação são diferentes daqueles obtidos nas medições (respectivamente, ganho e perda ou, vice-versa). Isto ocorreu para o primeiro coordenador executando em **usa1** e **usa2**, entretanto, considerando o erro máximo tolerado de 5% da média (para mais ou para menos), os subsistemas *consenso-GR* e *consenso-GRA* teriam o mesmo desempenho tanto pela simulação quanto pelas medições. A justificativa refere-se à sobreposição dos intervalos de confiança para a média, nesse caso, dois valores $v_1 \pm e_1$ e $v_2 \pm e_2$ serão considerados diferentes se $(v_1 - e_1) > (v_2 + e_2)$ (para $v_1 > v_2$) ou $(v_1 + e_1) < (v_2 - e_2)$ (para $v_1 < v_2$), *i.e.*, os intervalos de confiança não se interceptam. Por exemplo, os resultados para a simulação usando a lista ordenada de processos $[p_3, p_4, p_1, p_2, p_5]$, primeiro coordenador executando em **usa1**, foram $219,0 \pm 10,45$ (*consenso-GR*, v_1) e $197,4 \pm 9,87$ (*consenso-GRA*, v_2); observa-se que $198,55 < 207,27$ (ou seja, $(v_1 - e_1) < (v_2 + e_2)$), portanto, os intervalos de confiança se interceptam.

As conclusões apontadas acima se baseiam em algumas hipóteses as quais precisam ser melhor investigadas. No contexto das medições, deve-se investigar o impacto da sobrecarga de escalonamento para o subsistema *consenso-GRA*. Nesse caso, **1) se os dois subsistemas tivessem a mesma sobrecarga de escalonamento, o desempenho do subsistema adaptativo na rede real poderia ser melhor?** Por outro lado, os experimentos de simulação foram configurados com um trace de carga de rede constituído de dados de latência coletados em fev-2006 enquanto os primeiros experimentos na rede real ocorreram em jun-2006; **2) usar traces de carga de rede mais atuais tornaria os resultados de simulação e medição mais próximos?** Ainda nesse sentido, os experimentos de simulação envolvem a execução de 100 consensos cuja duração não chega a 1 minuto, em contra-partida, o experimento na rede real pode durar dias, isto significa que existe uma maior possibilidade das execuções na rede real experimentarem variações de carga na rede do que as execuções na rede simulada. Então, **3) realizar mais experimentos de simulação contribuiria para tornar cenários de simulação e de medição mais compatíveis quanto à carga de rede?** Em relação a simulações e medições argumenta-se sobre o impacto negativo do custo do mecanismo de adaptação; **4) é possível que o custo de adaptação sobreponha-se ao seu benefício?** Para

responder às questões apresentadas, foram executados novos experimentos, os quais serão relatados a seguir.

Investigando as simulações

Nesta seção deseja-se investigar a sobrecarga associada ao subsistema de consenso adaptativo, além do impacto de usar diferentes traces de carga de rede para a configuração de máquinas **ca-jp-usa1-se-usa2**, e a melhor forma de usar cada trace de carga de rede a fim de aproximar a configuração da rede simulada daquela que caracteriza a rede real.

A execução do subsistema *consenso-GRA* gera uma sobrecarga maior do que a do subsistema *consenso-GR*. Isto é devido ao custo do respectivo mecanismo de adaptação, o qual se refere à sobrecarga de execução introduzida pelo mesmo e, indiretamente, pelo oráculo de latência necessário para realizar ações de adaptação. A sobrecarga de execução envolve 1) o número de mensagens extras trocadas pelo oráculo de latência para calcular *rtts*, 2) o processamento realizado pelo mecanismo para disparar ações de adaptação e, 3) a imprecisão das ações de adaptação. Dentre os fatores enumerados, 1) e 3) influenciam tanto os resultados de simulação quanto de medição no PlanetLab. Por questões de simplicidade, optou-se por avaliar o impacto destes fatores em um ambiente controlado (simulações), apesar de acreditar-se que as conclusões sejam aplicadas, também, às execuções em um ambiente real.

Para avaliar os fatores 1) e 3), realizou-se um experimento usando um subsistema de consenso, denominado de *consenso-GRA**, que combina características dos subsistemas *consenso-GR* e *consenso-GRA*. Por um lado, o subsistema *consenso-GRA** apresenta a mesma sobrecarga de execução do *consenso-GRA*, mas o respectivo serviço de ordenação de processos não executa ações de adaptação e, assim, está livre das escolhas erradas. Nesse caso, o serviço comporta-se como aquele associado ao *consenso-GR*, fornecendo uma lista ordenada de processos definida a priori. O *consenso-GRA** é parametrizado com a mesma lista ordenada de processos usada para a execução do *consenso-GR*. O experimento foi configurado com o mesmo cenário descrito na seção anterior.

A Tabela 5.8 descreve o desempenho dos 3 subsistemas de consenso em questão. A coluna com ganhos/perdas de desempenho está associada aos subsistemas *consenso-GR* e *consenso-GRA**. Note que tais subsistemas usam a mesma lista ordenada de processos e diferem, apenas, pela carga extra de processamento e comunicação introduzida no *consenso-GRA**. Assim, a diferença de desempenho entre os subsistemas *consenso-GRA** e *consenso-GR* equivale à sobrecarga de comunicação do subsistema adaptativo. Esta diferença variou de -3% a $-21,3\%$. No caso do primeiro coordenador executar na máquina **usa2**, observou-

se que as perdas do subsistema *consenso-GRA* sobre o *consenso-GR* $(7,1\%)$ são inferiores às perdas do subsistema *consenso-GRA** sobre o *consenso-GR* $(9,8\%)$, isto significa que os possíveis benefícios oriundos do mecanismo de adaptação, para este cenário, são inferiores à sobrecarga introduzida pelo mesmo.

	consenso-GR(ms)	*consenso-GRA*(ms)*	*consenso-GRA(ms)*	perdas(%)
$[p_1, p_3, p_4, p_2, p_5]$	144,6	149,1	197,4	3,0
$[p_2, p_3, p_1, p_4, p_5]$	356,8	396,3	197,4	10,0
$[p_3, p_4, p_1, p_2, p_5]$	209,0	216,1	197,4	3,3
$[p_4, p_3, p_1, p_2, p_5]$	489,0	621,5	197,4	21,3
$[p_5, p_3, p_1, p_2, p_4]$	183,4	203,4	197,4	9,8

Tabela 5.8: Avaliação do custo de adaptação associado ao subsistema *consenso-GRA* usando simulações

Outra observação importante refere-se ao impacto das escolhas erradas por parte do mecanismo de adaptação, que pode ser percebido nos resultados para o cenário onde o primeiro coordenador executa na máquina **ca**. Nesse caso, o desempenho do *consenso-GRA** e *consenso-GRA* é de, respectivamente, $149,1ms$ e $197,4ms$, ou seja, o subsistema adaptativo tem desempenho inferior àquele apresentado pela outra versão adaptativa que, de fato, não realiza ações de adaptação. Dessa forma, o uso de adaptação em tal cenário teve impacto negativo e a razão está associada à ocorrência de escolhas erradas por parte do mecanismo de adaptação. A melhor escolha do mecanismo de adaptação seria a lista $[p_1, p_3, p_4, p_2, p_5]$ (como indicado pelo desempenho do *consenso-GRA**, cujo mecanismo de adaptação não comete escolhas erradas), entretanto, outros valores também foram escolhidos ao longo da execução do *consenso-GRA*, degradando seu desempenho. Este exemplo demonstra a possibilidade de ocorrência de escolhas erradas e seu impacto negativo sobre o subsistema adaptativo.

Analisando o uso de diferentes traces de carga de rede e a melhor forma de aproveitar os dados de latência que compõem os mesmos observa-se o seguinte. Os cenários de simulação considerados envolvem 5 configurações de lista ordenada de processos, 1 trace de carga de rede e 1 trace de carga da aplicação. Para justificar a repetição dos experimentos de simulação e, assim, aproveitar melhor os traces de carga de rede, faz-se necessário a existência de uma variável aleatória nos respectivos cenários. Nesse caso, pode-se usar o índice a partir do qual os arquivos de latência do trace de carga de rede são lidos. Até então, a leitura dos arquivos é feita a partir da primeira entrada do mesmo, ou seja, índice zero $(ind = 0)$. Sabendo que cada arquivo de latência possui 14.400 entradas pode-se usar uma

distribuição uniforme parametrizada com os valores 0 (mínimo) e 14.400 (máximo) para escolher o valor da variável *ind* em cada experimento de 100 consensos. Os experimentos seriam repetidos até alcançar nível de confiança de 95% e erro máximo de 5% da média.

Novos traces de carga de rede foram gerados para a configuração de máquinas do PlanetLab em estudo, **ca-jp-usa1-se-usa2**, com o objetivo de aproximar a carga da rede simulada daquela experimentada na rede real. Sendo assim, além de considerar o trace original gerado em fev-2006 foram incorporados traces de jul-2006 e ago-2006. A geração dos novos traces seguiu a mesma estratégia para coleta dos dados de latência do trace original (ver Seção 5.3.2). Ao todo, acumulou-se 5 traces com carga de jul-2006, 1 trace com carga de ago-2006 e 1 trace com carga de fev-2006.

Dessa forma, os experimentos de simulação foram reexecutados usando os traces de carga de rede citados acima, como também, a estratégia de índice aleatório para leitura dos arquivos de dados de latência que compõem os traces. Nesse caso, cada experimento envolvendo um determinado trace de carga de rede foi repetido a fim de alcançar 95% de nível de confiança sobre a média, com erro máximo de 5%, usando diferentes valores para o índice aleatório; em cada repetição, o mesmo índice serviu de parâmetro de entrada para as execuções de ambos os subsistemas de consenso. A Tabela 5.9 resume os resultados obtidos, em termos de ganhos/perdas de desempenho do subsistema *consenso-GRA* sobre o subsistema *consenso-GR*. Os resultados detalhados são mostrados nas Tabelas 5.10, 5.11 e 5.12.

