UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

PROPOSTA DE UM ESQUEMA DE TOLERÂNCIA A FALTAS BASEADO EM MÚLTIPLAS REPLICAÇÕES DE PROCESSOS: ESQUEMA MR

TESE SUBMETIDA À UNIVERSIDADE FEDERAL DE SANTA CATARINA PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS, ÁREA ENGENHARIA ELÉTRICA, ÁREA DE CONCENTRAÇÃO SISTEMAS DE INFORMAÇÃO

Luiz Nacamura Júnior

Florianópolis, 21 de Novembro de 1996.

PROPOSTA DE UM ESQUEMA DE TOLERÂNCIA A FALTAS BASEADO EM MÚLTIPLAS REPLICAÇÕES DE PROCESSOS: ESQUEMA MR

Luiz Nacamura Júnior

ESTA TESE FOI JULGADA ADEQUADA PARA A OBTENÇÃO DO TÍTULO DE

DOUTOR EM ENGENHARIA ELÉTRICA

ÁREA ENGENHARIA ELÉTRICA, ÁREA DE CONCENTRAÇÃO SISTEMAS DE INFORMAÇÃO, E APROVADA EM SUA FORMA FINAL PELO PROGRAMA DE PÓS-GRADUAÇÃO

JONI DA SILVA FRAGA, Dr.
(Orientador)

ADROALDO RAIZER, Dr.
(Coordenador do Curso)

BANCA EXAMINADORA:

JONI DA SILVA FRAGA, Dr.
(Presidente)

ORLANDO GOMES LOQUES FILHO, PhD.
(Relator)

ALDERICO R. DE PAULA JR, PhD.

JEAN-MARIE FARINES, Dr. Ing.

VITÓRIO BRUNO MAZZOLA, Dr.

À minha esposa Denise Luz Nacamura

AGRADECIMENTOS

Em primeiro lugar agradeço ao meu orientador e amigo Joni da Silva Fraga pela sua orientação, incentivo e conselhos dados durante estes longos anos de convivência.

Agradeço também aos membros da banca examinadora que aprovaram esta tese: Prof. Orlando Gomes Loques Filho, Prof. Alderico R. de Paula Jr., Prof. Jean-Marie Farines e Prof Vitório Bruno Mazzola.

A todos os professores, colegas e funcionários do Programa de Pós-Graduação em Engenharia Elétrica e do Laboratório de Controle e MicroInformática (LCMI), que de alguma forma, ou de outra, contribuíram para a realização deste trabalho. Em especial ao coordenador do LCMI Jean-Marie Farines pelo incentivo e esforço despendido para que este trabalho pudesse ser realizado, aos bolsistas Marcelo Sobral e Vladimir pela dedicação, ao amigo Lau pela companhia e longas conversas noturnas e aos colegas de Doutorado Rômulo, Keiko e Olinto.

Agradeço aos Professores e companheiros do CEFET-PR pela compreensão e paciência com que suportaram as idas e vidas à Florianópolis.

A minha esposa Denise Luz Nacamura, e aos meus pais, Luiz Nacamura e Zuleika V. Nacamura, e as minhas irmãs Luiza de F. Nacamura Silva, Liliane Nacamura e Luciane de C. Nacamura pelo apoio e compreensão.

Ao meu amigo Wallace F. Lobo Jr pela hospedagem e pelas conversas e discussões que tornaram o fim desta tese mais agradável.

Agradeço a UFSC e ao CNPq pelo apoio financeiro que permitiu a realização deste trabalho.

Proposta de um Esquema de Tolerância a Faltas baseado em Múltiplos Replicações de Processos: Esquema MR

Capítulo 3: Técnicas de Replicação em Sistemas Distribuídos

RESUMO

Neste trabalho é proposto um esquema de tolerância a faltas para sistemas distribuídos que faz uso extensivo dos recursos computacionais, através da replicação massiva de processos. Esse esquema, denominado de MR (Múltiplas Replicações), apresenta uma estrutura adaptável a aspectos ligados às características de evolução de uma aplicação.

No esquema MR, os algoritmos base de cada uma das replicações são soluções diferenciadas de um mesmo problema de aplicação. As diferentes replicações montadas a partir desses algoritmos são dispostas no esquema no sentido de atender os princípios da diversidade de projeto e da diversidade de dados. A tolerância a faltas de hardware é inerente ao esquema pelo uso explícito de replicações.

A cada execução do MR, com base em atributos dos dados de entrada é selecionada dinamicamente uma replicação própria para o processamento dos mesmos. Na verdade, essa seleção determina uma classe de algoritmos alternativos, apropriados para o processamento dos dados de entrada. Os algoritmos nessa classe estão dispostos, segundo critérios de precisão ou de desempenho, em uma ordem que determina o algoritmo preferencial e as alternativas. A replicação ativada executa inicialmente o algoritmo preferencial da classe associada.

Ao longo desse texto é mostrado todo o esforço realizado no sentido de verificar a viabilidade desse novo esquema. Usando o modelo de traços CSP [Hoare 85], procuramos mostrar a adequação do mesmo às provas de correção da Teoria de Processos Replicados [Mancini 88].

Estudos sobre o Desempenho e a Segurança de Funcionamento do MR, também são realizados no sentido da comparação com outros esquemas presentes na literatura.

ABSTRACT

This work presents a fault tolerance schema to distributed systems that makes use extensive computational resources. In this schema, named MR (Replication Multiples), the structure is adaptable to the evolution of application characteristics.

In the schema MR, the algorithms base of one replications are different solutions to the same application problem. The different replications of these algorithms are disposed in the schema MR, in order to support the principles of the design and data diversity. The fault tolerance of the hardware is inherent of the schema for the usage explicit of replications.

In each execution of the MR, one replication adequate is selected dynamically with base data input attributes. This selection determines one class of alternatives algorithms, appropriate to the processing of the data input. These algorithms classes are disposed, according to criterion the precision or the performance, in the order to determine the principal algorithm and the alternatives algorithms. Firstly, the replication execute the principal algorithm of the associated class.

This text represents a whole effort in the sense the to check the viability of the schema MR. Using the CSP model [Hoare 85], we show that it satisfies the correction proofs of the Theory of Replication Process [Mancini 88].

This text also presents a comparative study of the performance and dependability of the schema MR.

CAPÍTULO 1

INTRODUÇÃO

A proliferação de serviços fornecidos através de sistemas computacionais tem alterado profundamente o nosso cotidiano. Os requisitos impostos a esses sistemas dependem dos tipos de aplicação e da qualidade de serviço desejada. Nesse texto, estamos interessados em aplicações envolvendo a necessidade de serviços computacionais que sejam continuamente oferecidos mesmo na presença de componentes faltosos.

Anteriormente, as técnicas de tolerância a faltas eram limitadas às aplicações críticas onde falhas em sistemas computacionais poderiam representar risco de vidas ou ainda prejuízos financeiros. Alguns exemplos dessas aplicações são os sistemas de controle de vôo; sistemas de controle e navegação de mísseis, sistemas de monitoração de pacientes, sistemas bancários, etc. Nos sistemas atuais, devido a complexidade dos mesmos, mecanismos de tolerância a faltas são usados em diferentes níveis no sentido de manter as funcionalidades e os serviços importantes para a consistência do sistema como um todo. Como exemplo, podemos citar os serviços de nomes e serviços de arquivos em sistemas distribuídos onde técnicas de replicação são empregadas para manter a continuidade desses serviços.

Tolerância a faltas em sistemas distribuídos

Uma das mais desejadas propriedades de um sistema computacional distribuído é a sua potencial capacidade para tolerância a faltas de componentes de hardware. Em contrapartida, as características desses sistemas introduzem novos problemas. A **consistência interativa** é um desses problemas [Lamport 82].

A tolerância a faltas nesses sistemas tem a ênfase posta nas interações (nas comunicações) e as falhas de componentes são associadas a nível de nó de processamento. As semânticas de falhas assumidas nesses componentes vão determinar os custos envolvidos para manter a consistência interativa.

Serviços confiáveis são usualmente construídos, tomando como base a distribuição de réplicas de componentes de software entre diferentes nós de processamento do sistema; isto é, fazendo uso de técnicas de replicação de componentes de software [Powell 91]. O **conceito de grupo** [Liang 90] que a algum tempo atrás era pouco explorado, vem se difundindo como uma forma natural de expressar replicações em sistemas distribuídos

Muitos dos modelos de replicação presentes na literatura toleram faltas de hardware, dentro das limitações impostas nas suas hipóteses de falta. Esses modelos explicitam as interações entre as réplicas e as preocupações em manter a consistência de estado. Normalmente, características da aplicação são ignoradas nesses modelos. Para se tratar faltas por valor ou mesmo faltas de software são necessários mecanismos adicionais ou técnicas dependentes da aplicação.

A tolerância a faltas de software é normalmente fundamentada nos princípios da **diversidade de projeto** [Aviziens 84] e da **diversidade de dados** [Ammann 87]. Com isso, os esquemas de tolerância a faltas de software são construídos, combinando códigos de aplicação e mecanismos de estreita ligação com as semânticas da aplicação. Os esquemas usuais como o Bloco de Recuperação (RB) [Randell 75], Programação N-Versões (PNV) [Avizienis 77], Bloco de Recuperação Distribuído (DRB) [Kim 88], etc. apresentam estruturas estáticas pouco adaptáveis às características de processamento em sistemas distribuídos. Além disso, muitos desses esquemas convencionais têm o seu uso dependente de arquiteturas especiais de hardware. Esse é o caso do esquema DRB.

Proposta e objetivos do trabalho

Esta tese de doutorado tem como objetivo principal introduzir um esquema de tolerância a faltas para sistemas distribuídos que faça uso extensivo dos recursos computacionais, através da replicação massiva de processos.

Os modelos de execução desse esquema são definidos no sentido de explorar de maneira intensiva conceitos referentes a processamento de grupo. Embora a noção de replicação em base de dados distribuídos seja relativamente sedimentada (replicação de dados), o uso da noção de replicações de processos se tornou mais intensa e viável com a propagação recente (início dos anos 90) de suportes para processamento em grupo.

Por outro lado, procuramos nesse esquema proposto criar estruturas adaptáveis a aspectos ligados às características de evolução de uma aplicação. O esquema deve então ser flexível o suficiente diante das condições de momento no processamento de aplicações em sistemas distribuídos.

Esse esquema, que procura a flexibilização em suas estruturas, é constituído por Múltiplas Replicações (MR) de processos. Os algoritmos base de cada uma dessas replicações são soluções diferenciadas de um mesmo problema de aplicação. As diferentes replicações montadas a partir desses algoritmos são dispostas no esquema no sentido de atender os princípios da diversidade de projeto e da diversidade de dados. A tolerância a faltas de hardware é inerente ao esquema pelo uso explícito de replicações.

A cada execução do MR, com base em atributos dos dados de entrada é selecionada dinamicamente uma replicação própria para o processamento dos mesmos. Na verdade, essa seleção determina uma classe de algoritmos alternativos, apropriados para o processamento dos dados de entrada. Os algoritmos nessa classe estão dispostos, segundo critérios de precisão ou de desempenho, em uma ordem que determina o algoritmo preferencial e as alternativas. A replicação ativada executa inicialmente o algoritmo preferencial da classe associada.

O esquema MR foi desenvolvido inicialmente para se executar em equipamentos convencionais, de propósito geral. A não ser pelas premissas sobre as semânticas de falhas, nenhuma exigência é colocada sobre os elementos de processamento ou o suporte de comunicação.

Ao longo desse texto é mostrado todo o esforço realizado no sentido de verificar a viabilidade desse novo esquema. Usando o modelo de traços CSP [Hoare 85], procuramos mostrar a adequação do mesmo às provas de correção da Teoria de Processos Replicados [Mancini 88].

Estudos sobre o Desempenho e a Segurança de Funcionamento do MR, foram realizados com o sentido da comparação com outros esquemas presentes na literatura.

Um outro objetivo que nos colocamos foi o de analisar as potencialidades do MR no sentido de construir Sistemas Distribuídos Replicados (SDRs). Neste caso, os MRs seriam os elementos usados para a composição de programas distribuídos replicados. Com esse objetivo estudamos as estruturas do MR no sentido de verificar sua adequação a diferentes modelos de programação, usuais em sistemas distribuídos. Os estudos sobre SDRs, tinham como base os trabalhos de [Ezhichelvan 89] sobre *triads* e seriam testados em redes de *transputer*. A indisponibilidade de tal suporte em nosso laboratório (LCMI/UFSC) fez com que todos os esforços fossem deslocados para a discussão e implementação de um protótipo do esquema MR em um ambiente distribuído composto por estações de trabalho interconectadas por uma rede local *(Ethernet)*.

Viabilidade da proposta do ponto de vista da flexibilidade

A premissa de que o processamento do MR se ajusta conforme as características dos dados de entrada, pressupõe a possibilidade da geração de algoritmos especializados ou *customizados (customized)* para as entradas correspondentes. Nessa linha existe um conjunto de propostas recentes onde os programas se adaptam a diferentes situações de momento na evolução do sistema.

Esse é o caso das propostas em [Park 96] onde um programa (paralelo) se adapta durante a sua execução, modificando sua configuração, para atender requisitos de desempenho. Nessa proposta, o compilador gera várias versões parametrizadas de um mesmo programa e em tempo de execução, o sistema vai se utilizando dessas versões conforme informações de desempenho disponíveis em determinados momentos. Outro exemplo representativo que segue essa linha é o compilador da linguagem orientada a objetos SELF [Ungar 87] que utiliza técnicas de customização [Hoüzle 95] para criar diferentes versões de métodos, otimizadas para diferentes parâmetros de entrada. No instante da execução, a versão apropriada é selecionada segundo o parâmetro de entrada da chamada.