O uso do índice aleatório tornou a carga de rede das simulações mais heterogênea e dinâmica o que impactou o desempenho dos subsistemas de consenso em alguns cenários. Este foi o caso do subsistema *consenso-GR* para a lista de processos $[p_1, p_3, p_4, p_2, p_5]$, primeiro coordenador rodando na máquina **ca**, houve degradação no desempenho do subsistema em relação à execução com $ind = 0$ (ver Tabela 5.7), favorecendo o desempenho do subsistema *consenso-GRA* com ganhos de $6,7\%$ (se considerar o erro de 5%, o desempenho dos subsistemas foi equivalente). Analisando o impacto de usar novos traces de carga de rede (mais atuais em relação ao período dos experimentos na rede real), os resultados também demonstram variações na carga do ambiente de execução. Para todas as cargas o desempenho do *consenso-GRA* foi superior àquele apresentado pelo seu correspondente não-adaptativo quando o primeiro coordenador executou nas máquinas **jp** e **se**, com ganhos entre $52,2\%$ e 74% de acordo com o trace de rede utilizado. No caso do primeiro coordenador na máquina **ca**, o desempenho do subsistema adaptativo foi inferior, com perdas entre $3,6\%$ e $6,7\%$, porém, abaixo do erro máximo de 5% o que torna o desempenho dos subsistemas equivalentes. O mesmo ocorre para o caso do primeiro coordenador na

máquina **usa2** (ganhos entre $4,8\%$ e $6,7\%$; perdas entre $0,8\%$ e $5,4\%$). Quando o primeiro coordenador executa na máquina **usa1** o desempenho dos subsistemas são equivalentes (ganhos entre $4,3\%$ e $9,3\%$; perdas de $0,5\%$), exceto para uma situação onde o subsistema adaptativo obteve melhor desempenho (ganhos de 16%); isto demonstra uma variação na carga da máquina **usa1**, onde o uso de adaptação permitiu mascarar os respectivos efeitos. Portanto, o uso de traces de carga de rede gerou resultados de simulação diferentes, apesar, dos padrões se repetirem no geral.

	fev-2006	18-07-2006	21-07-2006	28-07-2006	31-07-2006	23-08-2006
$[p_1, p_3, p_4, p_2, p_5]$	6,7%	-6,1%	-4,0%	-3,6%	-4,9%	-6,5%
$[p_2, p_3, p_1, p_4, p_5]$	48,0%	68,0%	74,0%	63,0%	52,2%	61,0%
$[p_3, p_4, p_1, p_2, p_5]$	5,2%	4,3%	-0,5%	8,4%	16,0%	9,3%
$[p_4, p_3, p_1, p_2, p_5]$	50,3%	66,0%	70,0%	67,0%	60,9%	65,0%
$[p_5, p_3, p_1, p_2, p_4]$	-10,6%	6,7%	-0,8%	4,8%	-5,4%	6,7%

Tabela 5.9: Resultados de desempenho dos subsistemas *consenso-GR* e *consenso-GRA*, em termos percentuais (ganhos/perdas), usando diferentes traces de carga em simulações

	fev-2006	
	GR (ms)	*GRA (ms)*
$[p_1, p_3, p_4, p_2, p_5]$	224,0	208,0
$[p_2, p_3, p_1, p_4, p_5]$	401,3	208,0
$[p_3, p_4, p_1, p_2, p_5]$	219,9	208,0
$[p_4, p_3, p_1, p_2, p_5]$	419,1	208,0
$[p_5, p_3, p_1, p_2, p_4]$	186,3	208,0

Tabela 5.10: Resultados de simulação usando trace de fev-2006

	18-07-2006		21-07-2006		28-07-2006		31-07-2006	
	GR (ms)	*GRA (ms)*	*GR*	*GRA*	*GR*	*GRA*	*GR*	*GRA*
$[p_1, p_3, p_4, p_2, p_5]$	118,0	125,6	148,4	154,6	113,9	118,1	142,5	149,8
$[p_2, p_3, p_1, p_4, p_5]$	404,5	125,6	603,3	154,6	320,5	118,1	313,6	149,8
$[p_3, p_4, p_1, p_2, p_5]$	131,3	125,6	153,8	154,6	128,9	118,1	178,4	149,8
$[p_4, p_3, p_1, p_2, p_5]$	374,5	125,6	520,6	154,6	359,7	118,1	383,2	149,8
$[p_5, p_3, p_1, p_2, p_4]$	134,6	125,6	153,3	154,6	124,0	118,1	141,7	149,8

Tabela 5.11: Resultados de simulação usando trace de jul-2006

	23-08-2006	
	GR (ms)	*GRA (ms)*
$[p_1, p_3, p_4, p_2, p_5]$	115,8	123,9
$[p_2, p_3, p_1, p_4, p_5]$	318,1	123,9
$[p_3, p_4, p_1, p_2, p_5]$	136,6	123,9
$[p_4, p_3, p_1, p_2, p_5]$	360,8	123,9
$[p_5, p_3, p_1, p_2, p_4]$	132,6	123,9

Tabela 5.12: Resultados de simulação usando trace de ago-2006

Investigando as medições

Sobrecarga de escalonamento está associada ao atraso introduzido no escalonamento das atividades de cada subsistema de consenso durante a execução dos mesmos no PlanetLab. Se várias atividades podem ocorrer em um determinado instante de tempo t, o término de suas execuções será em $t + t'$ e não é possível garantir a ordem na qual tais atividades ocorrem. O mesmo não é válido para as simulações, pois o tempo de simulação não é contínuo e a execução de atividades é instantânea; o relógio do simulador progride até o próximo instante de tempo onde existem novas atividades para executar. Dessa forma, o desempenho dos subsistemas de consenso nas medições pode ser degradado devido à sobrecarga de escalonamento. Além disso, uma mesma atividade programada para o tempo t pode sofrer um atraso de escalonamento maior na execução do subsistema *consenso-GR* do que no subsistema *consenso-GRA* e vice-versa.

Uma forma de amenizar o impacto da sobrecarga de escalonamento é usando uma carga extra de processamento ou comunicação, de modo que, a sobrecarga de escalonamento seja diluída. Nesse caso, a carga extra deve ser suficientemente grande e igual para as execuções dos dois subsistemas de consenso. Através desta estratégia é também possível diluir o custo da adaptação (em particular, o efeito da sobrecarga de processamento). Portanto, a estratégia sugerida contribui para tornar os cenários executados por cada subsistema mais uniforme, permitindo uma comparação mais justa entre os desempenhos dos subsistemas adaptativo e não-adaptativo. Nas medições no PlanetLab a carga extra foi representada pelo tamanho da mensagem. Até então, as mensagens trocadas durante a execução dos subsistemas de consenso tinha tamanho 0bytes, a idéia foi aumentar o tamanho da mensagem gerando sobrecarga de comunicação.

O tamanho da mensagem foi configurado para 32Kbytes e usado em novos experimentos no PlanetLab, os quais ocorreram no período de 15-07-2006 a 18-08-2006. Além do tamanho da mensagem, os cenários de medição cobertos pelos experimentos incluíram os valores

especificados na Seção 5.4.3 para as execuções de jun-2006. Os resultados obtidos são resumidos na Tabela 5.13.

	consenso-GR (ms)		*consenso-GRA (ms)*		ganhos/perdas (%)	
	0bytes	32Kbytes	0bytes	32Kbytes	0bytes	32Kbytes
$[p_1, p_3, p_4, p_2, p_5]$	241,6	1.028,5	305,3	1.931,1	-20,9	-46,7
$[p_2, p_3, p_1, p_4, p_5]$	376,8	3.137,1	324,0	1.836,3	14,0	41,5
$[p_3, p_4, p_1, p_2, p_5]$	347,0	1.636,2	367,4	1.983,6	-5,6	-17,5
$[p_4, p_3, p_1, p_2, p_5]$	463,6	11.136,0	344,0	2.123,7	25,8	80,9
$[p_5, p_3, p_1, p_2, p_4]$	371,4	2.262,7	334,5	2.606,0	9,9	-13,2

Tabela 5.13: Resultados de desempenho dos subsistemas *consenso-GR* e *consenso-GRA*, usando medição, em cenários com tamanho de mensagens de 0bytes e 32Kbytes

Comparando os resultados descritos na Tabela 5.13 com aqueles obtidos nas execuções com mensagens de tamanho 0bytes, observa-se o impacto da sobrecarga introduzida pelo aumento do tamanho da mensagem. Considerando os ganhos/perdas em cada cenário tem-se que as diferenças se acentuam tanto nas perdas quanto nos ganhos, exceto para uma configuração de lista ordenada de processos (primeiro coordenador na máquina **usa2**). Isto ocorre porque a sobrecarga introduzida mascara a carga extra inerente ao subsistema adaptativo tornando ambos os subsistemas de consenso equivalentes quanto à carga, assim os ganhos proporcionados pelo mecanismo de adaptação ficam mais aparentes no caso do primeiro coordenador, da execução do subsistema não-adaptativo, executar nas máquinas **jp** e **se** (coordenadores lentos). Por outro lado, o impacto das escolhas erradas do mecanismo de adaptação é maior devido ao aumento da sobrecarga, desfavorecendo o desempenho do subsistema adaptativo quando o primeiro coordenador executa nas máquinas **ca**, **usa1** e **usa2** (coordenadores rápidos). Note que, os ganhos de desempenho observados para o subsistema adaptativo em relação ao subsistema não-adaptativo com primeiro coordenador na máquina **usa2** transformaram-se em perdas com o aumento do tamanho da mensagem. A justificativa para este fato pode estar associada ao impacto potencializado das escolhas erradas, devido à sobrecarga introduzida ou, ainda, a alguma variação de carga do ambiente de execução (picos); observe como o desempenho do subsistema adaptativo diminuiu neste cenário.

Durante a execução dos experimentos acima descritos foi observada uma mudança temporária de carga na rede, que se prolongou de 29-07-2006 a 14-08-2006. Uma das máquinas (**usa1**), considerada rápida até então, tornou-se lenta. Os resultados obtidos antes e durante o período de lentidão são descritos na Tabela 5.14; tais resultados satisfazem um nível

de confiança de 85% e 9% de erro sobre a média. Os resultados de antes do período de lentidão indicam que o subsistema adaptativo tem desempenho inferior ao seu correspondente não-adaptativo (configurado com a lista de processos $[p_3, p_4, p_1, p_2, p_5]$), alcançando perdas de $23,3\%$. Por outro lado, durante o período de lentidão, os resultados se invertem apontando ganhos de desempenho para o adaptativo de $87,7\%$. Este cenário demonstra a robustez do subsistema adaptativo em relação às variações de carga do ambiente de execução, além disso, contribui para a afirmação de que o PlanetLab possui carga heterogênea e dinâmica.

	28-07-2006 (antes)	29-07-2006 (durante)
consenso-GR (ms)	886,2	180.373,2
consenso-GRA (ms)	1.155,1	22.106,0
ganhos/perdas	-23,3	87,7

Tabela 5.14: Resultados de desempenho dos subsistemas *consenso-GR* e *consenso-GRA*, antes e durante o período de lentidão ocorrido ao longo dos experimentos de medição com mensagens de 32Kbytes

Períodos de lentidão podem ocorrer com qualquer máquina Internet, sendo causados por inúmeros fatores. Em particular, enumeram-se problemas de comunicação de/para a referida máquina e variações na carga de processamento da máquina. Para investigar estes fatores realizaram-se alguns testes, onde o primeiro deles foi feito com a ferramenta PING em um ambiente Linux. O objetivo do teste foi identificar possíveis gargalos de comunicação. O teste do PING consistiu do envio de 30 pacotes de 32Kbytes da máquina **usa1** para cada uma das demais máquinas da configuração **ca-jp-usa1-se-usa2**. O teste foi repetido durante (29-07-2006) e depois (14-08-2006) do período de lentidão. Na primeira etapa foi observado 98% de perdas de pacotes para todos os pares **usa1** $\rightarrow$ <**destino**>, enquanto na segunda etapa as perdas variaram de 0% a 6%. Possíveis razões para as altas perdas no período de lentidão incluem problemas nos canais através dos quais é dada a comunicação de/para **usa1** ou, diminuição de tráfego "ping" em **usa1**. As informações providas pelo PING são insuficientes para precisar problemas nos canais de comunicação de/para **usa1**; isto requer ferramentas mais robustas, tais como TRACEROUTE e IPERF. Por outro lado, diminuição do tráfego "ping" em uma máquina pode estar associada a restrições temporárias no limite de largura de banda de/para esta máquina, o que também requer outros tipos de teste para ser investigado.