O modelo **múltiplas versões** que é uma das técnicas de **computação imprecisa** usada em escalonamento tempo real também é um exemplo dessas técnicas de adaptação às condições

de momento na evolução de um sistema [Liu 91]. Nesse caso, na ativação de um programa, uma das versões é escolhida para executar no sentido de manter as restrições temporais impostas ao escalonamento do sistema; a versão mais precisa, por ser a mais lenta, só é escalonada quando as condições de cargas não indicam sobrecarga, o que determinaria a ocorrência de falhas de temporização no sistema.

Os trabalhos citados acima e alguns projetos atuais de pesquisa, como o sistema ***CHAMELEON*** [Parker 96], que visa desenvolver sistemas dinamicamente adaptáveis a condições de seus ambientes, mostram a possível importância da sugestão de estruturas flexíveis no esquema MR.

Propostas alternativas

Como citado anteriormente, a maioria dos esquemas convencionais de tolerância a faltas presentes na literatura apresenta uma estrutura fixa com escalonamento estático de versões (algoritmos), grau de replicação (número de componentes) constante e muitas vezes dependentes de uma arquitetura de hardware particular. Essas características de um modo geral dificulta o uso dos mesmos em sistemas distribuídos com aplicações evolutivas, caracterizando uma carga variável para o sistema.

Esses fatores têm encorajado o surgimento de novos esquemas como por exemplo o SCOP *(Self Configuring Optimal Programming)* [Bondavalli 93], PNV_TB (PNV com *Tie-Breaker*) [Tai 93b], que apresentam estruturas mais flexíveis, adaptáveis às condições de processamento do sistema.

Existem também propostas de suportes que incorporam dois ou mais esquemas de tolerância a faltas, e dependendo das condições do sistema (por exemplo, carga ou desempenho), estruturam os serviços no sistema segundo um desses esquemas. Em [Tai 94], os serviços confiáveis podem ser configurados como um esquema RB ou como um DRB. A escolha por um desses esquema depende da disponibilidade de elementos de processamento. Num primeiro momento, quando ainda existem recursos disponíveis no sistema os serviços são configurados segundo o esquema DRB que exige mais recursos de processamento. A medida que os recursos vão se tornando escassos os serviços passam a ser configurados através do esquema RBs. Nesta proposta, o conceito de performabilidade *(performability)* introduzido

em [Meyer 80] é usado para conduzir em tempo de execução a configuração desses serviços, construindo dessa forma uma tolerância a faltas adaptativa.

Finalmente, existem trabalhos que abordam o gerenciamento de réplicas em sistemas distribuídos e paralelos [Little 94],[Cristian 94] e [Wang 95]. O gerenciamento de réplicas nestes trabalhos tem por objetivo estabelecer o número ótimo de replicas, em tempo de execução, a cada ativação do serviço. Em [Little 94] e [Cristian 94] o número de réplicas é calculado segundo a qualidade de serviço desejada (desempenho, disponibilidade, etc.). Por outro lado, o número de réplicas em [Wang 95] é calculado a partir de requisitos de tempo de resposta máximo.

Nos estudos realizados foram deixados de fora modelos e técnicas de tolerância a faltas usados normalmente em sistemas transacionais. O escopo do trabalho foi limitado a replicação de processos.

Organização do trabalho

A organização deste trabalho reflete bem as etapas desenvolvidas durante o doutorado. No segundo capítulo é apresentada a taxionomia básica de segurança de funcionamento que é utilizada no decorrer deste texto. No restante do capítulo, são descritos os principais esquemas de tolerância a faltas de hardware e de software.

O terceiro capítulo enfoca os principais modelos de replicação usados em sistemas distribuídos. Embora esses modelos possam ser usados com diferentes propósitos em um sistema distribuído, a ênfase neste trabalho é dada para o emprego desses modelos na tolerância a faltas. Inicialmente neste capítulo são discutidos os principais conceitos relacionados com o conceito de grupo. A classificação de grupo baseada nos aspectos estruturais de seus componentes é apresentada, além das propriedades associadas aos diferentes protocolos de comunicação de grupo. As principais técnicas de replicação, o conceito de determinismo de réplicas e exemplos notórios do uso dessas técnicas em sistemas distribuídos são descritos na seqüência desse capítulo.

O capítulo 4 introduz o esquema MR fazendo uso do modelo de traço do CSP. O MR é analisado segundo a ótica da Teoria de Processos Replicados de [Mancini 88]. Na seqüência o MR é discutido como unidade de estruturação de aplicações distribuídas replicadas

O capítulo 5 trata do estudo comparativo do esquema MR com os principais esquemas presentes na literatura e discutidos no capítulo 2. A performabilidade *(performability)* [Meyer 80] é utilizada como medida de referência para a comparação dos esquemas.

Os resultados experimentais obtidos a partir de um protótipo do esquema MR implementado no LCMI/UFSC são apresentados no capítulo 6. O protótipo implementado faz uso da técnica de replicação Líder/Seguidores [Powell 91]. Os suportes usados nessa implementação caracterizam o modelo de execução do MR como assíncrono temporizado (*timed asynchronous* [Cristian 95]). Os valores de *timeout* usados nesse protótipo são obtidos a partir de uma análise de pior caso. Outros aspectos ligados às implementações das estruturas do MR, aos testes realizados também são objetos desse capítulo. Por fim são examinados alguns exemplos de aplicações que poderiam fazer uso do MR na implementação de serviços confiáveis.

Finalmente, no capítulo 7 são indicadas as principais conclusões e perspectivas futuras do trabalho.

CAPÍTULO 2

TOLERÂNCIA A FALTAS

Este capítulo tem por objetivo apresentar a terminologia e os conceitos básicos de **segurança de funcionamento** e **tolerância a faltas** a serem usados neste documento. A terminologia e os conceitos estão fortemente fundamentados nos trabalhos apresentados em [Anderson 81, Avizienis 77, Stok 90, Schepers 90] e, principalmente em [Laprie 92]. Os termos e conceitos utilizados em português estão baseados em um trabalho conjunto entre o INESC de Portugal e o LCMI/UFSC [Veríssimo 89, Lemos 88].

Além da taxionomia de segurança de funcionamento, é formalizada a semântica de falhas o que possibilita então caracterizar a classificação de faltas segundo seus efeitos. A classificação de faltas apresentada neste sentido, tem como base os trabalhos de [Powell 91, Cristian 90, Bondavalli 90, Veríssimo 89, Edwards 94].

Na continuação deste capítulo são examinados os principais esquemas de tolerância a faltas de hardware e de software presentes na literatura. Estes esquemas serão usados em estudos comparativos no decorrer deste texto.

2.1 Conceitos de segurança de funcionamento

Um **sistema** é, do ponto de vista estrutural, um conjunto de componentes configurados de maneira a interagir entre si. Um componente, por sua vez, também é um sistema. A recursividade neste caso, deve parar quando um sistema (componente) é considerado atômico: nenhuma estrutura interna pode ser discernida.

O **serviço** fornecido por um sistema é a abstração de seu comportamento, conforme este é percebido por seus usuários. O usuário é um outro sistema (físico ou humano) que interage com o sistema em questão. Desta forma, o serviço fornecido por um componente corresponde ao seu comportamento, tal como é percebido pelos outros componentes do sistema do qual faz parte.

A **segurança de funcionamento** (*dependability*) é a propriedade que permite a seus usuários terem confiança justificada no serviço fornecido pelo sistema.

Faltas, Erros e Falhas

Faltas, Erros e **Falhas** são imperfeições e portanto circunstâncias indesejáveis ao comportamento do sistema:

- o evento preocupante no comportamento do sistema é a **falha**, que corresponde a todo desvio das especificações de serviço. A falha é percebida pelo usuário e pode ser avaliada;
- o **erro** é a circunstância indesejável interna ao sistema. Os erros em um sistema podem ser detectados e recuperados. Sendo que durante a execução o sistema pode transitar entre estados corretos e por estados errôneos. A presença de estados errôneos pode levar à falha do sistema;
- a **falta** é a circunstância indesejável que provoca a ocorrência de erros no sistema. A ativação de uma falta provoca a transição que determina a passagem do sistema de um estado correto para um estado errôneo.

O presente modelo de imperfeições apresenta uma recursividade na sequência **falta** → **erro** → **falha** em relação a noção de sistema apresentada anteriormente, ou seja, a falha de um componente é uma falta a nível de sistema.

2.2 Semânticas de falhas e hipóteses de faltas

As classificações de faltas podem dar se segundo diversos fatores. Algumas das classificações apresentadas na literatura têm como motivação [Laprie 92]: a origem (física/humana, de projeto/de operação, etc.), a persistência (transitória/permanente), etc.

Um tipo de classificação importante e base para estratégias de tolerância a faltas de software, é aquela que considera a dependência entre faltas. Nesta classificação, diferentes faltas são ditas **independentes** quando os erros correspondentes são também distintos. Faltas relacionadas, resultantes de uma má especificação de projeto ou de uma má implementação, manifestam-se na forma de erros similares que conduzem as **falhas de modo comum**. Erros distintos, geralmente, causam **falhas separadas**.

Entretanto, a classificação de faltas mais difundida na literatura leva em conta as **semânticas de falhas** associadas. As semânticas de falhas de um servidor definem as maneiras pelas quais o servidor pode falhar. O projeto de servidores tolerantes a faltas deve utilizar-se de um conjunto de **hipóteses de faltas** (hipóteses sobre os possíveis tipos de faltas que podem ocorrer) no sentido de determinar as semânticas de falhas a serem previstas nas especificações de serviço.

As discussões que se seguem correspondem a uma síntese que tomou como base os trabalhos em [Cristian 90, Veríssimo 89, Bondavalli 90, Powell 90, Edwards 94], envolvendo a classificação de faltas segundo seus efeitos.

2.3 Classes de faltas segundo as semânticas de falhas associadas

2.3.1 Modelo de serviço

O modelo de serviço de um sistema com um único servidor pode ser definido em termos de ***ocorrências de serviços***, representado por o_i $(i=1,2,...)$, e caracterizadas por pares $<v_i, t_i>$, onde v_i é o valor ou conteúdo da ocorrência de serviço o_i e t_i o tempo ou instante de observação de o_i. Os valores e os tempos esperados para uma ocorrência o_i, correspondem a um conjunto, denominado de ***expectativa*** (E_i), tal que:

$$E_i \subseteq VS_i \times TS_i$$

onde:

VS_i : é o conjunto de valores esperados para a ocorrência de serviço o_i;

TS_i : é o conjunto de tempos especificados para o_i.

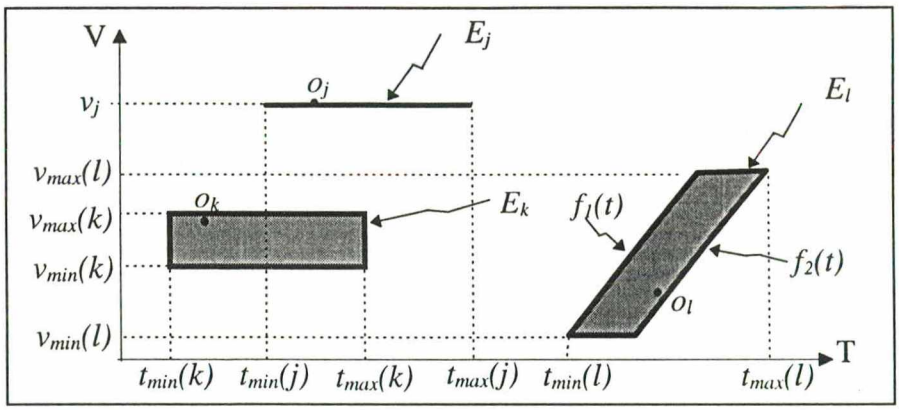


Figura 2.1: Exemplos de expectativas

A figura 2.1 ilustra para as ocorrências de serviço o_j, o_k e o_l as respectivas expectativas E_j, E_k e E_l são dadas por:

- $E_j = \{< v,t >: VS_j \text{ X } TS_j \mid (v=v_j) \wedge (t_{min}(j) \leq t \leq t_{max}(j)) \}$

  onde $VS_j=\{v_j\}$ e TS_j é dado pelo intervalo $[t_{min}(j); t_{max}(j)]$;

- $E_k = \{< v,t >: VS_k \text{ X } TS_k \mid (v_{min}(k) \leq v \leq v_{max}(k)) \wedge (t_{min}(k) \leq t \leq t_{max}(k))\}$

  TS_k e VS_k são dados pelos intervalos $[t_{min}(k); t_{max}(k)]$ e $[v_{min}(k); v_{max}(k)]$, respectivamente;

- $E_l = \{< v,t >: VS_l \text{ X } TS_l \mid (f_2(t) \leq v \leq f_1(t)) \wedge (t_{min}(l) \leq t \leq t_{max}(l))\}$

  na figura Figura 2.1, TS_l e VS_l são dados pelos intervalos $[t_{min}(l); t_{max}(l)]$ e $[v_{min}(l); v_{max}(l)]$, respectivamente. O espaço dos elementos $< v,t >$ de E_l é delimitado pelas curvas $f_1(t), f_2(t)$, $v_{min}(l)$ e $v_{max}(l)$.

Def. Uma ocorrência do serviço o_i é definida como **correta**, se e somente se:

$$((v_i \in VS_i) \wedge (t_i \in TS_i)) \wedge (o_i \in E_i)$$

2.3.2 Faltas no domínio dos valores

O espaço de valores v_i se utiliza, na representação dos mesmos, de um código *CV*. Segundo a teoria de codificação [Wakerley 78], os erros de valor podem ser classificados em **erros de valor no código** (*incode value error*) e em **erros de valor fora do código** (*noncode value error*) (ver figura 2.2). Tomando como base o citado, temos $VS_i \subseteq CV$ e, por conseqüência, as faltas no domínio dos valores podem ser distinguidas segundo seus modos de falhas:

- uma **falha por valor fora do código** ocorre em o_i, para $o_i = < v_i, t_i >$, quando:

$$(E_i \neq \Phi) \wedge (v_i \notin Vs_i) \wedge (v_i \notin CV)$$

- uma **falha por valor no código** ocorre em o_i, para $o_i = < v_i, t_i >$, quando:

$$(E_i \neq \Phi) \wedge (v_i \notin Vs_i) \wedge (v_i \in CV)$$

As faltas associadas exclusivamente com falhas por valor são identificadas como **faltas por valor** ou **faltas de computação incorreta** [Barborak 93].