O segundo teste envolveu a ferramenta TRACEROUTE. Foram coletados dados usando mensagens de 1Kbyte entre os pares de máquinas **ca-jp**, **usa1-jp**, **se-jp** e **usa2-jp**. Tem-se

dois objetivos para este teste: 1) analisar as rotas da máquina **usa1** até sair do domínio .edu ao qual a máquina pertence; 2) analisar as rotas da máquina **usa1** entrando em um novo domínio, nesse caso, .jp. Para alcançar o primeiro objetivo poderia ter sido escolhido qualquer destino (**ca**, **jp**, **se** ou **usa2**), haja vista que as rotas serão as mesmas. Em relação ao segundo objetivo, deseja-se observar se as rotas a partir da máquina **usa1** são as mesmas seguidas pelas outras máquinas. Problemas de comunicação podem ser percebidos através de atrasos altos nas transmissões de mensagens entre uma rota e outra de **usa1-jp**, como também, perdas de mensagens. O teste foi repetido durante (08-08-2006) e depois (17-08-2006) do período de lentidão. Os resultados não demonstraram anomalias na comunicação entre **usa1** e **jp** o que favorece a conclusão de que a rede estava livre de problemas.

Uma outra justificativa para o período de lentidão corresponde à sobrecarga na máquina **usa1**. Esta sobrecarga pode ser gerada pelo aumento na concorrência por recursos da mesma, tais como cpu e memória, devido ao aumento de aplicações rodando na máquina (grande número de "fatias" ativas). Sobrecarga de cpu causa atrasos mais longos entre acessos à cpu por parte de uma aplicação, enquanto a sobrecarga de memória causa maior frequência de *swapping* de dados[10]. Como consequência, o tempo de execução das aplicações pode estender-se. Ainda nesse contexto, destaca-se também a influência da taxa de transmissão/recepção de **usa1**, *i.e.*, largura de banda disponível. De acordo com a política de atribuição de largura de banda usada no PlanetLab, cada "fatia" associada a uma máquina possui um limite diário de acesso à banda desta máquina, entretanto, tal limite pode ser reduzido para garantir que todas as "fatias" naquela máquina tenham acesso "justo" à banda (fenômeno denominado de *traffic shapping*). A redução no limite de banda diário tem maior possibilidade de ocorrer quando o número de "fatias" associadas a uma máquina é alto. Os fatores até então enumerados (sobrecarga de cpu, sobrecarga de memória e largura de banda) podem ser investigados através de ferramentas específicas que monitoram, continuamente, o estado das máquinas do PlanetLab e as atividades desenvolvidas sobre as mesmas. Em particular tem-se as ferramentas comon/cotop (`http://codeen.cs.princeton.edu/`). Infelizmente, os dados coletados por estas ferramentas para o período de lentidão não incluem referências à máquina **usa1**. Os administradores das ferramentas foram consultados e alegaram que, por alguma razão, a execução das ferramentas naquela máquina parou e não foi possível ocorrer uma recuperação automática; a ferramenta voltou a ficar operacional em **usa1** a partir de 28-08-2006.

Os testes descritos acima não permitem identificar as causas exatas para o período

[10] O swapping é realizado pelo sistema operacional e consiste na cópia de dados de um processo da memória secundária (disco rígido) para a memória principal (RAM) e vice-versa.

de lentidão ocorrido durante os experimentos no PlanetLab. Isto porque existem vários fatores possíveis, os quais, muitas vezes, não podem ser investigados ou fogem ao controle de um usuário comum. Entretanto, os resultados dos testes apontam para alguns fatores importantes que contribuem para entender melhor o período de lentidão.

Comparando simulações e medições

A comparação entre os resultados das simulações e medições, apresentados anteriormente, gerou alguns questionamentos, que deveriam ser discutidos a fim de confirmar as conclusões parciais alcançadas sobre o desempenho dos subsistemas adaptativo e não-adaptativo, são eles:

1. se os dois subsistemas tivessem a mesma sobrecarga de escalonamento, o desempenho do subsistema adaptativo na rede real poderia ser melhor?
2. usar traces de carga de rede mais atuais tornaria a comparação dos resultados de simulação e medição mais próxima?
3. realizar mais experimentos de simulação contribuiria para tornar cenários de simulação e de medição mais compatíveis quanto à carga de rede?
4. é possível que o custo de adaptação sobreponha-se ao seu benefício?

O questionamento 4 foi discutido no contexto das simulações, revelando que, de fato, o custo de adaptação pode ultrapassar os seus benefícios, principalmente, devido à possibilidade de escolhas erradas por parte do mecanismo de adaptação. Além disso, os efeitos das escolhas erradas são mais evidentes quando comparado o desempenho do subsistema adaptativo com aquele do subsistema não-adaptativo configurado com o primeiro coordenador sendo rápido. O questionamento 1 estava relacionado ao impacto da sobrecarga de escalonamento, nesse caso, aumentou-se o tamanho da mensagem nas execuções dos subsistemas de consenso a fim de equilibrar a sobrecarga associada a cada um deles. Comparando os resultados das medições com tamanho de mensagem de 0bytes e 32Kbytes observou-se que os ganhos do subsistema adaptativo foram bem melhores, entretanto, as perdas se acentuaram. A razão é que, o aumento da sobrecarga do subsistema adaptativo também causa o aumento do impacto das escolhas erradas por parte do mecanismo de adaptação. Por fim, os questionamentos 2 e 3 resultaram em novas simulações usando traces de carga de rede mais atuais e o índice aleatório. Este último refere-se a uma variável aleatória cujo valor indica a posição a partir da qual os arquivos de latência que compõem os traces serão lidos;

atribuindo diferentes valores ao índice aleatório foi possível aproveitar melhor os traces de rede no sentido da abrangência e, por conseguinte, tornar o ambiente simulado mais heterogêneo e dinâmico.

As discussões acerca dos questionamentos supracitados geraram cenários de simulação e medição, que foram testados através de novos experimentos. As Tabelas 5.15 e 5.16 descrevem os resultados obtidos para as medições com mensagens de 32Kbytes e simulações com cargas mais atuais e uso do índice aleatório. Neste último caso, os valores representam a mediana[11] de todos os valores obtidos para as cargas de jul-2006 (18-07-2006, 21-07-2006, 28-07-2006 e 31-07-2006).

	consenso-GR (ms)	*consenso-GRA (ms)*	ganhos/perdas (%)
$[p_1, p_3, p_4, p_2, p_5]$	130,3 ± 5,2	137,7 ± 5,0	-5,4
$[p_2, p_3, p_1, p_4, p_5]$	362,5 ± 8,4	137,7 ± 5,0	62,0
$[p_3, p_4, p_1, p_2, p_5]$	142,6 ± 3,8	137,7 ± 5,0	3,4
$[p_4, p_3, p_1, p_2, p_5]$	378,9 ± 12,0	137,7 ± 5,0	63,7
$[p_5, p_3, p_1, p_2, p_4]$	138,2 ± 3,2	137,7 ± 5,0	0,7

Tabela 5.15: Resultados finais de simulação para os subsistemas *consenso-GR* e *consenso-GRA*

	consenso-GR (ms)	*consenso-GRA (ms)*	ganhos/perdas (%)
$[p_1, p_3, p_4, p_2, p_5]$	1.028,5 ± 37,2	1.931,1 ± 93,1	-46,7
$[p_2, p_3, p_1, p_4, p_5]$	3.137,1 ± 106,5	1.836,3 ± 89,0	41,5
$[p_3, p_4, p_1, p_2, p_5]$	1.636,2 ± 80,1	1.983,6 ± 87,5	-17,5
$[p_4, p_3, p_1, p_2, p_5]$	11.136,0 ± 199,3	2.123,7 ± 104,4	80,9
$[p_5, p_3, p_1, p_2, p_4]$	2.262,7 ± 109,8	2.606,0 ± 122,0	-13,2

Tabela 5.16: Resultados finais de medição para os subsistemas *consenso-GR* e *consenso-GRA*

Note que os resultados de simulação e medição são diferentes para os casos onde o primeiro coordenador executa nas máquinas **ca**, **usa1** e **usa2** (coordenadores rápidos). Isto é causado pelo impacto da sobrecarga introduzidas pelo aumento no tamanho da mensagem combinado com as escolhas erradas pelo subsistema adaptativo, dessa forma,

[11] Nesse caso, a mediana foi calculada considerando uma amostra de 4 valores, assim, esta corresponde à média entre os segundo e terceiro menores valores da amostra. Sejam tais valores representados por $v_2 \pm e_2$ e $v_3 \pm e_3$, a mediana med é dada por $med = (v_2 + v_3)/2 \pm e_{med}$. Onde, $e_{med} = \lceil |med - [(v_2 + e_2) + (v_3 + e_3)/2]| \rceil$ ou $e_{med} = \lceil |med - [(v_2 - e_2) + (v_3 - e_3)/2]| \rceil$.

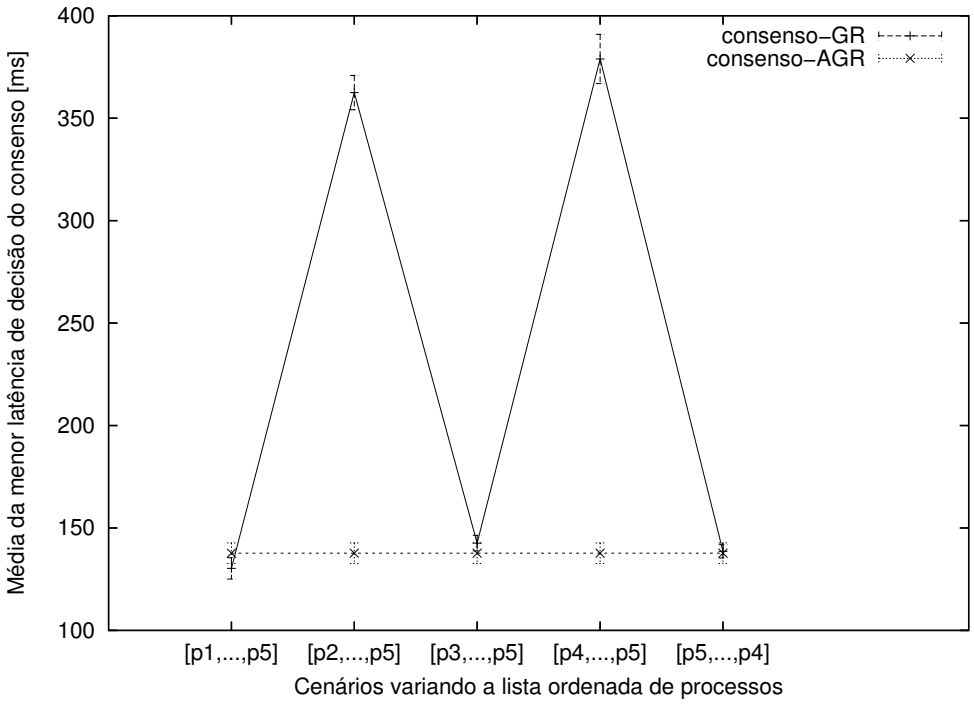
as perdas da simulação tornam-se maiores nas medições e os pequenos ganhos convertem-se em perdas. Em outras palavras, as diferenças entre o subsistema adaptativo e seu correspondente não-adaptativo são acentuadas, desfavorecendo o uso de adaptação. Por outro lado, quando o primeiro coordenador executa nas máquinas **jp** e **se** (coordenadores lentos) a diferença entre os ganhos do subsistema adaptativo nas medições e simulações foi menor em relação aos resultados preliminares descritos na Tabela 5.7. Isto porque, o aumento do tamanho da mensagem combinado ao fato do coordenador ser lento degrada substancialmente o desempenho do subsistema não-adaptativo nas medições, amenizando o impacto das escolhas erradas do mecanismo de adaptação; quanto mais lento o primeiro coordenador do subsistema não-adaptativo (nesse caso, a máquina **se**) maiores os ganhos do subsistema adaptativo, podendo estes ganhos superarem aqueles obtidos nas simulações.

É ainda importante enfatizar que, considerando o desempenho como a média de todos os cenários executados, observa-se a superioridade do subsistema adaptativo tanto nas simulações quanto nas medições com ganhos de desempenho equivalentes, de $40,3\%$ e $45,4\%$, respectivamente. Uma razão para este fato refere-se à maior robustez do subsistema adaptativo quanto às variações de carga no ambiente de execução (ver Figura 5.11). O desempenho do subsistema não-adaptativo sofre variações de $1.028,5ms$ a $11.136,0ms$, nas medições e, $130,3ms$ a $378,9ms$, nas simulações, para as configurações de lista ordenada de processos (as quais representam diferentes cenários de carga para o subsistema não-adaptativo). Ou seja, variações de $90,8\%$ e $65,6\%$, respectivamente, enquanto, as variações de desempenho do subsistema adaptativo (nas medições) é bem menor, alcançando índices de $29,5\%$.

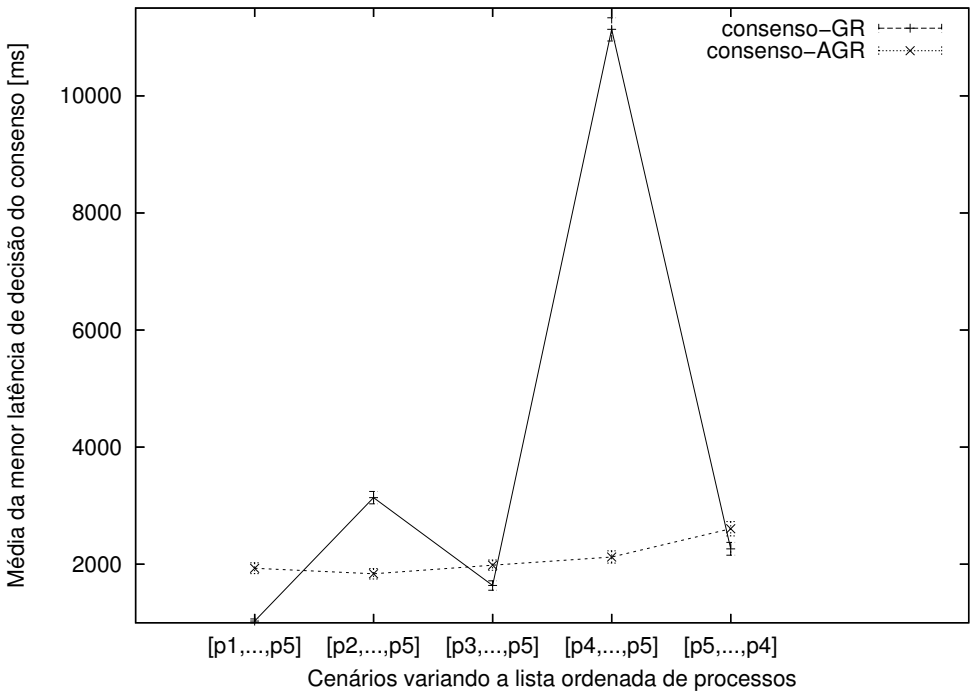
5.5 Conclusões parciais

Neste capítulo foi descrita a avaliação de desempenho do subsistema *consenso-GRA* cujo objetivo foi medir a eficiência de uma solução para ordenação de processos adaptativa baseada em oráculos de latência. Para tal, realizou-se um estudo comparativo entre os desempenhos do subsistema adaptativo e de sua versão não-adaptativa, denominada de subsistema *consenso-GR*. O estudo foi feito por meio de experimentos de simulação e de medição em um ambiente real, usando como métrica a média do menor tempo de decisão do consenso em várias execuções dos respectivos protocolos de consenso. Os resultados demonstraram a superioridade do subsistema *consenso-GRA*, o qual pode alcançar ganhos de desempenho de até $76,1\%$.

Nos experimentos foram considerados cenários sem falhas ou falsas suspeições, como



(a) Resultados de simulação



(b) Resultados de medição

Figura 5.11: Compação do desempenho dos subsistemas *consenso-GR* e *consenso-GRA* usando simulação e medição

também, execuções sequenciais do consenso onde o intervalo entre as execuções é determinado pela aplicação cliente do consenso (sistema de arquivos distribuído para tolerância a intrusões na Internet). A carga da aplicação foi gerada através de uma distribuição de probabilidade exponencial parametrizada com valores de uma aplicação real (BODNARCHUK; BUNT, 1991). Todos os experimentos envolveram 5 processos participantes em cada subsistema de consenso.

Os experimentos de simulação executaram cenários realistas, cada qual representando a combinação de valores para as variáveis: lista ordenada de processos, carga da rede, carga da aplicação. Em particular, a carga de rede foi obtida a partir de experimentos em um ambiente real (PlanetLab (PATERSON et al., 2003; SPRING et al., 2006)), envolvendo vários grupos de 5 máquinas escolhidas aleatoriamente a partir de um conjunto de 160 máquinas. Os valores da lista ordenada de processos resultam da permutação do conjunto de 160 máquinas distribuídas geograficamente. Esta estratégia teve o objetivo de tornar a rede simulada um ambiente sujeito à carga heterogênea e dinâmica. Ainda nesse contexto, a carga de rede original foi filtrada através da aplicação de modelos de previsão e mecanismos para remoção de picos, originando três tipos de carga de rede: carga com picos, carga sem picos e carga de previsão. Foi alcançado 95% de nível de confiança nos resultados dos experimentos com erro de 5% sobre a média. Os resultados favorecem o subsistema adaptativo que permite ganhos de desempenho de até $76,1\%$.

Os experimentos de medição foram executados no PlanetLab e tiveram o objetivo de validar os resultados de simulação. Dessa forma, escolheu-se um cenário de simulação que foi modificado no sentido de incluir variações da lista ordenada de processos. Este cenário foi reproduzido no PlanetLab e repetiu-se a simulação. As variações na lista ordenada de processos resultaram em configurações com o primeiro elemento da lista sendo rápido ou lento. Os primeiros resultados demonstraram compatibilidade entre simulação e medição, *i.e.*, ganhos/perdas nas simulações implicaram, normalmente, em ganhos/perdas nas medições, apesar dos índices de desempenho serem diferentes. Em alguns casos houve diferenças, mas com sobreposição dos respectivos intervalos de confiança, indicando equivalência entre os resultados.

No sentido de melhor comparar os resultados de simulação e medição, investigou-se alguns fatores que podem influenciar um ou outro método de avaliação e, portanto, devem ser considerados. Dois fatores importantes referem-se à sobrecarga de escalonamento e à sobrecarga de execução (carga de processamento, comunicação e escolhas erradas) do subsistema adaptativo. A sobrecarga de escalonamento caracteriza as execuções em um ambiente real, podendo influenciar no desempenho do *consenso-GR* ou *consenso-GRA* de

maneiras e proporções diferentes. Por outro lado, a sobrecarga de execução do subsistema adaptativo caracteriza tanto as simulações quanto as medições, entretanto, o impacto sobre as medições pode ser maior devido ao efeito da sobrecarga de escalonamento. Na tentativa de amenizar o impacto da sobrecarga de escalonamento nas medições optou-se por utilizar mensagens maiores (32Kbytes), nesse caso, os resultados de medição demonstraram o aumento nos ganhos de desempenho com o subsistema adaptativo, para os casos onde o primeiro coordenador era lento, e aumento das perdas, para os casos onde o primeiro coordenador era rápido. Na primeira situação, a sobrecarga de execução do subsistema não-adaptativo com coordenador lento é muito alta e usar adaptação é vantajoso, porém, se o coordenador for rápido (segunda situação), a sobrecarga de execução do subsistema adaptativo supera os benefícios causados pelo mesmo, em particular, o impacto das escolhas erradas é muito alto. Comparando com os resultados de simulação, as diferenças se acentuaram para os cenários onde o primeiro coordenador era rápido.

De modo geral, conclui-se que o uso de adaptação no subsistema *consenso-GRA* pode trazer ganhos de desempenho em alguns cenários, como também, perdas para outros. A questão principal a ser observada refere-se à robustez no desempenho do subsistema adaptativo; para um mesmo conjunto de cenários o desempenho do subsistema adaptativo oscila em um intervalo pequeno enquanto o de seu correspondente não-adaptativo oscila em um intervalo muito maior. Como o ambiente de execução considerado caracteriza-se pelas variações de carga, a possibilidade de ocorrer cenários eficientes e ineficientes durante a execução do subsistema não-adaptativo aumenta. Isto foi comprovado nos experimentos de medição realizados para o tamanho da mensagem de 32Kbytes quando uma máquina, considerada rápida, tornou-se lenta por alguns dias. Nesse caso, houve degradação no desempenho do subsistema não-adaptativo e ganhos de desempenho para o subsistema adaptativo (antes inferior ao desempenho do subsistema não-adaptativo).