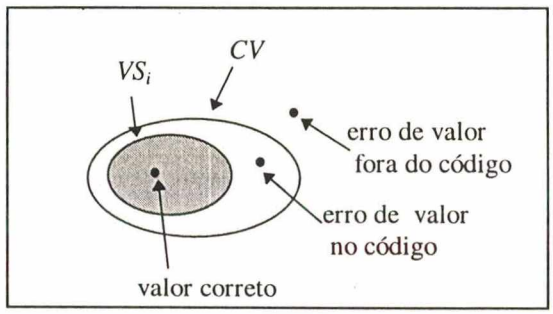


Figura 2.2: Representação de erros de valores

2.3.3 Faltas no domínio do tempo

As faltas de desempenho ou de temporização se caracterizam por ocorrências de serviços percebidas fora do tempo especificado. Estas faltas podem dar origem a:

- ***falhas por antecipação*** (*early timing failure*) em que a ocorrência de serviço o_i é percebida antes do tempo especificado:

$$(E_i \neq \Phi) \wedge (t_i \notin TS_i) \wedge (t_i < t_{min}(i)) \qquad para\ o_i = < v_i, t_i >$$

- ***falhas por atraso*** (*early timing failure*) em que o_i ocorre após o tempo especificado:

$$(E_i \neq \Phi) \wedge (t_i \notin TS_i) \wedge (t_i > t_{max}(i)) \qquad para\ o_i = < v_i, t_i >$$

Um caso particular das faltas de desempenho é aquele em que a ocorrência do serviço o_i nunca é recebida. As faltas relacionadas com este tipo de falha são identificadas como **faltas de omissão**:

$$(E_i \neq \Phi) \wedge (t_i \notin TS_i) \wedge (t_i \rightarrow \infty)$$

Um serviço está em ***crash*** se deixa de responder sistematicamente às solicitações de serviço. Desta forma, o comportamento de um serviço com a semântica de falha de *crash* é dado por:

$$\forall\, i\,.\ ((E_i \neq \Phi) \wedge (t_i \notin TS_i) \wedge (t_i \rightarrow \infty)) \wedge (\forall j\,.\ ((j > i) \wedge (E_j \neq \Phi) \wedge (t_j \notin TS_j) \wedge (t_j \rightarrow \infty)))$$

A figura 2.3 ilustra as faltas no domínio do tempo com suas subdivisões.

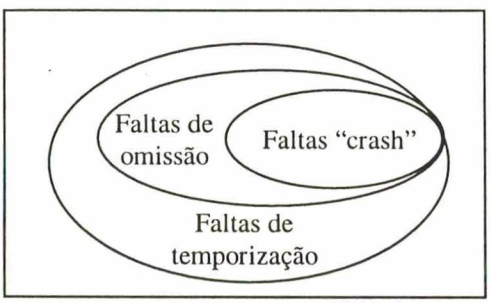


Figura 2.3: Faltas no domínio do tempo

2.3.4 Faltas arbitrárias

A semântica de **falhas arbitrárias** envolve falhas nos domínios de valor e do tempo. A representação do comportamento arbitrário para uma ocorrência de serviço o_i, é dada por:

$$(E_i \neq \Phi) \wedge ((t_i \notin TS_i) \wedge (v_i \notin VS_i)) \vee ((t_i \in TS_i) \wedge (v_i \in VS_i) \wedge (o_i \notin E_i))$$

Algumas classes de **faltas arbitrárias** têm merecidas atenção especial na literatura [Lamport 82, Powell 91, Tully 90b, Veríssimo 89, Bondavalli 90, Edwards 94]:

- ***faltas maliciosas:*** o comportamento malicioso de componentes faltosos corresponde ao que se pode ter de mais intrigante e portanto indesejável em um serviço. Para um serviço $o_i = < v_i, t_i >$ a semântica de falha maliciosa é dada por:

  $$(E_i \neq \Phi) \wedge (v_i \in VS_i) \wedge (t_i \in TS_i) \wedge (o_i \notin E_i)$$

  Desta forma, a simples comparação de valores de serviço e o seu instante de ocorrência com os conjuntos VS_i e TS_i, respectivamente, não é determinante para identificar se a ocorrência de serviço é correta ou não, uma vez que podemos ter $E_i \subseteq VS_i \times TS_i$. Para se confirmar a correção de o_i é necessário que $< v_i, t_i >$ mantenha as propriedades dos elementos de E_i ($o_i \in E_i$).

- ***faltas de improvisação:*** um caso particular de comportamento arbitrário é quando o sistema gera espontaneamente ocorrências de serviços que não são esperadas:

$$(E_i = \Phi) \wedge (v_i \in CV) \wedge (t_i \in T)$$

onde os valores de serviço v_i são erros dentro do código (CV) e t_i é um instante qualquer no eixo do tempo $(T = \cup TS_k)$;

Estas faltas denominadas de faltas de improvisação em [Powell 91] e de **ocorrências não esperadas** em [Edwards 94] são incluídas por outros autores no grupo de faltas maliciosas[1].

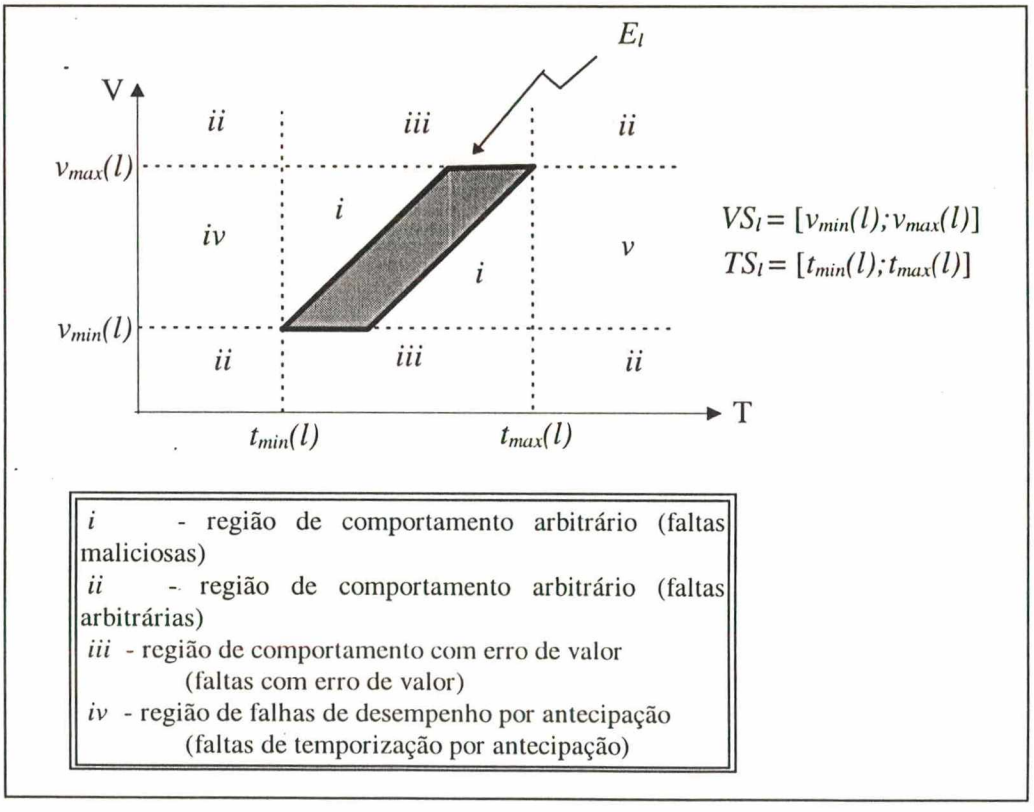


Figura 2.4: Regiões de falhas

A figura 2.4 resume para um conjunto expectativa E_l, as regiões dos diferentes comportamentos de elementos faltosos. Na figura 2.4, as falhas de omissão e de *crash* não

---

[1] Muitos autores também consideram faltas por valor dentro do código (item 2.3.2) como faltas maliciosas. Nós mantemos a separação na nossa classificação a exemplo de [Little 91] e [Barborak 93], porque acreditamos que isto simplifica e viabiliza um melhor entendimento de faltas com semânticas de falha maliciosa.

são representadas porque são casos especiais de falhas de desempenho por atraso. As falhas decorrentes de faltas de improvisação não aparecem na figura porque a premissa para suas ocorrências é que E_l seja um conjunto vazio.

2.4 Tolerância a Faltas

A tolerância a faltas é a propriedade do sistema fornecer, por redundância, o serviço de acordo com a especificação desejada, mesmo na presença de faltas. Estas redundâncias podem ser de software, hardware ou ainda temporal. A tolerância a faltas envolve várias fases. Na literatura, normalmente, são identificadas as fases de detecção de erros, confinamento de erros, processamento de erros e tratamento de faltas.

No que se refere ao processamento de erros, as técnicas envolvidas se dividem em **compensação do erro** e **recuperação do erro** [Anderson 81, Laprie 92]:

- **compensação de erro:** envolve a redundância de informação ou de processamento para mascarar os efeitos de elementos faltosos eventuais. As técnicas de compensação de erro são baseadas na replicação ativa (item 3.2.1), aplicadas sistematicamente em um sistema;

- **recuperação de erro:** esta técnica está baseada na **detecção de erro**. A recuperação de erro consiste em substituir um estado errôneo por um estado sem erro. Duas formas de recuperação de erro são identificadas na literatura:

  * **recuperação em retrocesso do erro:** esta técnica exige que informações de estado do sistema sejam regularmente armazenados em pontos específicos, denominados de **pontos de recuperação**. A partir da detecção de um erro, o estado do sistema é restaurado com os valores do último ponto de recuperação estabelecido;

  * **recuperação em avanço do erro:** está técnica consiste em transformar o estado errôneo em um novo estado livre de erros, a partir do qual o sistema retorna a sua operação normal, possivelmente em modo degradado.

Ao contrário das técnicas de compensação de erro, que independentemente da presença ou não de erro, apresentam sempre o mesmo *overhead* de processamento, as técnicas de recuperação de erro apresentam acréscimos de processamento somente quando um erro é detectado no sistema.

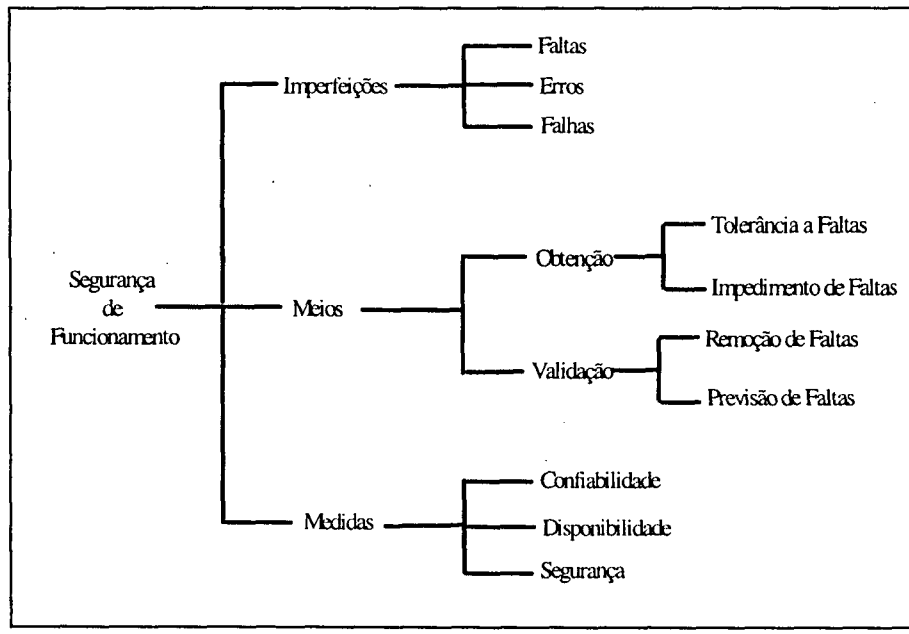


Figura 2.5: Taxionomia de Segurança de Funcionamento

A fase de **tratamento de faltas** é responsável pelos procedimentos que visam impedir a reativação de uma falta. Desta forma, após o processamento de erro, é necessário identificar os elementos faltosos, de modo a recuperá-los ou retirá-los do sistema. A primeira etapa do tratamento de faltas é então a **diagnose de faltas** que consiste em determinar a localização e a natureza das faltas. A segunda etapa, denominada de **passivação de faltas**, consiste em prevenir que as faltas sejam novamente ativadas. Após a passivação da falta, a reconfiguração do sistema somente é necessária quando os serviços fornecidos não atendem mais aos requisitos mínimos de tolerância a faltas. As diversas fases da tolerância a faltas e seus mecanismos associados são examinadas em várias publicações [Anderson 81, Laprie 92, Lemos 88].

A tolerância a faltas é apenas um dos meios utilizados na concepção de sistemas com atributos de segurança de funcionamento. Outros meios, tais como a **prevenção a faltas** e a **remoção de faltas**, bem como a taxionomia completa de segurança de funcionamento são abordados

em detalhes em [Anderson 81, Laprie 92]. A figura 2.5 apresenta a arborescência que resume a taxionomia de segurança de funcionamento.

2.5 Esquemas de tolerância a faltas

Existem na literatura, atualmente várias propostas de esquemas para a tolerância a faltas de software e/ou hardware. Estes esquemas integram componentes redundantes (de hardware e/ou software), no sentido de manter a continuidade de serviços implementados a partir destes componentes, mesmo em presença de elementos faltosos.