Note que a eficiência de um mecanismo de adaptação baseado em oráculos de latência está fortemente relacionada com a eficiência do oráculo de latência utilizado. O estudo de caso apresentado neste capítulo, considera uma implementação simples de oráculo de latência, caracterizada pelo uso de serviços do ambiente de execução para diminuir custos, assim como, o uso de uma estratégia de geração das estimativas de latência fundamentada no último *rtt* calculado. Por um lado, é preciso avaliar a relação custo-benefício de não usar serviços do ambiente de execução, podendo ajustar suas configurações de forma independente. Além disso, pode-se utilizar uma estratégia mais robusta para geração de estimativas de latência, por exemplo, modelos de previsão (NUNES; JANSCH-PôRTO, 2002; SAMPAIO et al., 2005).

Capítulo 6

Conclusões e trabalhos futuros

6.1 Resultados obtidos e contribuições

O objeto de estudo desta tese foram os protocolos distribuídos simétricos pelo texto, onde os processos se comunicam por troca de mensagens. Tais protocolos podem ser configurados quanto aos papéis assumidos por seus participantes ao longo da execução dos mesmos, além disso, a configuração dos papéis determina o desempenho do protocolo. Nesse caso, uma configuração reflete sobre a dinâmica da troca de mensagens de/para cada participante (padrão de comunicação entre os participantes), sendo fortemente influenciado pelas variações na carga do ambiente. Isto significa que, quando o protocolo executa em ambientes sujeitos à carga heterogênea e dinâmica, uma configuração eficiente pode tornar-se ineficiente ao longo do tempo, degradando seu desempenho. Portanto, protocolos simétricos pelo texto podem sofrer perdas de desempenho quando usam uma configuração definida a priori para execuções em ambientes sujeitos à carga heterogênea e dinâmica.

Considerando as limitações de desempenho de protocolos simétricos pelo texto em ambientes sujeitos à carga heterogênea e dinâmica, foi proposto o uso de adaptação por valor baseada em oráculos de latência. Adaptação foi aplicada no sentido de tornar a configuração dos papéis, em protocolos simétricos pelo texto, sensível às variações de carga do ambiente de execução. Dessa forma, a hipótese de tese investigada diz respeito aos ganhos de desempenho provenientes da solução adaptativa proposta. A validação da hipótese de tese constou de estudos analítico e experimental (simulação e medição em um ambiente real) sobre a eficiência de uma solução adaptativa baseada em oráculos de latência. No primeiro caso, avaliou-se a eficiência da solução em si, enquanto no segundo caso, avaliou-se a eficiência de um protocolo simétrico pelo texto equipado com a solução adaptativa proposta. A validação comprovou a hipótese de tese como mostram os resultados resumidos a

seguir.

A primeira etapa da validação da hipótese de tese (Capítulo 3) fundamentou-se na definição formal de um oráculo de latência. Diferentemente dos trabalhos já publicados na literatura sobre o assunto, a definição proposta considera o fator tempo, nesse caso, o oráculo fornece estimativas de latência sobre quaisquer dois processos do sistema em qualquer instante de tempo. Em particular, o oráculo definido possui duas características: 1) o oráculo provê estimativas sobre latências no presente, passado e futuro e, 2) o oráculo provê informação global. Foi realizado um estudo analítico envolvendo um protocolo de consenso equipado com uma solução adaptativa baseada no oráculo de latência definido. Nesse caso, a solução resolve o problema da ordenação de processos, o qual representa a necessidade de uma lista ordenada de processos para atribuição de papéis em protocolos simétricos pelo texto. O protocolo de consenso usa uma lista ordenada de processos para escolher a identidade dos processos que desempenham o papel de coordenador em cada rodada do protocolo; adaptação serve para escolher configurações eficientes para a lista ordenada de processos de acordo com as informações providas pelo oráculo de latência. O estudo analítico consistiu da elaboração de um modelo matemático que permite representar o desempenho do protocolo de consenso adaptativo, para todas as configurações de lista ordenada de processos, em função da definição do oráculo de latência. Usando tal modelo demonstra-se que, o mecanismo de adaptação poderá escolher a lista ordenada de processos através da qual o protocolo alcançará melhores índices de desempenho. Em outras palavras, demonstra-se a eficiência de um mecanismo de adaptação baseado em oráculos de latência.

A segunda etapa da validação da hipótese de tese fundamentou-se nos aspectos de engenharia de uma solução adaptativa baseada em oráculos de latência. Sendo assim, elaborou-se um estudo de caso, incluindo descrição, projeto e implementação (Capítulo 4) e, realizou-se a avaliação de desempenho do mesmo por meio de experimentos de simulação e medição (Capítulo 5).

O estudo de caso envolveu, mais uma vez, um protocolo de consenso simétrico pelo texto equipado com uma solução adaptativa para ordenação de processos. Utilizou-se a mesma solução adaptativa, baseada em oráculos de latência, empregada no estudo analítico, porém, considerando um novo protocolo de consenso e objetivos diferentes. O protocolo de consenso em questão foi encapsulado em um subsistema de consenso associado a uma aplicação para tolerância a intrusões na Internet, cujo papel restringiu-se a determinar a frequência de requisições de consenso. O projeto e implementação do subsistema de consenso adaptativo seguiu uma abordagem modular funcional, obedecendo ao princípio da separação de conceitos. Nesse processo, um fato importante referiu-se à redefinição

do oráculo de latência a ser implementado. A razão para tal é que não seria possível implementar o oráculo definido anteriormente nas condições desejadas (solução distribuída em um ambiente assíncrono).

De acordo com a nova definição, o oráculo de latência provê estimativas de latência globais, considerando tempo lógico ao invés de tempo físico. Outra preocupação observada referiu-se à análise da relação custo-benefício para a tomada de decisões de projeto e implementação da solução adaptativa, onde custo refere-se à sobrecarga de comunicação. O maior impacto no custo da solução adaptativa refere-se à implementação do oráculo de latência, mais especificamente, às necessidades de 1) monitoramento dos processos do sistema para calcular latências, de 2) difusão de informações de latência coletadas localmente para garantir abrangência e, de 3) acordo para garantir informação de latência global. Nos dois primeiros casos, uma alternativa para amenizar os custos é usar mensagens de aplicações do ambiente de execução, evitando mensagens extras. No terceiro caso, a sugestão é usar o próprio subsistema de consenso para obter acordo sobre as informações do oráculo de latência. A desvantagem de tais estratégias é que a implementação do oráculo de latência fica dependendo da implementação e configuração de outros componentes, os quais podem não ser os mais eficientes.

A avaliação de desempenho do estudo de caso consistiu de um conjunto de passos, desde a definição dos objetivos a serem alcançados até a análise estatística dos resultados obtidos. Nesse processo, priorizou-se o uso de métricas e cenários realistas, como também, métodos de avaliação apropriados. Optou-se por uma avaliação comparativa entre os desempenhos do subsistema de consenso adaptativo e seu correspondente não-adaptativo, usando experimentos de simulação e medição. A métrica de desempenho é dada em função do tempo de terminação do consenso e os cenários foram construídos a partir de impressões de um ambiente real (*e.g.* carga da rede simulada) ou valores fundamentados na estatística (combinações da lista ordenada de processos nas execuções do subsistema não-adaptativo) ou valores justificados na literatura (*e.g.* carga da aplicação). Os resultados da avaliação de desempenho demonstraram a eficiência do mecanismo de adaptação utilizado. Nas simulações, o subsistema adaptativo obteve ganhos de desempenho de até $76,1\%$; os resultados de simulação representam uma média de desempenho para os 64.714 cenários executados.

No sentido de validar os resultados de simulação, definiu-se 5 cenários (a partir daqueles anteriormente simulados) para os quais executou-se medições no PlanetLab, comparando os resultados com aqueles obtidos em novas simulações nos mesmos cenários. Estes cenários foram ajustados ao longo dos experimentos a fim de permitir uma comparação mais justa entre os resultados de simulação e medição. Analisando os resultados finais para cada ce-

nário, observou-se duas situações: 1) o subsistema adaptativo pode apresentar desempenho pior do que o subsistema não-adaptativo e, 2) os índices de desempenho (perdas/ganhos) para o subsistema adaptativo podem ser diferentes na simulação e medição. A primeira situação ocorre tanto nas simulações quanto medições, em cenários onde o valor da lista ordenada de processos é muito eficiente para o subsistema não-adaptativo, sendo assim, o custo da adaptação (em particular, o custo de escolhas erradas) supera os seus benefícios. A segunda situação é consequência da sobrecarga extra inerente a um ambiente real, o que potencializa ganhos/perdas nas medições. É importante ressaltar que, considerando uma média de desempenho para todos os cenários, os resultados de simulação e medição foram semelhantes (respectivamente, ganhos de $40,3\%$ e $45,4\%$ para o subsistema adaptativo). E ainda, existem cenários para os quais o desempenho do subsistema adaptativo é inferior ao desempenho do subsistema não-adaptativo e vice-versa, porém, aquele é mais estável para todos os cenários. Além disso, cenários eficientes podem tornar-se ineficientes em ambientes sujeito à carga heterogênea e dinâmica, nesse caso, o subsistema não-adaptativo sofrerá degradação de desempenho, enquanto o subsistema adaptativo escolherá uma nova configuração, sendo assim, mais robusto às variações na carga do ambiente. Isto foi observado nos experimentos de medição, quando, para um dado cenário, o desempenho do subsistema adaptativo alcançou índices de $-23,3\%$ e $87,7\%$ sobre o desempenho do seu correspondente não-adaptativo.

Os resultados alcançados nesta tese representam contribuições relevantes para a comunidade científica, as quais são apontadas a seguir:

- uso de adaptação em protocolos simétricos pelo texto através de uma solução baseada em oráculos de latência: os trabalhos disponíveis na literatura sobre adaptação em protocolos distribuídos não consideram oráculos de latência de forma explícita, ou seja, explorando questões de engenharia e impacto da implementação dos mesmos sobre as soluções adaptativas que os utilizam;

- investigação de questões práticas e teóricas de oráculos de latência em soluções adaptativas;

- sob o ponto de vista teórico, formalização do oráculo de latência e aplicação deste em um estudo analítico que permitiu demonstrar a eficiência de uma solução adaptativa baseada em oráculos de latência;

- sob o ponto de vista prático, projeto e implementação de uma solução adaptativa baseada em oráculos de latência de acordo com princípios da engenharia de software:

a transição entre a especificação (teoria) e implementação (prática) da solução revelou a necessidade de simplificar a especificação para garantir a implementação da solução da maneira desejada, além disso, o uso de princípios da engenharia ajuda a lidar com a complexidade introduzida pela adaptação;

- ainda sob o ponto de vista prático, avaliação de desempenho experimental da solução adaptativa baseada em oráculos de latência: houve a preocupação de usar modelos e métricas capazes de capturar, o mais fielmente possível, o comportamento real dos sistemas e, assim, favorecer uma análise realista. Nesse sentido, enfatiza-se, ainda, o uso de cenários e configurações adequadas;

- tanto na avaliação de desempenho analítica quanto na experimental foi usado como estudo de caso o consenso, que é um bloco básico para construir sistemas distribuídos tolerante a faltas: as vantagens de um sistema baseado em consenso incluem a separação de conceitos e modularização, por outro lado, a desvantagem é que sistemas dessa natureza geram muita sobrecarga. Nesse caso, melhorar o desempenho do consenso significa diminuir tal desvantagem. Vale ainda ressaltar a ausência de trabalhos na literatura sobre uso de adaptação em protocolos de consenso para fins de desempenho.