2.5.1 Tolerância a faltas de hardware

A tolerância a faltas de hardware é conseguida, basicamente, na replicação de componentes de hardware. Neste texto, nós limitamos o entendimento de componente de hardware ao de um **elemento de processamento**. Um elemento de processamento (processador, memória, E/S, etc.) suporta as operações básicas de hardware, necessárias à execução de um algoritmo de aplicação.

- **Esquemas Duplex**

Esquemas usando a execução redundante de um programa em dois elementos de processamento permitem a detecção de uma falta de hardware. A comparação dos resultados dos componentes não mascara o elemento faltoso, porém possibilita que se monte estratégias de estações ou nós de processamento com comportamento de falha controlada. A estação composta por dois elementos, no desacordo dos resultados num processo de comparação, para de funcionar *(fail-silent)*. Na bibliografia o uso deste esquema *duplex* é bastante difundido [Shirivastava 92, Brasileiro 95].

Outro esquema duplex bastante difundido é o das estações PSP (*pair of self-checking processing units*) [Kim 92b, Nelson 90]. Neste caso, uma estação é estruturada usando dois módulos auto-testáveis. Estes módulos auto-testáveis incorporam mecanismos de detecção de erros, permitindo que nos esquemas *duplex* formados a partir dos mesmos se obtenha a operação contínua na presença de uma falta de hardware. As estações PSP funcionam em modo *primary/shadow*. A figura 2.6(a) ilustra o esquema PSP onde dois módulos auto-testáveis idênticos (A e B) são mostrados formando uma estação PSP. Supondo o módulo A

no papel de primário, A e B devem então processar as mesmas entradas sendo que o módulo A é o responsável pelo envio dos resultados. O módulo primário pode ser substituído pelo reserva (B) no envio dos resultados quando da sinalização de falha do mesmo (falha de A).

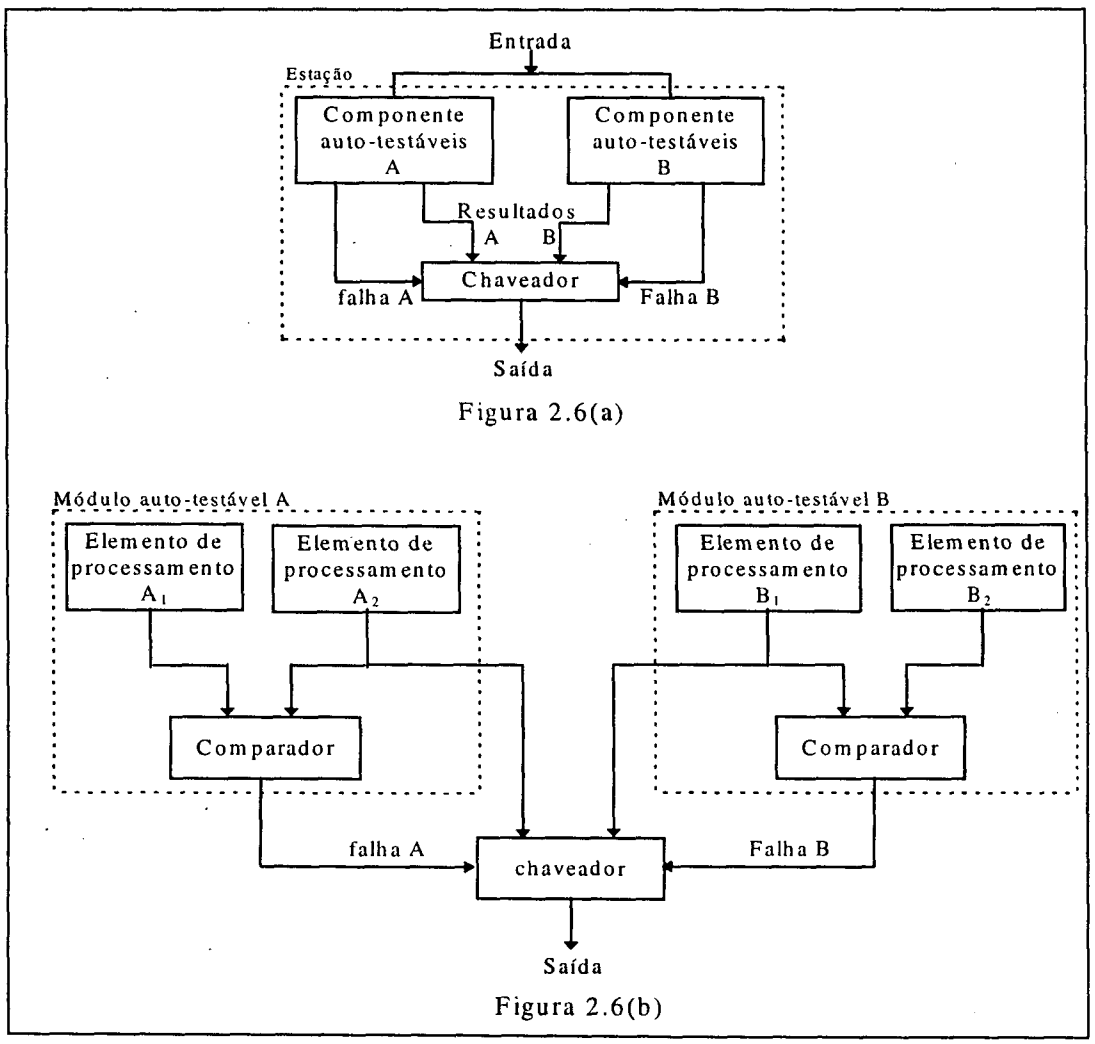


Figura 2.6: Esquema duplex com módulos auto-testáveis

A implementação destes módulos auto-testáveis e das funções de detecção de erros está fundamentada em replicações de hardware. Ou seja, cada módulo auto-testável é implementado usando dois elementos de processamento. A comparação entre os resultados dos elementos de um módulo gera a sinalização de falha do módulo. Esta comparação pode ser feita tanto por software como por hardware. A figura 2.6(b) mostra duas comparações usadas nas detecções de erros dos módulos auto-testáveis de uma estação PSP. O sistema Stratus usa um esquema semelhante ao da figura 2.6(b) [Webber 91].

- **Técnica NMR *(N-Modular Redundancy)***

O esquema NMR *(N-Modular Redundancy)* [Wensley 78] é tradicionalmente reconhecido como um esquema para tolerância a faltas de hardware. Este esquema é fundamentado na replicação *(N)* dos componentes de hardware. Os *N* elementos de processamento idênticos têm seus resultados submetidos a um processo de comparação, identificado como votação [Nelson 90, Laprie 92]. A estratégia de votação normalmente utilizada no esquema NMR é a **votação majoritária**, isto é, a maioria absoluta *(N/2 + 1)* resultados iguais define o valor da saída correta do esquema. Desta forma, o esquema NMR com votação majoritária mascara a presença de até *(N/2 -1)* componentes faltosos.

O TMR (*Triple Modular Redundancy*) é um caso particular de esquema NMR, onde o número de réplicas é três. Vários sistemas conhecidos, como por exemplo os sistemas SIFT [Wensley 78] e FTMP [Rennels 84], utilizam o esquema TMR. O sistema TMR com votação majoritária permite a tolerância a uma falta de hardware.

2.5.2 Tolerância a faltas de software

A tolerância a faltas de software também está baseada no uso de componentes redundantes que no caso são software. Normalmente, estes componentes redundantes (ou alternativas) são integrados com alguns mecanismos de detecção de erros numa mesma estrutura (ou esquema). Esta integração oferece algumas vantagens, como o confinamento de erros sendo delimitado aos limites destas estruturas. Estas estruturas podem ser facilmente incorporadas, como extensões, em linguagens de programação [Randell 94, Bondavalli 95, Fraga 91].

Diversidade de projeto

Na tolerância a faltas de software, os componentes mesmo apresentando a mesma funcionalidade, não são cópias idênticas. O desenvolvimento de componentes redundantes de software com a mesma funcionalidade, porém diferentes, está baseado no **princípio da diversidade de projeto** [Avizienis 84]. A diversidade de projeto pressupõe a possibilidade de desenvolver vários programas (ou algoritmos) a partir de uma mesma especificação, de maneira a [Lyu 91][Kelly 91]:

- reduzir os efeitos de omissões, equívocos e inconsistências gerados durante o processo de desenvolvimento de uma aplicação;
- minimizar a probabilidade de ocorrências de faltas relacionadas no processo de desenvolvimento dos vários componentes (alternativas).

A diversidade de projeto pode ser conseguida de maneira forçada ou aleatória [Lyu 91]. A maneira aleatória é alcançada, na construção dos componentes de software, usando diferentes equipes de desenvolvimento, uma para cada componente. A diversidade forçada, por outro lado, procura encontrar diferentes soluções (algoritmos, ferramentas, etc.) para se implementar as diferentes versões de uma mesma especificação de um software.

Os dois principais esquemas de tolerância a faltas de software baseados na diversidade de projeto são: **Bloco de Recuperação e Programação *N*-Versões**.

- **Esquema de Bloco de Recuperação [Randell 75]**

Um Bloco de Recuperação (RB) é composto de um algoritmo principal, uma ou mais alternativas (algoritmos alternativos) e de um teste de aceitação. A escolha do algoritmo principal e a ordenação das alternativas são feitas segundo critérios pré-estabelecidos, por exemplo, grau de precisão, desempenho, etc. Quando o bloco é ativado, o algoritmo principal é executado e o resultado produzido é submetido ao teste de aceitação. Caso este resultado não passe pelo teste, como técnica de recuperação em retrocesso (*backward*), o estado anterior à execução do algoritmo principal é restaurado, e a primeira alternativa na ordenação de algoritmos é então executada e seus resultados também testados. Este processo de testes, recuperações e ativações de algoritmos se dá até o ponto em que uma das alternativas produza um resultado que passe pelo teste de aceitação, ou então que não exista mais alternativas disponíveis. Neste último caso, uma exceção é gerada, devendo ser tratada nos níveis superiores. A figura 2.7 ilustra a estrutura do esquema RB com k algoritmos de aplicação.

Por diferentes razões, o teste de aceitação constitui-se no aspecto mais crítico do esquema de RB. Em alguns casos, o teste de aceitação deve verificar a completa correção dos resultados fornecidos, porém este tipo de testes, tão abrangente, nem sempre pode ser implementado. Em decorrência de não existir uma metodologia geral para a elaboração de testes genéricos, os

testes de aceitação são sempre construídos relacionados com os algoritmos de aplicação [Anderson 81].

O número máximo de faltas de software toleradas no esquema de Bloco de Recuperação corresponde ao número de alternativas menos 1, ou seja, no exemplo da figura 2.7 a tolerância de k-1 faltas.

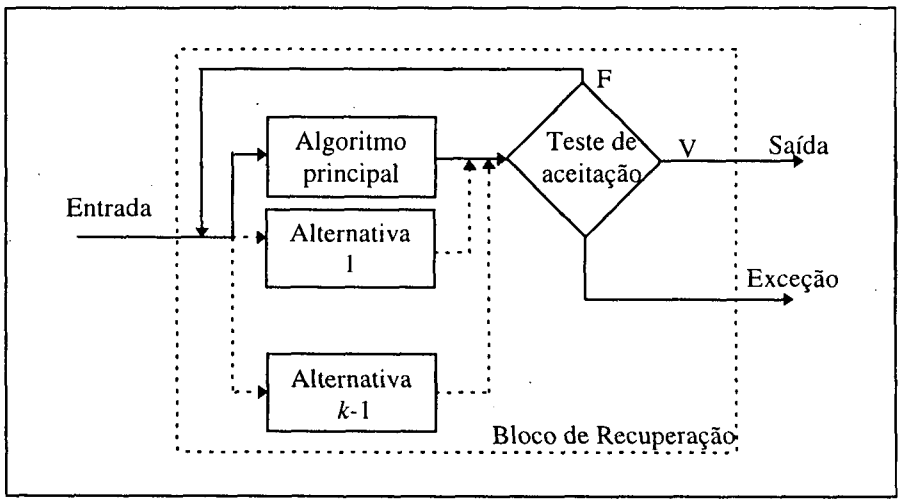


Figura 2.7: Estrutura básica do esquema de Bloco de Recuperação

- **Esquema de Programação *N*-Versões [Avizienis 77]**

A técnica de Programação N-Versões (PNV) consiste na construção independente de N ($N \geq 3$) algoritmos (versões) de um mesmo programa com a mesma funcionalidade, desenvolvidos a partir de uma mesma especificação. Quando um serviço é solicitado as N versões são executadas paralelamente e os resultados produzidos são enviados ao *ajustador* [Giadomenico 90]. O ajustador é o mecanismo responsável pela obtenção de um valor de consenso a partir dos resultados obtidos pelas N versões. Em algumas aplicações o ajustador se resume a comparação bit-a-bit dos N resultados obtidos. Neste caso, o algoritmo implementado pela função de ajuste é denominado de **votação exata**. Entretanto, nem sempre esta estratégia de detecção de erros é possível, já que em muitas situações os resultados de réplicas, embora corretos, produzem valores diferentes. Este é o caso de aplicações numéricas onde as aproximações podem levar a resultados diferentes, porém dentro de uma precisão aceitável. Os algoritmos de ajuste implementado para estas situações formam o que é identificado na literatura como **votação inexata** [Shin 89, Blough 90, Giadomenico 90].

Neste caso, a saída pode pertencer ao conjunto dos *N* valores de resultado ou então ser uma composição (valor médio, mediano, etc.) destes valores.