Comprovando a relevância do trabalho na comunidade científica, parte dos resultados obtidos nesta tese foram publicados em veículos especializados (SAMPAIO et al., 2003; SAMPAIO; BRASILEIRO; MOREIRA, 2004; SAMPAIO; BRASILEIRO, 2005; SAMPAIO et al., 2005). Os resultados finais serão resumidos em um novo artigo a ser submetido para uma revista com foco na área de estudo.

6.2 Trabalhos futuros

Sugere-se três possibilidades de trabalhos futuros no contexto desta tese, quais sejam: 1) otimizações na implementação do oráculo de latência, 2) novas aplicações para ordenação de processos adaptativa baseada em oráculos de latência, ou, de forma genérica, aplicações para adaptação baseada em oráculos de latência e, 3) um serviço de ordenação de processos adaptativo independente.

Uma característica do oráculo de latência implementado é o baixo custo, traduzido, principalmente, pela sobrecarga de execução associada à solução. Nesse caso, optou-se por usar serviços oferecidos pelo ambiente de execução: serviço de detecção de falhas *pull* para

cálculo de *rtts*, *piggybacking* em mensagens do protocolo de consenso para difundir informações de cada módulo local do oráculo de latência e, protocolo de consenso para garantir informação global. Esta estratégia nem sempre é eficiente porque a configuração dos serviços do ambiente de execução pode ser ineficiente para o oráculo (ou, não tão eficientes quanto poderiam ser). Outra característica do oráculo de latência é a pouca exatidão de suas informações, isto porque, tais informações são construídas a partir do último *rtt* calculado, podendo comprometer a aproximação de latência no futuro realizada pelo oráculo. Ambas as características apontadas podem influenciar na eficiência do serviço provido pelo oráculo de latência e, por conseguinte, o serviço provido pela solução adaptativa baseada no mesmo.

Uma implementação diferente de oráculo de latência deveria considerar a relação custo-benefício de não usar serviços do ambiente de execução, podendo ajustar suas configurações de forma independente, além disso, usaria mecanismos mais robustos para gerar informações de latência, tal como modelos de previsão. Neste último caso, foi sugerido no Capítulo 4 o projeto e implementação de um oráculo de latência baseado em preditores, porém, tal implementação não foi considerada na avaliação de desempenho. Acredita-se que as estratégias sugeridas podem gerar implementações otimizadas de oráculos de latência. Otimizações para o oráculo de latência devem influenciar na exatidão das informações providas, diminuindo a possibilidade de escolhas erradas por parte da solução adaptativa baseada no mesmo. Em outras palavras, otimizações refletem na eficiência dos oráculos de latência. Uma questão relevante sobre eficiência de oráculos de latência refere-se à definição de métricas de qualidade de serviço. Este assunto também pode gerar trabalhos futuros interessantes.

Nesta tese, oráculos de latência foram usados para construir uma solução adaptativa para o problema da ordenação de processos. Além disso, utilizou-se como estudo de caso protocolos de consenso simétricos pelo texto. Por um lado, seria interessante investigar novas aplicações para ordenação de processos adaptativa. Da mesma forma, pode-se investigar outras soluções adaptativas baseadas em oráculos de latência, as quais seriam aplicadas a protocolos simétricos pelo texto.

Ainda no contexto de soluções adaptativas para ordenação de processos, uma observação importante refere-se ao critério de ordenação utilizado. Na solução implementada, o critério é representado por uma função baseada na mediana, a qual é aplicada sobre as informações globais providas pelo oráculo de latência. Possivelmente, existem critérios mais eficientes. Um exemplo seria capacitar o serviço de ordenação para ser configurado quanto ao critério de ordenação, que seria fornecido pela aplicação usuário do serviço. Esta estratégia tem sido

aplicada no contexto de detecção de falhas para construir serviços adaptativos (DéFAGO et al., 2005). Outra observação importante refere-se à implementação de uma solução adaptativa para ordenação de processos através de serviços independentes do sistema. Isto envolveria alterações em todos os componentes da solução, incluindo o oráculo de latência, tornando-os disponíveis para diferentes aplicações.

Bibliografia

AGUILERA, M. K. et al. Stable leader election. In: Proceedings of the 15th International Symposium on Distributed Computing, LNCS 2180. Lisbon, Portugal: Springer, 2001. p. 108–122.

AKSIT, M.; CHOUKAIR, Z. Dynamic, adaptive and reconfigurable systems - overview and prospective vision. In: Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops (ICDCSW'2003). Providence, Rhode Island: IEEE Computer Society, 2003. p. 84–89.

AMIR, Y. et al. Scaling byzantine fault-tolerant replication towide area networks. In: Proceedings of the International Conference on Dependable Systems and Networks (DSN'2006). Philadelphia, USA: IEEE Computer Society, 2006. p. 105–114.

ANDERSEN, D. et al. Resilient overlay networks. Operating Systems Review, v. 35, n. 5, p. 131–145, dez. 2001.

ANDRZEJAK, A. et al. Algorithms for Self-Organization and Adaptive Service Placement in Dynamic Distributed Systems. [S.l.], 2002.

ANTONIS, K. et al. A hierarchical adaptive distributed algorithm for load balancing. Journal of Parallel and Distributed Computing, v. 64, n. 1, p. 151–162, jan. 2004.

ATIGHETCHI, M. et al. Building auto-adaptive distributed applications: The quo-apod experience. In: Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops (ICDCSW'2003). Providence, Rhode Island: IEEE Computer Society, 2003. p. 104–109.

ATTIYA, H.; BORTNIKOV, V. Adaptive and efficient mutual exclusion. Distributed Computing, v. 15, n. 3, p. 177–189, 2002.

ATTIYA, H.; FOUREN, A. Adaptive and efficient algorithms for lattice agreement and renaming. SIAM Journal on Computing, v. 31, n. 2, p. 642–664, 2001.

BARBETTA, P. A.; REIS, M. M.; BORNIA, A. C. Estatística para cursos de engenharia e informática. São Paulo, Brazil: Editora Atlas, 2004.

BERMAN, F. et al. Adaptive computing on the grid using AppLeS. Communications of the ACM, v. 45, n. 5, p. 30–33, maio 2002.

BODNARCHUK, R.; BUNT, R. B. A synthetic workload model for a distributed system file server. In: Proceedings of the 1991 ACM SIGMETRICS conference on Measurement and modeling of computer systems. San Diego, United States: ACM Press, 1991. p. 50–59.

BONDAVALI, A.; GIANDOMENICO, F. D.; XU, J. A cost-effective and flexible scheme for software fault tolerance. Journal of Computer Systems Science and Engineering, v. 8, n. 4, p. 234–244, 1993.

BRASILEIRO, F. V.; EZHILCHELVAN, P. D.; SPEIRS, N. A. TMR processing without explicit clock synchronisation. In: Proceedings of the 14^{th} IEEE Symposium on Reliable Distributed Systems (SRDS'95). Bad Neuenahr, Germany: IEEE Computer Society, 1995. p. 186–195.

BRASILEIRO, F. V. et al. Consensus in one communication step is possible. In: Proceedings of the Sixth International Conference on Parallel Computing Technologies. Novosibirsk, Russia: Springer, 2001. (Lecture Notes in Computer Science, v. 2127), p. 42–50.

BRUSILOVSKY, P. Knowledgetree: a distributed architecture for adaptive e-learning. In: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters. New York, USA: ACM Press, 2004. p. 104–113.

BRUSILOVSKY, P.; MAYBURY, M. T. From adaptive hypermedia to the adaptive web. Communication of the ACM, v. 45, n. 5, p. 30–33, maio 2002.

BUHRMAN, H. et al. On the importance of having an identity or, is consensus really universal? Distributed Computing, v. 18, n. 3, p. 167–176, fev. 2006.

CACHIN, C.; PORITZ, J. A. Secure intrusion-tolerant replication on the internet. In: Proceedings of the International Conference on Dependable Systems and Networks (DSN'2002). Bethesda,USA: IEEE Computer Society, 2002. p. 167–176.

CACHIN, C.; SAMAR, A. Secure distributed dns. In: Proceedings of the International Conference on Dependable Systems and Networks (DSN'2004). Florence, Italy: IEEE Computer Society, 2004. p. 423–432.

CAMARGOS, L. J.; PEDONE, F.; MADEIRA, E. R. M. Optimal and practical wab-based consensus algorithms. In: Proceedings of 12th International Euro-Par Conference (Euro-Par'2006). Dresden, Germany: Springer, 2006. p. 549–558.

CATãO, B.; BRASILEIRO, F.; OLIVEIRA, A. C. Engineering a failure detection service for widely distributed systems. In: Anais do VI Workshop de Testes e Tolerância a Faltas (WTF'2005). Fortaleza, Brasil: [s.n.], 2005. p. 111–122.

CHANDRA, R.; RAMASUBRAMANIAN, V.; BIRMAN, K. P. Anonymous gossip: Improving multicast reliability in mobile ad-hoc networks. In: Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS'2001. Phoenix, USA: IEEE Computer Society, 2001. p. 275–283.

CHANDRA, T. et al. On the impossibility of group membership. In: Proceedings of the 15th ACM Symposium on Principles of Distributed Computing (PODC'96). Philadelphia, PA, USA: ACM, 1996. p. 322–330.

CHANDRA, T.; TOUEG, S. Unreliable failure detectors for reliable distributed systems. Journal of the ACM, v. 43, n. 2, p. 225–267, mar. 1996.

CHANDRA, T. D.; HADZILACOS, V.; TOUEG, S. The weakest failure detector for solving consensus. Journal of the ACM, v. 43, n. 4, p. 685–722, jul. 1996.

CHEN, W.; TOUEG, S.; AGUILERA, M. K. On the quality of service of failure detectors. In: International Conference on Dependable Systems and Networks (DSN'2000). New York, USA: IEEE Computer Society, 2000. p. 191–200.

CHEN, W.-K.; HILTUNEN, M. A.; SCHLICHTING, R. D. Constructing adaptive software in distributed systems. In: Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS'2001. Phoenix, Arizona, USA: IEEE Computer Society, 2001. p. 635–643.

CHOCKLER, G.; MALKHI, D.; REITER, M. Evaluating the running time of a communication round over the internet. In: Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS'2001). Phoenix, USA: IEEE Computer Society, 2001. p. 11–20.