A obtenção da saída é o aspecto mais complexo do esquema de PNV, pois não existe um procedimento de ajuste genérico que pode ser empregado em qualquer situação e independente da aplicação [Giandomenico 90, Shin 89, Blough 90]. Se a partir dos resultados fornecidos não é possível de obter uma saída, uma exceção é gerada pelo ajustador, devendo ser tratada nos níveis superiores. A figura 2.8 ilustra um componente construído a partir do esquema de PNV.

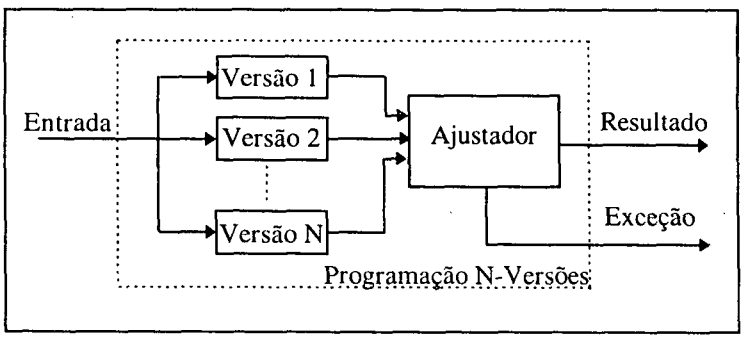


Figura 2.8: Esquema de Programação N-Versões

Diversidade de dados

Em alguns casos, as faltas residuais que permanecem num software somente são ativadas quando o sistema é submetido a entradas particulares, sendo que a simples alteração destas entradas adequadamente, sem que se perca propriedades lógicas, permite que o mesmo software não mais apresente o comportamento errôneo quando novamente executado [Jalote 94]. Esta constatação deu origem ao princípio da **diversidade de dados** [Ammann 87] que tenta reexpressar os dados de entrada sem que se percam as propriedades lógicas.

Nos esquemas que empregam a diversidade de dados, a um componente de software está associado um conjunto de dados de entrada relacionados. O procedimento que gera este conjunto de dados relacionados é chamado de reexpressão de dados. Os dados obtidos a partir destes algoritmos de reexpressão são logicamente equivalentes.

Em geral, os algoritmos de reexpressão de dados dependem da aplicação e exigem uma análise cuidadosa da especificação do serviço como forma de determinar as transformações válidas para gerar os dados equivalentes [Jalote 94]. Além disto, a reexpressão de dados pode não ser possível para algumas aplicações.

Os principais esquemas que empregam a diversidade de dados são [Ammann 87]: o *Retry Block* e o *N-Copy*.

- **Esquema *Retry Block* [Ammann 87]**

A semântica do esquema *Retry Block* é semelhante ao do Bloco de Recuperação, exceto que ao invés de utilizar vários algoritmos alternativos, o esquema possui um único algoritmo de aplicação. Na primeira execução, o algoritmo de aplicação é processado, usando o dado de entrada. Caso o resultado obtido não passe pelo teste de aceitação, o estado inicial do bloco é restaurado e o algoritmo é novamente executado após a reexpressão do dado de entrada. Este procedimento é repetido até que um resultado passe pelo teste de aceitação ou que o *deadline* especificado para o esquema se esgote. A funcionalidade do *Retry Block* é mostrada na figura 2.9.

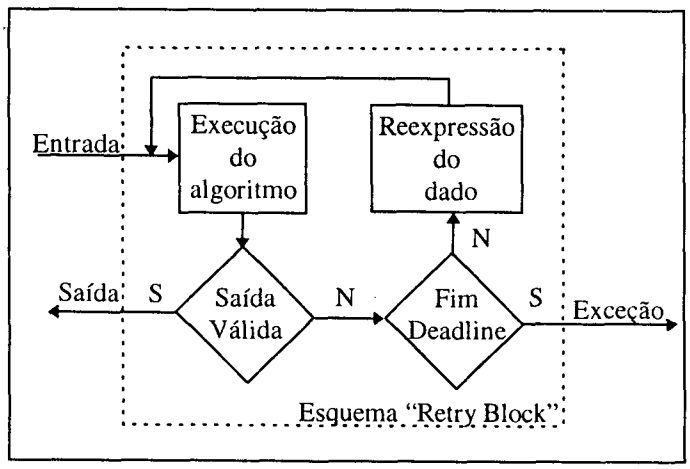


Figura 2.9: Diagrama funcional do esquema *Retry Block*

- **Esquema *N-Copy* [Ammann 87]**

No esquema *N-Copy*, as *N* cópias idênticas de um programa são executadas em paralelo. Entretanto, ao contrário da Programação *N*-Versões em que as *N* versões de um mesmo

programa são executadas para o mesmo dado de entrada, neste esquema as cópias dos programas são executadas para dados de entrada diferentes, sendo que cada um destes dados são reexpressões de um mesmo dado de entrada original. A seguir, os resultados gerados pelas cópias são submetidos a um procedimento de ajuste, responsável pelo mascaramento de erro. Da mesma forma que na Programação *N*-Versões o ajustador é o fator crítico e deve ser implementado a partir de um algoritmo de votação inexata. A funcionalidade do esquema de *programação N-Copy* é mostrada na figura 2.10.

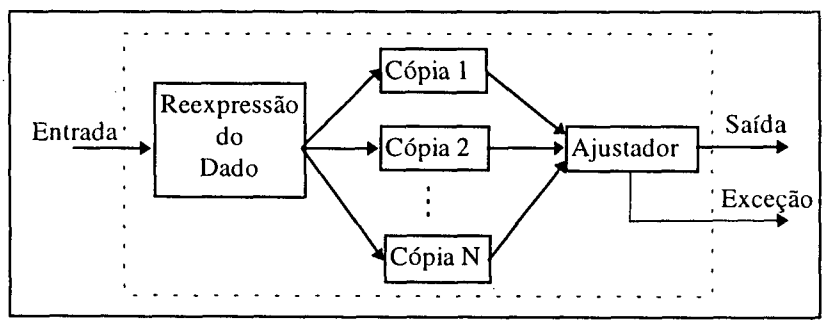


Figura 2.10: Esquema *N-Copy*

2.5.3 Outros esquemas

Na literatura são citados um grande número de outros esquemas. De uma maneira geral estes esquemas combinam propriedades dos anteriores, no sentido de aumentar as faixas de faltas toleradas ou detectadas. Neste texto, nós nos limitamos a seguir a descrever alguns destes esquemas que reputamos como significativos.

- **Esquema DRB [Kim 88]**

O esquema DRB ou estação DRB (*Distributed Recovery Block*) consiste em um esquema duplex, formado por dois elementos de processamento executando em cada um deles um Bloco de Recuperação (RB). Os dois elementos de processamento executam no modo primário/reserva e seus RBs são constituídos de duas alternativas (dois algoritmos de aplicação).

Do ponto de vista estrutural, os blocos de recuperação executados nos dois elementos de processamento são idênticos, isto é, apresentam os mesmos algoritmos alternativos e o mesmo teste de aceitação. A distinção ocorre em tempo de execução, quando os dois blocos têm

como algoritmo principal alternativas diferentes, ou seja, se A e B são duas alternativas de algoritmos de aplicação, um dos elementos executa o algoritmo A enquanto o outro elemento de processamento executa B. Ambos os elementos recebem os dados de entrada e após a execução de suas alternativas, os resultados são submetidos ao teste de aceitação. Se a saída (resultado) produzida no elemento de processamento primário, através de sua alternativa principal passa pelo teste, o resultado é enviado como resposta do serviço. Em caso contrário, o controle é transferido para o elemento secundário que passa a ser o responsável pelo envio do resultado. Se a saída do secundário também não passar pelo teste de aceitação um sinal de exceção é produzido para níveis superiores. A estrutura do DRB pode variar entre ciclos de processamento quando da ocorrência de faltas no elemento primário, ou seja, a função de primário é desempenhada pelo antigo secundário, com o elemento faltoso (primário do ciclo anterior) passando a ser o secundário após a reconfiguração.

Em termos de desempenho podemos afirmar que o DRB não envolve o retrocesso no caso de ocorrência de uma falta de software. Porém, o preço desta agilidade é a dependência de todo esquema em relação ao teste de aceitação. Este esquema pode tolerar duas faltas (uma de software e uma de hardware).

- **Esquema de *N*-Componentes Auto-Testáveis [Laprie 87]**

O esquema *N*-Componentes Auto-Testáveis (*N Self-Checking Programiming* - NSCP) é composto de *N* módulos incorporando mecanismos de detecção de erros. Cada módulo é estruturado a partir de dois elementos de processamento e um comparador. Os elementos de processamento executam dois algoritmos distintos de aplicação (versões). Na proposta apresentada, $2*N$ componentes de software são executados paralelamente nos *N* módulos que compõem o esquema. Entre estes *N* módulos auto-testáveis somente um é o ativo e os outros são *spares*. Os resultados fornecidos pelas versões do módulo ativo são comparados e caso sejam *iguais* (a noção de igualdade depende da função de ajuste), a execução do componente é considerada correta e os resultados obtidos pelas versões servem de base para o cálculo da saída. Por outro lado, se os resultados não coincidem, o controle é chaveado para o próximo módulo *spare* que envia uma saída, caso seja possível obtê-la, ou passa o controle adiante. Esta sequência prossegue até que um dos módulos envie uma saída ou então se esgote o

número de *spare*. Em [Laprie 90] é descrito e analisado uma série de combinações de 2*N versões configuradas em módulos auto-testáveis.

- **Esquema PNV-TB (PNV com *Tie-Breaker*) [Tai 93b]**

O esquema PNV-TB é uma variante do esquema PNV clássico que incorpora uma estratégia de sincronização na função de ajuste. Na implementação, por exemplo, do esquema PNV-TB com três versões o ajustador tenta primeiramente obter a saída do esquema a partir dos dois primeiros resultados gerados pelas versões. Se a tentativa de comparação é bem sucedida a saída do esquema é obtida, sem a necessidade do terceiro resultado. Entretanto, se a saída não pode ser obtida através dos dois valores, a função de ajuste é novamente executada considerando agora os três resultados, tomando então este esquema a funcionalidade do esquema PNV clássico. O esquema PNV-TB apresenta um desempenho melhor do que o PNV quando a freqüência de saídas a partir de dois resultados de versões é alta. Se esta freqüência for baixa o desempenho é pior do que o do PNV pois a função de ajuste é aplicada duas vezes no PNV-TB (detecção de erros em duas fases: a primeira com dois resultados sem sucesso e a segunda com três resultados).

- **Esquema de Bloco de Recuperação em Consenso [Scott 87]**

O esquema de Bloco de Recuperação em Consenso (CRB) procura minimizar os pontos críticos dos esquemas PNV e RB, que são as dependências dos mesmos em relação aos mecanismos de detecção de erros, respectivamente, o ajustador e o teste de aceitação. O CRB utiliza estes dois mecanismos de detecção. Sendo assim, a estrutura do esquema CRB consiste de N alternativas, um teste de aceitação e um ajustador. Quando da chamada do serviço, todas as versões são executadas paralelamente e os resultados são submetidos ao ajustador. Se o ajustador consegue obter o resultado de consenso, o funcionamento do CRB é equivalente ao do esquema PNV. Se no processo de ajuste o valor de consenso não é obtido, os resultados fornecidos são submetidos individualmente, da mesma forma que no Bloco de Recuperação, a um teste de aceitação. No CRB, como no RB, as versões (alternativas) são ordenadas segundo critérios pré-estabelecidos. Os testes são executados sobre os resultados segundo esta ordem de versão. O primeiro valor a passar pelo teste será a saída do esquema. A implementação proposta em [Scott 87] simplifica a construção do ajustador, assumindo que

não existem faltas relacionadas entre as versões (não ocorrem falhas de modo comum). Desta forma, a existência de apenas dois resultados iguais já determina a saída do bloco. A função de ajuste que implementa este tipo de votação (escolha de uma saída a partir de dois resultados iguais) é denominada de votação *2-out-of-N*.

- **Esquema PNV-TA [Tai 93b]**

O esquema PNV-TA (PNV com Teste de Aceitação) é uma variante do esquema CRB em que mesmo quando a função de ajuste obtém uma saída, o teste de aceitação é executado para verificar se esta saída é correta ou não. Esta disposição do teste de aceitação sendo executado sempre, elimina a possibilidade de saídas incorretas obtidas através do ajustador alimentado por dois ou mais resultados errôneos de versões (faltas relacionadas, por exemplo). No PNV-TA somente as saídas do ajustador que passam pelo teste de aceitação são enviadas, caso contrário uma exceção é gerada.

- **Esquema SCOP [Bondavalli 93]**

O esquema SCOP (*Self Configuring Optimal Programming*) tenta eliminar a rigidez e a não flexibilidade dos esquemas anteriores. Neste esquema a execução dos algoritmos estão organizadas em fases de execução, sendo que subconjuntos das versões disponíveis a serem executados em cada uma das fases são selecionados dinamicamente. Esta seleção dinâmica tem por objetivo flexibilizar o esquema, permitindo que uma saída aceitável seja obtida a partir dos recursos disponíveis no momento. A seleção pode ser parametrizada em relação ao nível de tolerância a faltas, quantidade de recursos disponíveis e tempo de resposta desejável.

O funcionamento do esquema envolve, na primeira fase, a execução de um subconjunto das alternativas (versões) disponíveis, com número suficiente para produzir uma saída se nenhum erro ocorrer. Na ocorrência de erro, uma nova fase de execução é iniciada onde versões adicionais são executadas. Os resultados obtidos são novamente submetidos aos mecanismos de detecção de erro (ajustador) e então, se mais versões são necessárias devido a não obtenção do valor de saída, uma outra fase de execução é iniciada. Este processo vai se desenvolvendo até que se obtenha um valor de saída ou que as versões disponíveis se esgotem.

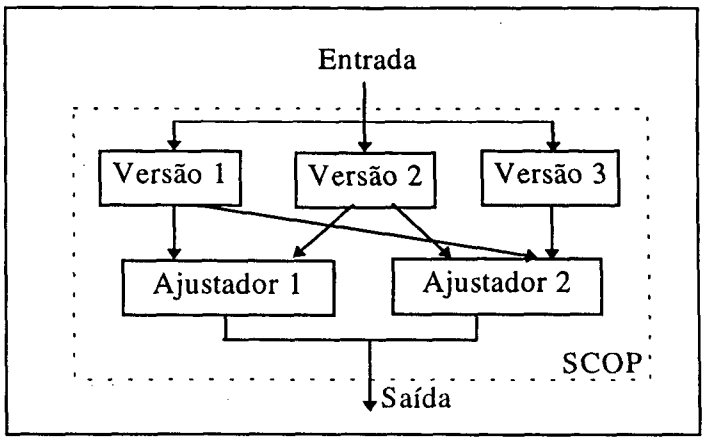


Figura 2.11: Esquema SCOP

A figura 2.11 ilustra uma exemplo de implementação do esquema SCOP com 3 versões. Na primeira fase, duas versões (1 e 2) são executadas e os resultados são enviados ao ajustador (1). Se a saída do esquema pode ser obtida através destes resultados a execução do esquema termina e a terceira versão não é executada. Caso contrário, a terceira versão é executada e os três resultados são enviados ao ajustador (2) para se obter a saída do esquema (segunda fase). Desta forma, a estrutura de funcionamento do esquema é semelhante ao do esquema PNV-TB. Entretanto, a principal contribuição deste esquema está na proposta de uma estratégia dinâmica na seleção dos conjuntos de versões que são executadas em cada interação.

2.5.4 Considerações Finais

A maioria dos esquemas atuais apresenta uma estrutura fixa com escalonamento estático de versões (algoritmos), com um grau de replicação (número de componentes) constante, alguns dependentes da estrutura de hardware ou se executando em hardware dedicado. Nos últimos anos algumas propostas de esquemas de tolerância a faltas com algum tipo de flexibilidade têm sido apresentadas na literatura [Bondavalli 93, Tai 94]. Estes esquemas visam tornar a tolerância a faltas adaptável às condições de carga e às restrições temporais do sistema:

- O esquema SCOP seleciona dinamicamente as versões que serão executadas no atendimento do serviço. Esta seleção considera as condições de carga do sistema de maneira obter o melhor desempenho do sistema contanto que as saídas sejam aceitáveis;

- Em [Tai 94] é proposto um sistema que implementa serviços com base nos esquemas RB e DRB. No instante de execução do serviço é feita a seleção de qual dos esquemas (RB ou DRB) será executado no atendimento do serviço. O algoritmo de seleção considera o número de elementos de processamento livres no sistema, no instante do pedido de serviço. Se existe capacidade suficiente de processamento paralelo o serviço é executado na forma DRB. Se as necessidades de processamento paralelo do DRB não estão disponíveis, então o serviço é executado na forma de um RB.

Embora vários outros aspectos possam ser usados na comparação entre estes esquemas, neste texto, nós estudamos o desempenho e a segurança de funcionamento a partir da performabilidade [Meyer 80] de alguns destes esquemas (capítulo 5). Uma simulação numérica destes esquemas foi feita no sentido de medir a eficiência dos mecanismos e técnicas de redundância definidos nos mesmos [Nacamura 94].

Muitos dos esquemas apresentados neste capítulo, embora não citado têm a tolerância a faltas de hardware implementada através da execução dos algoritmos alternativos em elementos de processamento distintos de um sistema distribuído ou de um sistema multiprocessador. A ênfase normalmente dada é para faltas de software na literatura de origem.

Neste estudo foram deixadas de fora as técnicas de **objetos persistentes**, **memória estável**, etc., usadas normalmente em sistemas transacionais. Isto porque limitamos o escopo de nosso trabalho ao uso de processamento replicado e grupo de processos, conforme será visto nos próximos capítulos. Estas técnicas não são adaptadas para o contexto citado.

2.6 Conclusão

Neste capítulo foram apresentados os principais conceitos e a taxionomia básica da disciplina segurança de funcionamento. Examinamos a classificação de faltas segundo as semânticas de falhas associadas.

Na seqüência, os principais esquemas de tolerância a faltas presentes na literatura foram descritos. Os esquemas, principalmente, os propostos na década de oitenta apresentam estruturas pouco flexíveis, e um comportamento que se mantém sempre o mesmo. As exigências de aplicações atuais, envolvendo principalmente sistemas distribuídos, têm

incentivado o surgimento de novas propostas de esquemas com características dinâmicas, adaptáveis à evolução de carga e de desempenho tão presentes nestas aplicações. Existe um campo vasto que acreditamos não explorado de esquemas flexíveis onde suas estruturas seriam adaptáveis às semânticas envolvidas na aplicação.

CAPÍTULO 3

TÉCNICAS DE REPLICAÇÃO EM SISTEMAS DISTRIBUÍDOS

Nos últimos anos vários sistemas distribuídos têm utilizado técnicas de replicação de dados e de processos para aumentar a disponibilidade e o desempenho dos serviços fornecidos e para permitir, através da incorporação de mecanismos de tolerância a faltas, que estes serviços sejam fornecidos mesmo quando da ocorrência de falhas no sistema. Neste trabalho, nós estamos particularmente interessados no uso de técnicas de replicação com fins de tolerância a faltas.

Inicialmente neste capítulo é apresentada a abstração de **grupo** que vem se difundindo como uma forma natural de explorar a replicação em sistemas distribuídos. Os diferentes tipos de grupo e os protocolos de comunicação de grupo também são discutidos.

Os tipos de técnicas de replicação usadas em sistemas distribuídos são discutidos na seqüência do capítulo. O conceito de **determinismo de réplicas** é introduzido considerando técnicas de réplicas ativas e semi-ativas. Os controles necessários para a implementação do determinismo de réplicas nas replicações citadas são discutidos também neste texto. Nos itens subseqüentes são apresentados os modelos mais representativos que fazem uso de replicações para a tolerância a faltas em sistemas distribuídos.

3.1 Processamento em grupo

Nos últimos anos, a noção de **grupo** vem se difundido como uma forma natural de explorar modelos de replicação em sistemas distribuídos. Um grupo é fundamentado em uma associação em que seus membros apresentam uma relação abstrata comum, semânticas de aplicação comuns e/ou políticas internas comuns [Lea 94]. A noção de grupo está

diretamente associada a **disseminação de mensagens** entre os membros do grupo. Esta disseminação é realizada usando o que normalmente é identificado como mecanismos de **comunicação de grupo** (ver item 3.1.2).

As motivações para a inclusão do conceito de processamento de grupo nos modelos de programação distribuída são muitas. Processamento em grupo pode ser usado no sentido de dar suporte a aplicações como trabalho cooperativo (*groupware*), facilitando as interações nestas aplicações. Grupo pode ser usado em sistemas distribuídos para aumentar a disponibilidade de recursos compartilhados ou ainda no processamento replicado por razões de tolerância a faltas.

Várias classificações de tipos de grupo têm sido apresentadas na literatura [Birman 93, Kaashoek 92, Liang 90]. Estamos particularmente interessados na classificação apresentada em [Liang 90] que se baseia em aspectos estruturais, permitindo que se tenha a idéia da abrangência do conceito de grupo.

3.1.1 Tipos de grupo

Um grupo G é composto pelos processos P_1, P_2, P_3,..., P_n que implementam o serviço S a partir de um conjunto de operações $O=[o_1, o_2, ...,o_k]$ que manipulam por sua vez, um conjunto de dados $D=[d_1, d_2,...d_m]$. Admitindo que um processo P_i tenha uma estrutura formada por:

O_i : conjunto de todas as operações de P_i;
D_i : conjunto de todos os dados de P_i.

Os grupos então podem ser classificados de acordo com a estrutura de seus processos como se segue [Liang 90]:

- **Grupo DOH** (***Data and Operation Homogeneous***)**:** todos os processos do grupo compartilham na execução do serviço S dos mesmos dados e operações, ou seja:

$$\forall i \,.\, (P_i \in G) \wedge (O_i \cap O = O) \wedge (D_i \cap D = D)$$

os grupos DOH são usados principalmente para aumentar a confiabilidade e a disponibilidade de serviços. Grupos no estilo *peer-member scheme* fazem parte desta classe [Birman 93][Kaashoek 92].

- **Grupo OHO** (***Operation Homogeneous Only***)**:** neste tipo de grupo o espaço de dados é particionado entre os membros do grupo. Cada membro porém suporta o mesmo conjunto de operações:

$$\forall\, i,j.\ i \neq j\ \ (P_i \in G) \wedge\ (O_i \cap O = O) \wedge (D_i \cap D_j = \Phi)\ \wedge\ \ (\cup D_i = D)$$

este grupo é usado principalmente para distribuir a carga entre seus membros. Este é o caso de alguns *servidores de nome* projetados de modo que o espaço global de nomes é particionado e cada servidor de nome mantém a sua partição. O servidor de nomes do sistema Clouds [Dasgupta 91] segue este modelo. Um grupo de servidores de impressão que atendem os pedidos conforme suas disponibilidades faz também parte dos grupos OHO.

- **Grupo DHO** (***Data Homogeneous Only***)**:** os membros nestes grupos compartilham os dados; as operações são diferentes:

$$\forall\, i,j.\ i \neq j\ ((P_i \in G) \wedge\ (D_i \cap D = D) \wedge (O_i \cap O_j = \Phi))\ \wedge\ (\cup O_i = O)$$

estes grupos são chamados de grupo funcionalmente distribuído em [ANSA 90]. Em uma requisição de cliente, os membros deste tipo de grupo agem diferentemente. O uso corrente destes grupos são para aplicações cooperativas. A técnica de replicação coordenador/coordenados (ver item 3.3.3) para tolerância a faltas também faz parte desta classe de grupo.

- **Grupo Het** (***Heterogeneous***)**:** neste caso, tanto os dados como as operações podem ser heterogêneos:

$$\forall\, i,j.\ i \neq j\ \ ((P_i \in G) \wedge (D_i \cap D_j = \Phi) \wedge (O_i \cap O_j = \Phi)) \wedge (\cup O_i = O)\ \wedge\ (\cup D_i = D)$$

neste tipo de grupo pode haver ou não a cooperação entre os membros. A associação entre seus membros é principalmente no sentido de facilitar as interações entre o cliente e os membros do grupo. Sistema de conferência, grupos *News* e plantas industriais podem fazer uso de grupos heterogêneos.

Neste trabalho, nós nos concentramos em grupos DOH e DHO que são as classes mais apropriadas para suporte na construção de servidores tolerantes a faltas.

3.1.2 Comunicação de grupo

A disseminação de mensagens entre os componentes de um grupo pode se dar em diferentes formas. Na literatura os diferentes algoritmos que possibilitam a comunicação de grupo são normalmente identificados por **protocolos de difusão** (*broadcast* ou *multicast*). Em alguns dos tipos de grupos, citados no item anterior, é possível o uso de **difusão não confiável**, onde na disseminação não existe garantia que todos os processos do grupo recebam a mensagem enviada.

Nós estamos interessados em grupos como suporte para a tolerância a faltas em sistemas distribuídos e neste caso, difusões não confiáveis não são suportes adequados devido as inconsistências que poderiam ser geradas nos diferentes membros do grupo. O suporte desejado neste caso é uma coleção de protocolos com propriedades bem definidas, identificados como **algoritmos de difusão confiável** (*reliable broadcast*).

3.1.2.1 Difusão confiável

Um protocolo de difusão confiável suporta um serviço de envio de mensagens onde é possível garantir, mesmo em presença de falhas, um comportamento bem definido em relação às mensagens difundidas no grupo. Para isto, a condição de protocolo de difusão confiável está fundamentada em três propriedades básicas [Cristian 85, Powell 91]: acordo, ordenação e terminação. Uma mensagem recebida em uma difusão só é considerada aceita em um participante do grupo se as propriedades de difusão confiável se verificam para esta mensagem. O ato de **aceitação** é chamado na literatura de engajamento da mensagem (*message commit*).

Conforme o maior ou menor rigor das propriedades de acordo, ordenação e terminação são identificadas diferentes categorias de protocolos de difusão confiável.

Acordo

A propriedade de acordo (*agreement*) deve garantir mesmo em presença de falhas, que ou todos os participantes corretos engajam a mensagem difundida ou todos eles desconsideram

esta mensagem. Porém, o espectro de faltas, como visto no capítulo anterior, é bem amplo: o que implica que quanto mais restritivas forem as hipóteses de faltas, menos complexo e de menor custo são os algoritmos que atendem a propriedade de acordo.

Assim, enquanto os protocolos de [Birman 87a] suportam, por exemplo, apenas faltas de *crash*, os protocolos de [Chang 84, Luan 90, Melliar-Smith 90] garantem a propriedade de acordo na presença de faltas de omissão. Em [Cristian 85] são propostas versões de protocolo de difusão confiável que garantem, separadamente, o acordo em presença de faltas de omissão, de temporização e arbitrárias com mensagens autenticadas. Protocolos que atendem hipóteses mais gerais de faltas são mais complexos; este é o caso do protocolo de consistência interativa (*interactive consistency algorithm*) [Lamport 82] onde o acordo é conseguido em presença de faltas arbitrárias.

Em outro sentido, alguns protocolos têm sido propostos com o objetivo de relaxar a propriedade de acordo. Para isto foram definidas algumas semânticas de comunicação de grupo menos restritivas [Powell 91]:

- ***at least K*:** qualquer mensagem entregue a um participante deve ser entrega a no mínimo *K* participantes corretos;
- ***best effort K*:** a ausência de falhas, qualquer mensagem entregue a um participante, deve ser entregue a *K* participantes.

Estas semânticas enfraquecem as necessidades de acordo para o engajamento de uma mensagem.