CIRNE, W. et al. Labs of the world, unite!!! Journal of Grid Computing, v. 4, n. 3, p. 225–246, 2006.

COCCOLI, A. et al. Performance analysis of a consensus algorithm combining stochastic activity networks and measurements. In: International Conference on Dependable Systems and Networks (DSN'2002). Washington, D.C., USA: IEEE Computer Society, 2002. p. 551–560.

CRISTIAN, F.; FETZER, C. The timed asynchronous distributed system model. IEEE Transactions on Parallel and Distributed Systems, v. 10, n. 6, p. 642–657, jun. 1999.

DABEK, F. et al. Designing a dht for low latency and high throughput. In: Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI '04). San Francisco, USA: Usenix Association, 2004. p. 85–98.

DEAN, J.; GHEMAWAT, S. Mapreduce: Simplified data processing on large clusters. In: Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI'2004). San Francisco, USA: Usenix Association, 2004. p. 137–150.

DELPORTE-GALLET, C.; FAUCONNIER, H. Latency measures and lower bounds for consensus with failure detectors. In: Proceedings of the 9th International Colloquium on Structural Information and Communication Complexity (SIROCCO'2002). Andros, Greece: Carleton Scientific, 2002. p. 87–100.

DELPORTE-GALLET, C. et al. The weakest failure detectors to solve certain fundamental problems in distributed computing. In: Proceedings of the XXIII Symposium on Principles of Distributed Computing. St. John's, Canada: ACM Press, 2004. p. 338–346.

DESWARTE, Y. et al. Saturne: a distributed computing system which tolerates faults and intrusions. In: Proceedings of the Workshop on Future Trends of Distributed Computing Systems in the 1990's. Hong Kong: IEEE Computer Society, 1988. p. 329–338.

DESWARTE, Y.; POWELL, D. Intrusion tolerance for internet applications. In: Proceedings of the IFIP 18th World Computer Congress. Toulouse, France: Kluwer Academic Publishers, 2004. p. 241–256.

DESWARTE, Y.; POWELL, D. Internet security: an intrusion-tolerance approach. Proceedings of the IEEE, v. 94, n. 2, p. 432–441, fev. 2006.

DEVORE, J. L. Probability and Statistics for Engineering and the Sciences. Pacific Grove, USA: Duxbury Press Belmont, 2000.

DOLEV, D.; DWORK, C.; STOCKMEYER, L. On the minimal synchronism needed for distributed consensus. Journal of the ACM, v. 34, n. 1, p. 77–97, jan. 1987.

DUTTA, P.; GUERRAOUI, R. Fast indulgent consensus with zero degradation. In: Proceedings of the 4*th* European Dependable Computing Conference (EDCC4). Toulouse, France: Springer, 2002. (LNCS, v. 2485), p. 191–208.

DWORK, C.; LYNCH, N. A.; STOCKMEYER, L. Consensus in the presence of partial synchrony. Journal of the ACM, v. 35, n. 2, p. 288–323, abr. 1988.

DéFAGO, X. et al. Definition and specification of accrual failure detectors. In: Proceedings of the International Conference on Dependable Systems and Networks (DSN'2005). Yokohama, Japan: IEEE Computer Society, 2005. p. 206–215.

FABRE, J.-C.; DESWARTE, Y.; RANDELL, B. Designing secure and reliable applications using fragmentation-redundancy-scattering: an object-oriented approach. In: Proceedings of the First European Dependable Computing Conference. Berlin, Germany: Springer, 1994. p. 21–38.

FELBER, P. et al. Failure detector as first class objects. In: International Symposium on Distributed Objects and Applications. Edinburgh, Scotland: IEEE Computer Society, 1999. p. 132–141.

FICH, F.; RUPPERT, E. Hundreds of impossibility results for distributed computing. Distributed Computing, v. 16, n. 2-3, p. 121–163, set. 2003.

FISCHER, M. J.; LYNCH, N. A.; PATERSON, M. D. Impossibility of distributed consensus with one faulty process. Journal of ACM, v. 32, n. 2, p. 374–382, abr. 1985.

FITZPATRICK, T. et al. Design and application of toast: An adaptive distributed multimedia middleware platform. In: Proceedings of the 8th International Workshop on Interactive Distributed Multimedia Systems. Lancaster, UK: Springer, 2001. (LNCS, v. 2158), p. 111–123.

FLOCCHINI, P. et al. Sorting and election in anonymous asynchronous rings. Journal of Parallel and Distributed Computing, v. 64, n. 2, p. 254–265, fev. 2004.

FRAGA, J.; POWELL, D. A fault and intrusion-tolerant file system. In: Proceedings of the 3rd International Conference on Computer Security. Dublin, Ireland: [s.n.], 1985. p. 203–218.

FRIEDMAN, R.; RAYNAL, M. On the benefits of the functional modular approach to distributed data management systems. In: Proceedings of the 23rd IEEE Symposium

on Reliable Distributed Systems (SRDS'2004). Florianópolis, Brazil: IEEE Computer Society, 2004. p. 10–20.

GALLENI, A.; POWELL, D. Consensus and Membership in Synchronous and Asynchronous Distributed Systems. 1996. LAAS Report 96104.

GREVE, F. et al. Primary component asynchronous group membership as an instance of a generic agreement framework. In: Proceedings of the 15th International Symposium on Autonomous Decentralized Systems (ISADS'2001). Dallas, Texas, USA: IEEE Computer Society, 2001. p. 93–100.

GREVE, F.; NARZUL, J.-P. L. Designing a configurable group service with agreement compontents. In: Proceedings of the XXII Brazilian Symposium on Computer Networks Workshops (SBRC'2004). Gramado, Brazil: [s.n.], 2004. p. 129–140.

GREVE, F.; NARZUL, J.-P. L. Generating fast atomic commit from hyperfast consensus. In: Proceedings of the Second Latin-American Symposium on Dependable Computing (LADC'2005). Salvador, Brazil: [s.n.], 2005. p. 226–244.

GUERRAOUI, R. et al. Consensus in asynchronous distributed systems: A concise guided tour. Springer LNCS: Advances in Distributed Systems, n. 1752, p. 33–47, 2000.

GUERRAOUI, R.; LARREA, M.; SCHIPER, A. Non blocking atomic commitment with an unreliable failure detector. In: Proceedings of the 14th Symposium on Reliable Distributed Systems (SRDS'1995). Bad Neuenahr, Germany: IEEE Computer Society, 1995. p. 41–50.

GUERRAOUI, R.; RAYNAL, M. The information structure of indulgent consensus. IEEE Transactions on Computers, v. 53, n. 4, p. 453–466, abr. 2004.

GUERRAOUI, R.; RUPPERT, E. What can be implemented anonymously? In: 19th International Symposium on Distributed Computing (DISC'2005). Cracow, Poland: [s.n.], 2005. p. 244–259.

GUERRAOUI, R.; SCHIPER, A. Consensus: The big misunderstanding. In: Proceedings of the 6th IEEE Workshop on Future Trends in Distributed Computing Systems (FTDCS-6). Tunis, Tunisia: IEEE Computer Society, 1997. p. 183–188.

GUERRAOUI, R.; SCHIPER, A. The generic consensus service. IEEE Transactions on Software Engineering, v. 27, n. 1, p. 29–41, jan. 2001.

GUMMADI, K. P.; SAROIU, S.; GRIBBLE, S. D. King: estimating latency between arbitrary internet end hosts. In: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurement. Marseille, France: ACM Press, 2002. p. 5–18.

HILTUNEN, M. A.; SCHLICHTING, R. D. Adaptive distributed and fault-tolerant systems. Computer systems science and engineering, v. 11, n. 5, p. 125–133, set. 1996.

HURFIN, M. et al. A consensus protocol based on a weak failure detector and a sliding round window. In: Proc. of the 20th IEEE Symposium on Reliable Distributed Systems (SRDS'2001). New-Orleans,LA,USA: IEEE Computer Society, 2001. p. 120–129.

HURFIN, M. et al. A general framework to solve agreement problems. In: Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS'99). Lausanne, Switzerland: IEEE Computer Society, 1999. p. 56–65.

HURFIN, M. et al. A fault-tolerant protocol for resource allocation in a grid dedicated to genomic applications. In: Proceedings of the 5th International Conference on Parallel Processing and Applied Mathematics, Special Session on Parallel and Distributed Bioinformatic Applications (PPAM-03). Czestochowa, Poland: Springer, 2003. (LNCS, v. 3019), p. 1154–1161.

JACOBSON, V. Congestion avoidance and control. In: Proceedings of the SIGCOMM'88. Stanford, CA, USA: ACM Press, 1988. p. 314–332.

JAIN, R. The art of computer systems performance analysis - Techniques for experimental design, measurement, simulation and modeling. New York: John Wiley & Sons, Inc., 1991.

KEIDAR, I. Challenges in evaluating distributed algorithms. In: Future Directions in Distributed Computing. [S.l.]: Springer, 2003. (Lecture Notes in Computer Science, LNCS, v. 2584), p. 40–44.

KEIDAR, I.; RAJSBAUM, S. Open questions on consensus performance. In: Future Directions in Distributed Computing, LNCS-2584, Springer. [S.l.]: Springer, 2003. (LNCS, v. 2584), p. 35–39.

KILLIJIAN, M.-O.; FABRE, J.-C. Adaptive fault tolerant systems: reflective design and validation. In: Proceedings of 17th International Parallel and Distributed Processing Symposium Workshops (IPDPS'2003). Nice, France: IEEE Computer Society, 2003. p. 212.

KON, F. et al. The case for reflective middleware. Communications of the ACM, v. 45, n. 6, p. 33–38, jun. 2002.

KRANAKIS, E. Symmetry and computability in anonymous networks. In: Proceedings of the 3rd International Colloquium on Structural Information and Communication Complexity (SIROCO'1996). Siena, Italy: [s.n.], 1996. p. 1–16.

LAMPORT, L. A theorem on atomicity in distributed algorithms. Distributed Computing, v. 4, n. 2, p. 59–68, jun. 1990.

LAMPORT, L. Paxos made simple. SIGACT News, v. 32, n. 4, p. 18–25, dez. 2001.

LARREA, M.; FERNáNDEZ, A.; ARéVALO, S. Optimal implementation of the weakest failure detector for solving consensus. In: 19th IEEE Symposium on Reliable Distributed Systems (SRDS'2000). Nurnberg, Germany: IEEE Computer Society, 2000. p. 52–59.

LIN, K.; HADZILACOS, V. Asynchronous group membership with oracles. In: Proceedings of the 13th International Symposium on Distributed Computing (DISC'1999). Bratislava, Slavak Republic: [s.n.], 1999. p. 79–93.

LYNCH, N. A. Distributed Algorithms. California: Morgan Kaufmann Publishers, Inc., 1996.

MOSTEFAOUI, A.; MOURGAYA, E.; RAYNAL, M. An introduction to oracles for asynchronous distributed systems. Future Generation Computer Systems, v. 18, n. 6, p. 757–767, maio 2002.