Ordenação

A propriedade de ordenação garante que todas as mensagens difundidas em um grupo estarão sujeitas a uma ordem de engajamento. Em [Shrivastava 92] são identificados os tipos de ordem **sem ordenação, FIFO, causal e ordem total**. Com base nestes tipos de ordenação podemos classificar os protocolos de difusão confiável em:

- **Protocolos de difusão sem ordenação**

Nesta categoria se encontram protocolos cujas mensagens difundidas são engajadas sem nenhum critério de ordenação. A Figura 3.1 mostra a difusão das mensagens m_1 e m_2 sem ordenação.

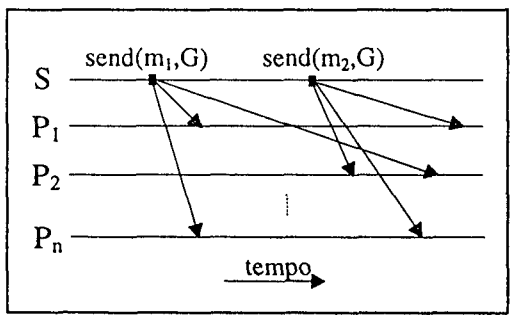


Figura 3.1: Difusão não ordenada e não confiável

- **Protocolos de difusão FIFO**

Nesta classe as mensagens difundidas por um mesmo processo são engajadas pelos participantes do grupo na mesma ordem de emissão (ordem FIFO). Mensagens difundidas por processos diferentes podem ser recebidas em membros distintos do grupo em ordens opostas. A Figura 3.2 mostra o caso em que dois emissores (S_1 e S_2) difundem mensagens em um mesmo grupo G e todos os processos em G recebem m_1 antes de m_2.

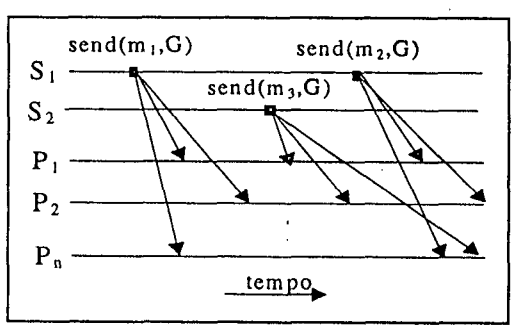


Figura 3.2: Protocolo de difusão FIFO

- **Protocolo de difusão causal**

Protocolos desta classe estendem a ordenação do tipo anterior para a **causalidade potencial** no envio de diferentes emissores. Na Figura 3.3, os emissores S_1 e P_1 difundem,

respectivamente, as mensagens m_1 e m_3. A recepção de m_1 em P_1 pode ser a **causa potencial** da emissão de m_3. Então, a ordem causal deve garantir que todos os processos participantes do grupo engajem m_1 antes de m_3. Outras mensagens que não apresentem relação de causa potencial podem ser engajadas em qualquer ordem; este é o caso de m_2 e m_4.

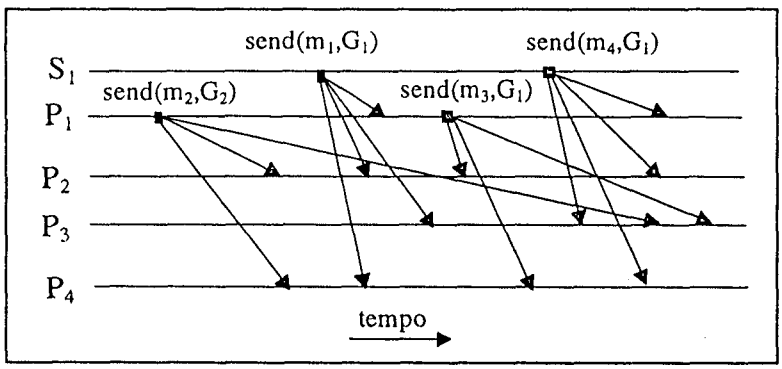


Figura 3.3: Protocolo de Difusão Causal

- **Protocolo de difusão com ordem total**

  Neste caso, todas as mensagens difundidas são engajadas numa mesma ordem por todos os participantes do grupo. A ordem em que todos os participantes devem engajar as mensagens, envolve mensagens de diferentes emissores e não necessariamente corresponde à ordem cronológica de emissão das mensagens. Na literatura, os protocolos desta classe são identificados como **protocolos de difusão atômica**. A Figura 3.4 ilustra os protocolos de difusão atômica.

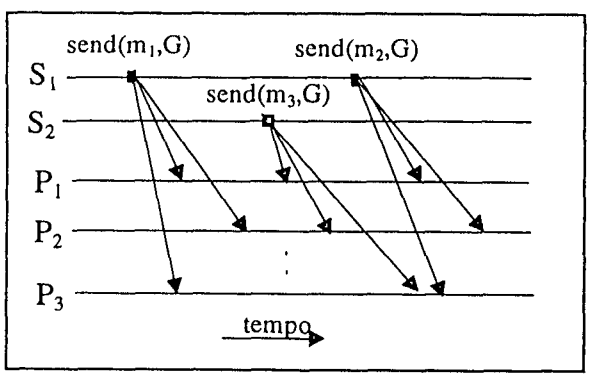


Figura 3.4: Protocolo de difusão atômica

A implementação de mecanismos de ordenação pode se dar diferentemente nos protocolos de difusão confiável:

- **Ordenação baseada no histórico das mensagens:** o princípio básico desta abordagem é que quando um processo envia uma mensagem, junto com a mensagem existem informações sobre o histórico das mensagens recebidas anteriormente. Este histórico permite que os receptores eliminem inconsistências de ordem. Este tipo de protocolo garante uma ordenação **causal** e eventualmente **ordenação total**. Este é o mecanismo usado no protocolo CBCAST do sistemas ISIS [Birman 87a];

- **Ordenação centralizada:** neste caso, a ordenação total é estabelecida por um processo centralizado. Porém, neste caso, é necessário que o protocolo utilize mecanismos que permitam a detecção da falha do processo que impõe a ordem. O protocolo proposto em [Chang 84] é um exemplo deste tipo de implementação;

- **Commit multi-fases:** estes protocolos fornecem uma ordenação total, através do uso de algoritmos multi-fases (normalmente, dois ou três *rounds* de troca de mensagens). O emissor envia as mensagens e recebe informações que permitem ao mesmo emissor impor uma ordenação única a todos os receptores. A mensagem é considerada entregue somente se todas as fases do protocolo são completadas. O protocolo ABCAST do sistema ISIS [Birman 87a] é exemplo desta classe;

- **Baseada em relógios:** estes protocolos fazem parte de uma classe importante de algoritmos multi-fases, e assume a existência de uma base de tempo global. Com base neste tempo global, os *timestamps* das mensagens são usados para impor uma ordenação total. Tais protocolos podem fornecer uma latência de comunicação constante, tendo o atrativo de que se um emissor difunde uma mensagem no instante T, então existe a garantia de que todos os receptores corretos recebem a mensagem no instante $T+\Delta$, onde Δ é a latência do protocolo;

Terminação

A propriedade de terminação está ligada a que cada participante do grupo tenha engajado a mensagem difundida. Normalmente, o tempo máximo para o engajamento de uma mensagem é denominado de latência do protocolo de difusão.

Em [Veríssimo 89] são introduzidos alguns parâmetros que servem para caracterizar a terminação de um protocolo. Seja:

- ***Tempo de execução*** *(Te)* é o intervalo entre o pedido de difusão de uma mensagem é o *delivery* no último membro do grupo;
- ***Tempo de Inconsistência*** *(Ti)* é o maior intervalo de tempo entre dois engajamentos (*delivery*) de uma mensagem por dois membros distintos do grupo em uma execução de protocolo.

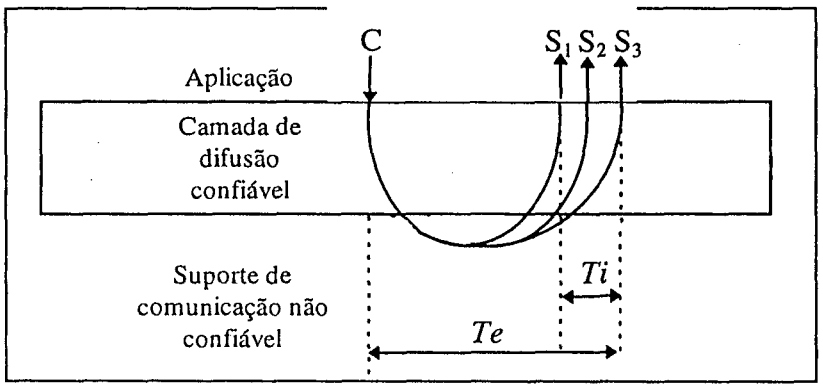


Figura 3.5: Tempo de Execução e de Inconsistência

A Figura 3.5 ilustra os tempos de execução e de inconsistência. Fazendo uso de *Te* e *Ti* em [Veríssimo 89] são introduzidas duas medidas: a estabilidade σ de um protocolo, definida como a máxima diferença entre duas execuções quaisquer do protocolo, e a rigidez τ do protocolo como sendo o máximo tempo de inconsistência:

$$|Te_j - Te_l| \leq \sigma \quad \text{e} \quad |Ti| \leq \tau \quad \text{para } \sigma > 0 \text{ e } \tau > 0$$

Com estas duas medidas fica fácil estabelecer critérios para se classificar protocolos segundo seus graus de sincronismo. Um protocolo é tanto mais síncrono quanto menores forem seus τ e σ. Os protocolos que se aproximam da rigidez e estabilidade completa, isto é, τ e σ desprezíveis em relação a *Te* são ditos com **sincronismo forte**. É o caso, por exemplo, do protocolo de [Cristian 85].

No sentido oposto, apresentam-se os protocolos assíncronos. Em situações onde os protocolos apresentam os limites *(bounds)* σ e τ, mas que não são desprezíveis em relação a valores de *Te* estes protocolos se caracterizam como **assíncronos temporizados** (***timed***

asynchronous) [Cristian 95] ***ou parcialmente síncronos*** [Dwork 88]). Nesta categoria podemos colocar protocolos como os de [Chang 84] e [Birman 87a].

Em condições de assincronismo máximo, a estabilidade e a rigidez não são delimitadas. Neste caso, a única exigência razoável que se pode fazer é que o protocolo termine, ou seja, é necessário que se tenha uma noção probabilista de limite máximo da terminação do protocolo. Para isto, é estipulado um Te_{max}, tal que:

$$\Pr\{Te > Te_{max}\} \cong 0$$

A classificação apresentada de protocolos de difusão, tomando como base as propriedades de acordo, ordenação e terminação conforme apresentada neste item, não é totalmente difundida na literatura. Outros autores apresentam classificações que se diferenciam das apresentadas[1]. Nós mantemos esta classificação por acreditá-la mais concisa e de acordo com os conceitos apresentados neste texto.

3.2 Replicação em sistemas distribuídos

Nós estamos interessados no uso de grupo como base na implementação de modelos replicação, no sentido da tolerância a faltas em sistemas distribuídos. Se considerarmos as

---

[1] Um exemplo destas classificações é a apresentada em [Hadzilacos 93] onde as propriedades de acordo, ordem e terminação não são entendidas ou usadas como no nosso texto. Na classificação citada a propriedade de acordo é encerrada em protocolos identificados como difusão confiável. Os outros protocolos são considerados como extensão deste pelo acréscimo de ordens específicas:

- Protocolos de difusão confiável: nestes protocolos todas as mensagens difundidas ou são engajadas em todos os processos corretos ou não são em nenhum destes processos;
- Protocolos de difusão FIFO: são protocolos de difusão confiável, onde é adicionado a ordem FIFO de mensagens;
- Protocolos de difusão causal: protocolos de difusão confiável acrescentados de ordenação causal.

Nesta classificação a ordem causal inclui a ordem FIFO, e a ordem total é uma propriedade independente. Com base nisto são definidos:

- Protocolos de difusão atômica: são protocolos de difusão confiável que garante ordem total, sem garantir a ordem FIFO e a ordem causal nas mensagens engajadas;
- Protocolos de difusão atômica FIFO: é uma extensão do anterior que além da ordem total garante ordenação FIFO das mensagens engajadas;
- Protocolos de difusão atômica causal: mantém propriedades de ordenação total e de ordenação causal entre mensagens engajadas.

Em nossa classificação assumimos a ordem total incluindo as outras ordens mais fracas (FIFO e causal), tomando como base a nossa experiência com protocolos de difusão atômica conhecidos da literatura.

Na classificação de [Hadzilacos 93], a propriedade de terminação não é considerada. Esta propriedade é na verdade reduzida a outra que os autores chamam de ***Δ-timeliness*** que simplesmente garante a existência de um *bound* em um protocolo. Nestas condições, na literatura citada, somente os protocolos síncronos são identificados como satisfazendo a propriedade de ***Δ-timeliness***.

interações seguindo o modelo cliente/servidor, um grupo pode implementar um serviço específico, escondendo detalhes operacionais de seus usuários. O serviço seria visível a seus usuários através de suas especificações, ou seja, como um conjunto de operações que podem ser ativadas pela passagem do tempo ou por eventos de entrada ativados pelos seus usuários.

Um grupo é então um conjunto de servidores replicados, fornecendo a nível de interface de serviço a transparência da replicação. Esta característica favorece mecanismos, por exemplo, como o mascaramento de erro: a saída do grupo é função das saídas das réplicas membros do grupo (por exemplo, o voto majoritário) e neste caso, falhas de membros são transparentes aos usuários do serviço replicado.

Desta forma, explorar a replicação a partir da abstração de grupo de processos passa a ser um problema de gerenciar as interações do cliente com o grupo, sem se preocupar com as interações individuais de cada membro do grupo. Além disto, a falha de um processo particular do grupo é tratada como uma questão separada do fornecimento do serviço pelo grupo.