MOSTEFAOUI, A.; RAYNAL, M. Solving consensus using chandra toueg's unreliable failure detectors: a general quorum based approach. In: Proceedings of the 13^{th} International Symposium on Distributed Computing (DISC'99). Bratislava, Slovaquia: [s.n.], 1999. p. 49–63.

MOSTEFAOUI, A.; RAYNAL, M.; TRAVERS, C. Exploring Gafni's reduction land: from Ω^k to wait-free adaptive (2p - [p/k])-renaming via k-set agreement. [S.l.], 2006.

NIEUWPOORT, R. V. van; KIELMANN, T.; BAL, H. E. Efficient load balancing for wide-area divide-and-conquer applications. In: Proceedings of the $8th$ ACM SIGPLAN symposium on Principles and Practices of Parallel Programming. Snowbird, USA: ACM, 2001. p. 34–43.

NIEUWPOORT, R. V. van et al. Adaptive load balancing for divide-and-conquer grid applications. Journal of Supercomputing, 2006. Accepted for publication.

NUNES, R. C. Adaptação dinâmica do timeout de detectores de defeitos através do uso de séries temporais. Tese (Doutorado) — Universidade Federal do Rio Grande do Sul - UFRGS, 2003.

NUNES, R. C.; JANSCH-PôRTO, I. Modelling communication delays in distributed systems using time series. In: Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems (SRDS'2002). Osaka, Japan: IEEE Computer Society, 2002. p. 268–273.

NUNES, R. C.; JANSCH-PôRTO, I. Qos of timeout-based self-tuned failure detectors: the effects of the communication delay predictor and the safety margin. In: Proceedings of the International Conference on Dependable Systems and Networks (DSN'2004). Florence, Italy: IEEE Computer Society, 2004. p. 753–761.

PATERSON, L. et al. A blueprint for introducing disruptive technology into the internet. Computer Communication Review, v. 33, n. 1, p. 59–64, jan. 2003.

PEDONE, P. et al. Solving agreement problems with weak ordering oracles. In: Proceedings of the 4^{th} European Dependable Computing Conference (EDCC-4). Toulouse, France: [s.n.], 2002. p. 44–61.

RASCHID, L. et al. Latency profiles: performance monitoring for wide area applications. In: Proceedings of the III IEEE Workshop on Internet Applications. San Jose, USA: [s.n.], 2003. p. 74–78.

RAVINDRAN, B.; DEVARASETTY, R. K.; SCHIRAZI, B. Adaptive resource management algorithms for periodic tasks in dynamic real-time distributed systems. Journal of Parallel and Distributed Computing, v. 62, n. 10, p. 1527–1570, out. 2002.

RAVINDRAN, K. N.; KWIAT k. A.; SABBIR, A. Adapting distributed voting algorithms for secure real-time embedded systems. In: Proceedings of the 24th International Conference on Distributed Computing Systems Workshops (ICDCSW'2004). Hachioji, Tokyo, Japan: IEEE Computer Society, 2004. p. 347–353.

RAYNAL, M. Distributed Algorithms and Protocols. [S.l.]: John Wiley & Sons, Ltd., 1988.

REN, Y. et al. Aqua: An adaptive architecture that provides dependable distributed objects. IEEE Transactions on Computers, v. 52, n. 1, p. 31–50, jan. 2003.

RHEA, S. et al. Fixing the embarrassing slowness of opendht on planetlab. In: Proceedings of the Second Workshop on Real, Large Distributed Systems (WORLDS'05). San Francisco, USA: Usenix Association, 2005. p. 25–30.

RICCIARDI, A. M.; BIRMAN, K. P. Using process groups to implement failure detection in asynchronous environments. In: Proceedings of the 10th ACM Symposium on Principles of Distributed Computing. Montreal, Quebec, Canada: ACM Press, 1991. p. 341–353.

RODRIGUES, L. The road to a more configurable and adaptive communication and coordination support. In: Proceedings of the 9th Workshop on Future Trends of Distributed Computing Systems. San Juan, Puerto Rico: IEEE Computer Society, 2003. p. 16–22.

RODRIGUES, L.; FONSECA, H.; VERíSSIMO, P. Totally ordered multicast in large-scale systems. In: Proceedings of the 16th International Conference on Distributed Computing Systems (ICDCS'1996). Hong Kong: IEEE Computer Society, 1996. p. 503–510.

RODRIGUES, L. et al. The globdata fault-tolerant replicated distributed object database. In: Proceedings of the First EurAsian Conference - Information and Communication Technology. Shiraz, Iran: Springer, 2002. p. 426–433.

SAMPAIO, L. et al. Evaluating the impact of simultaneous round participation and decentralized decision on the performance of consensus. In: Proceedings of the International Conference on Dependable Systems and Networks (DSN'2007). [S.l.: s.n.], 2007. Accepted for publication.

SAMPAIO, L. M. R.; BRASILEIRO, F. V. Adaptive indulgent consensus. In: Proceedings of the International Conference on Dependable Systems and Networks (DSN'2005). Yokohama, Japan: IEEE Computer Society, 2005. p. 422–431.

SAMPAIO, L. M. R. et al. How bad are wrong suspicions? towards adaptive distributed protocols. In: Proceedings of the International Conference on Dependable Systems and Networks (DSN'2003). San Francisco, California, USA: IEEE Computer Society, 2003. p. 551–560.

SAMPAIO, L. M. R.; BRASILEIRO, F. V.; MOREIRA, A. L. C. Performance analysis of adaptive consensus protocols based on slowness oracles. In: Proceedings of the 24th

International Conference on Distributed Computing Systems Workshops (ICDCS'2004). Hachioji, Tokyo, Japan: IEEE Computer Society, 2004. p. 340–346.

SAMPAIO, L. M. R.; BRITO, A. E. M.; OLIVEIRA, E. W. A. de. Detectores de falhas em sistemas assíncronos (tutorial). In: Anais do XXI Workshop de Testes e Tolerância a Faltas (WTF'2003). Natal, Rio Grande do Norte, Brasil: [s.n.], 2003. p. 3–29.

SAMPAIO, L. M. R. et al. Efficient and robust adaptive consensus services based on oracles. Journal of the Brazilian Computer Society, v. 10, n. 3, p. 33–43, 2005.

SCHANTZ, R. E.; SCHMIDT, D. C. Research advances in middleware for distributed systems. In: Proceedings of the IFIP Conferece. Montréal, Québec, CA: Kluwer, 2002. p. 1–36.

SCHEIDER, F. B. What good are models and what models are good. In: _____. Distributed Systems. New York: ACM Press, 1993. cap. 2, p. 17–25.

SCHIPER, A. Early consensus in an asynchronous system with a weak failure detector. Distributed Computing, v. 10, n. 3, p. 149–157, abr. 1997.

SERGENT, N.; DéFAGO, X.; SCHIPER, A. Impact of a failure detection mechanism on the performance of consensus. In: Proceedings of the 2001 Pacific Rim International Symposium on Dependable Computing (PRDC'2001). Seoul, Korea: IEEE Computer Society, 2001. p. 137–145.

SHARMA, P. et al. Estimating network proximity and latency. ACM SIGCOMM Computer Communication Review, v. 36, n. 3, p. 39–50, jul. 2006.

SILVA, F. J. S.; ENDLER, M.; KON, F. Developing adaptive distributed applications: a framework overview and experimental results. In: Proceedings of the International Symposium on Distributed Objects and Applications. Catania, Sicily, Italy: Springer, 2003. (LNCS, v. 2888), p. 1275–1291.

SMITH, C. U.; WILLIAMS, L. G. Performance and scalability of distributed software architectures: An SPE approach. Parallel and Distributed Computing Practices, v. 3, n. 4, dez. 2000.

SOUZA, A. D. D. de. Structuring Adaptive Applications Using AspectJ. Dissertação (Master's thesis) — Informatics Center, Federal University of Pernambuco, Recife, fev. 2004.

SPRING, N. et al. Using planetlab for network research: myths, realities, and best practices. Operating Systems Review, v. 40, n. 1, p. 17–24, jan. 2006.

TUREK, J.; SHASHA, D. The many faces of consensus in distributed systems. IEEE Computer, v. 25, n. 6, p. 8–17, jun. 1992.

URBáN, P.; DéFAGO, X.; SCHIPER, A. Contention-aware metrics for distributed algorithms: comparison of atomic broadcast algorithms. In: Proceedings of the 9th IEEE International Conference on Computer Communications and Networks (IC3N'2000). Las Vegas, Nevada, USA: IEEE Computer Society, 2000. p. 80–92.

URBáN, P.; DéFAGO, X.; SCHIPER, A. Neko: a single environment to simulate and prototype distributed algorithms. In: Proceeding of the 15th International Conference on Information Networking (ICOIN-15). Beppu City, Japan: IEEE Computer Society, 2001. p. 503–511.

URBáN, P. et al. Performance comparison of a rotating coordinator and a leader based consensus algorithm. In: Proceedings of the 23rd Symposium on Reliable Distributed Systems (SRDS'2004). Florianópolis, Brazil: IEEE Computer Society, 2004. p. 4–17.

URGAONKAR, B.; SHENOY, P. Cataclysm: policing extreme overloads in internet applications. In: Proceedings of the 14th International Conference on World Wide Web (WWW'2005). Chiba, Japan: ACM Press, 2005. p. 740–749.

VERíSSIMO, P. et al. Intrusion-tolerant middleware: The road to automatic security. IEEE Security & Privacy, v. 4, n. 4, p. 54–62, jun. 2006.

WOLSKI, R. Experiences with predicting resource performance on-line in computational grid settings. SIGMETRICS Performance Evaluation Review, v. 30, n. 4, p. 41–49, mar. 2003.

WOLSKI, R.; SPRING, N.; HAYES, J. The network weather service: A distributed resource performance forecasting service for metacomputing. Journal of Future Generation Computing Systems, v. 15, n. 5-6, p. 757–768, out. 1999.

WONG, B.; SILVKINS, A.; SIRER, E. G. Meridian: a lightweight network location service without virtual coordinates. In: Proceedings of the ACM SIGCOMM 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. Philadelphia, USA: ACM Press, 2005. p. 85–96.

YAN, Y.; ZHANG, X.; YANG, H. A fast token-chasing mutual exclusion algorithm in arbitrary network topologies. Journal of Parallel and Distributed Computing, v. 35, n. 2, p. 156–172, jun. 1996.

ZADOROZHNY, V. et al. Wide area performance monitoring using aggregate latency profiles. In: Proceedings of the Fourth International Conference on Web Engineering (ICWE'2004). Munich, Germany: Springer, 2004. (LNCS 3140), p. 386–390.

ZINKY, J.; BAKKEN, D.; SCHANTZ, R. Architectural support for quality of service for corba objects. Theory and Practice of Objects Systems, v. 1, n. 3, p. 55–73, abr. 1997.