O uso de técnicas de replicação em sistemas distribuídos não é recente. Os modelos existentes exploraram a redundância inerente a estes sistemas. O uso de replicação está baseada em hipóteses de que faltas que ocorrem em sistemas distribuídos são independentes. Em parte isto é verificado nestes sistemas, onde nós não compartilham recursos exceto pelo suporte de comunicação [Poledna 94]. Os modelos de replicação tratam usualmente com faltas de hardware permanentes ou transitórias.

Na seqüência deste texto, introduzimos a conceituação usada e os principais exemplos de técnicas de replicação presentes na literatura.

3.2.1 Modelos de replicação

As técnicas de replicação são uma alternativa para que serviços continuem a ser fornecidos em sistemas distribuídos, mesmo em presença de nós com falhas. A unidade de replicação ou *réplica* é um componente de software (módulo, objetos, processo, etc.) que encapsula dados identificados, normalmente como **estado da réplica**. As réplicas são distribuídas entre diferentes nós da rede (elementos de processamento).

A **coordenação** em uma replicação define como as diferentes réplicas devem interferir no processamento no sentido de manter a consistência e a transparência do conjunto. As necessidades dos **protocolos de coordenação** ou de **controle de réplica** são distintos para os diferentes modelos de replicação. Estes protocolos podem também ser bastante simplificados conforme o suporte usado em suas implementações.

As técnicas de replicação variam pelo grau de sincronismo e pelos tipos de réplicas envolvidas. O aspecto de sincronismo será tratado no capítulo 6. Na literatura, normalmente, são identificados modelos de replicação passiva, ativa e semi-ativa [Powell 91, Poledna 94, Veríssimo 89]:

- **Replicação Ativa**

Nos modelos de replicação ativa, todas as réplicas recebem os pedidos de serviço (dados de entrada) processam de forma paralela e produzem as mesmas saídas quando isentas de faltas. A consistência dos estados das réplicas implica, nestes modelos também chamados de **Máquina de Estados** [Schneider 90], na necessidade do **determinismo de réplicas** que é obtido em condições de acordo e de mesma ordem sobre os pedidos de serviço (item 3.2.2).

Em relação a semântica de falhas, o modelo de réplicas ativas é o único onde o serviço sobrevive a falhas arbitrárias de seus componentes. Entretanto, os custos no sentido de manter o determinismo de réplicas e por conseqüência, a consistência entre réplicas, são grandes. As restrições ao uso de funções não deterministas são parte deste custo. A replicação ativa é essencialmente distribuída e simétrica.

- **Replicação Passiva**

Na **replicação passiva** existe a figura de uma réplica privilegiada que recebe os pedidos de serviço, processa os mesmos e envia os resultados ao cliente. A réplica privilegiada é a única ativa no modelo; as demais são passivas e têm seus estados atualizados periodicamente a partir da privilegiada, usando mecanismos de *checkpointing* (transferência de estado).

Caso a réplica privilegiada apresente comportamento falho, uma das réplicas passivas deve substituí-la. Uma destas réplicas reservas deve partir do último *checkpoint*, executando os pedidos de serviço posteriores ao *checkpoint* até chegar ao estado da antiga réplica

privilegiada livre de erro. Esta recuperação de estado envolve portanto retrocesso o que implica em limitações no seu uso em aplicações com restrições temporais fortes.

Este modelo é limitado a um espectro de faltas mais restritivo. O modelo não tolera o comportamento arbitrário da réplica privilegiada. A semântica de falhas das réplicas neste modelo está restrita a *crash*, omissão e falhas de desempenho.

- **Replicação Semi-Ativa**

Alguns autores identificam um terceiro grupo de replicações que é o das semi-ativas [Powell 91, Poledna 94]. A replicação semi-ativa apresenta características dos modelos anteriores: as réplicas são todas ativas e existe uma réplica privilegiada.

Neste modelo todas as réplicas executam os pedidos de serviço, porém como a coordenação é centralizada na replicação semi-ativa, é a réplica privilegiada a responsável pela imposição da ordem de execução dos pedidos e também pela resposta ao cliente. Na falha da réplica privilegiada, uma das réplicas restantes assume o seu papel. Neste modelo não ocorre a recuperação de estado baseada em retrocesso. Este modelo permite a execução de algumas funções não deterministas uma vez que as decisões são impostas pela réplica privilegiada (item 3.2.2).

O modelo em questão se limita a semântica de falhas de *crash*, omissão e de desempenho.

3.2.2 Determinismo de réplicas

O conceito de **determinismo de réplicas** foi introduzido em [Schneider 90] inicialmente como condição para que réplicas ativas se mantenham consistentes:

> ***Determinismo de réplicas [Schneider 90]:*** *"Uma replicação é determinista ou formada por réplicas deterministas se réplicas idênticas e não faltosas, partindo do mesmo estado inicial e processando as mesmas entradas, produzirem as mesmas saídas";*

As exigências para o determinismo de réplicas, inicialmente colocadas eram as necessidades de acordo e ordem sobre as entradas:

- **acordo:** todas as réplicas não faltosas recebem os mesmos pedidos de serviço;
- **ordenação:** todos os pedidos de serviço são executados na mesma ordem pelas réplicas não faltosas.

O conceito de determinismo de réplicas conforme apresentado é restrito somente a técnica de replicação ativa. Entretanto é interessante que se estenda este conceito para uma faixa mais ampla de estratégias de replicações. Por exemplo, em algumas variantes de replicação ativa, nem todas as réplicas produzem saídas, como o modelo de componentes *shadows* do MARS [Kopetz 89] (ver item 3.3.4). Em [Poledna 94] este conceito é revisto abrangendo estas variantes e também introduzindo a noção de tempo:

> ***Determinismo de réplicas [Poledna 94]:*** *"Réplicas corretas mostram a mesma correspondência de saídas e/ou de mudanças de estado quando partem do mesmo estado inicial, executando os mesmos pedidos de serviço em um dado intervalo de tempo";*

Esta definição é mais genérica que a anterior e não está restrita somente a replicação ativa. A replicação semi-ativa apresenta a dita correspondência de mudanças de estados entre a réplica privilegiada e as restantes. A idéia de *correspondência de estado* facilita o entendimento de um esquema do tipo Programação *N*-Versões com voto inexato como uma estratégia de replicação ativa. Obter as saídas do serviço em dados intervalos de tempo descreve o comportamento síncrono necessário para aplicações em tempo real.

Em [Poledna 94] é admitido que em replicações passivas, o determinismo de réplicas é também mantido. A base para isto é a correspondência de estados entre a réplica privilegiada e as réplicas passivas no instante após a realização de operações de *checkpointing*.

Neste texto, nós nos limitamos, provavelmente por conveniência ao determinismo de réplicas como condição para a consistência em replicações ativas e semi-ativas. É também esta a visão corrente na literatura em geral.

3.2.3 Fontes de não determinismo

Em [Tully 90b, Poledna 94, Liang 90] são examinadas possíveis fontes de não determinismo em estratégias de replicação. Abaixo, nós resumimos as principais fontes citadas nestas referências:

- **inconsistência de entradas:** se valores de entrada são apresentados para réplicas de maneira inconsistente, os estados das réplicas devem divergir no fim do processamento. Leituras de diferentes réplicas feitas de diferentes sensores analógicos para o controle de uma variável numa planta é um exemplo típico deste problema;

- **inconsistência de ordem:** se os pedidos de serviço submetidos as réplicas são consumidos pelas mesmas em diferentes ordens a consistência também é perdida;

- **inconsistência de informações de *membership*:** em replicações com o tratamento de faltas, a dinâmica de sair e voltar a fazer parte do servidor replicado por parte das réplicas, pode determinar em informações desatualizadas de *membership* de um servidor replicado em diferentes usuários do serviço;

- **construções não deterministas em linguagens:** muitas linguagens apresentam construções que são fontes de não determinismo. Estas construções como o *Select, Alternate*, etc. determinam escolhas arbitrárias de fluxos de controle em programas replicados gerando inconsistências de estado;

- **informações locais:** se decisões são para ser tomadas com base em eventos locais, estas informações podem não estar prontamente disponíveis em todas as réplicas do servidor, conduzindo à inconsistência. Estes são os casos, por exemplo, do acesso ao tempo em relógios locais, do tratamento de exceções por mecanismos de *timeout*, etc;

- **escalonamento dinâmico:** as decisões de escalonamento, tomadas dinamicamente principalmente em situações de carga dinâmica com as réplicas em processadores diferentes, são fontes de não determinismo difíceis de serem controladas.

Outros aspectos, como heterogeneidade de máquinas, poderiam também ser citados. Na verdade o determinismo de réplicas tem que ser implementado fazendo uso de técnicas de projeto e mecanismos de tempo de execução.

3.2.4 Controle interno e controle externo

As fontes de não determinismo citadas acima podem em parte ser tratadas usando protocolos que implementam a coordenação da estratégia de replicação. Estes protocolos constituem o **controle externo** na implementação do determinismo de réplicas. Por outro lado, uma boa parte destas fontes de não determinismo podem ser evitadas por construção; os meios usados neste caso são identificados como **controle interno** na obtenção do determinismo de réplicas [Poledna 94].

O **controle interno** está fundamentado então na preocupação de evitar o uso de construções de linguagens com semânticas não deterministas, de basear a tomada de decisões em informações globais, de evitar o uso de *timeouts* e decisões de escalonamento dinâmico, usar serviços de tempo baseados na sincronização de relógios físicos (levando em consideração o desvio máximo permitido), etc. Todas as decisões locais que podem levar ao não determinismo deixam de ser um problema de **controle interno** e passam a ser resolvidas usando protocolos de consenso (controle externo). Este é o caso, por exemplo, da leitura de sensores replicados em que a decisão do valor que representa a leitura passa por um problema de consenso (conjunto de acordos).

Os não determinismos causados pela comunicação, podem ser resolvidos com protocolos de difusão confiável. Neste caso, as necessidades de acordo e ordem das entradas são garantidas por estes protocolos de comunicação de grupo. As definições de ordem e acordo nestes protocolos simplificam também os problemas relacionados com a obtenção de informações de *membership*. O sistema ISIS [Birman 87a] faz uso destes protocolos para disseminar as *views* dos participantes atuais do grupo. A resolução de consensos também são facilitadas fazendo uso destes protocolos.

As propriedades de protocolos de difusão confiável e de *membership* quando usados como suporte para processamento replicado reduzem o não determinismo, porém não são suficientes para prevenir intervalos de inconsistência. Nestes protocolos, o *delivery* e a visão de *membership* podem provocar inconsistências entre as diferentes réplicas, em certos intervalos

de tempo, pois nem todos os protocolos de difusão confiável são síncronos e a assíncronia acarreta em períodos de inconsistência. O conceito de **sincronismo virtual** introduzido em [Birman 87b] relaxa a idéia de determinismo de réplicas, considerando as características de algoritmos assíncronos.

Em resumo, podemos afirmar que o uso de protocolos de difusão confiável como suporte de replicações ativas, simplifica as necessidades ou a realização do controle externo. E que a implantação do determinismo de réplicas é uma negociação entre as restrições funcionais de serviço (controle interno) e os custos de comunicação (controle externo) [Poledna 94].

3.2.5 Coordenação centralizada e distribuída

O grau de centralização é um ótimo parâmetro para classificar os protocolos de controle externo. A abordagem **centralizada ou assimétrica** define uma réplica privilegiada que impõe decisões e ordem sobre as demais réplicas, em outras palavras, impõe as condições para o determinismo de réplicas sobre as demais. Estratégias de replicação semi-ativa são exemplos desta abordagem centralizada.

A outra abordagem é a **distribuída ou simétrica** onde a figura do líder ou réplica privilegiada não existe. O determinismo de réplicas é alcançado através de consensos quando de decisões que poderiam causar o não determinismo.

	Técnicas de Replicação		
	Replicação passiva	**Replicação ativa**	**Replicação semi-ativa**
Processamento de erros	Recuperação em retrocesso	Compensação de erro	Recuperação em avanço
Semântica de falhas	*crash*/ desempenho	Sem restrições - falhas arbitrárias	*crash*/ desempenho
Tempo de recuperação	Alto	Baixo	Baixo/Médio
Acordo e ordem	Imposição da réplica privilegiada	Decisão distribuída	Imposição da réplica privilegiada
Réplicas deterministas	Em instantes precisos	Sim	Sim

Tabela 3.1: Principais características das técnicas de replicação

3.2.6 Síntese comparativa das estratégias de replicação

Nos itens anteriores foram introduzidas os tipos de técnicas de replicação diferenciados na literatura. Na seqüência foram vistos conceitos e estratégias no sentido de manter a consistência destas réplicas. A Tabela 3.1 faz um resumo deste estudo comparando as técnicas de replicação passiva, ativa e semi-ativa.

3.3 Modelos de replicação usados em sistemas distribuídos

Vários modelos de replicação têm sido introduzidos com o propósito de tolerância a faltas. No estudo que apresentamos abaixo, nós nos resumimos a alguns modelos significativos, presentes na literatura.

3.3.1 Modelo de Máquina de Estados [Schneider 90]

Este modelo é retomado aqui pela sua significação no entendimento de modelos de réplicas ativas. O modelo de Máquina de Estados tem sido usado como um *framework* ou mesmo como base conceitual para diversos outros trabalhos [Lim 92].

Máquina de Estados

O modelo de Máquina de Estados [Schneider 90] é uma abordagem geral para implementação de serviços replicados e da coordenação das interações entre clientes e as réplicas do servidor. Neste modelo, uma máquina de estado é constituída de **variáveis**, que representam seu estado, e por **comandos** que manipulam este estado. Os comandos apresentam duas propriedades: **atomicidade e determinismo**. A atomicidade garante a indivisibilidade dos comandos, ou seja, os comandos se executam completamente ou não na presença de faltas. O determinismo garante que uma máquina de estado, sob as mesmas condições de entrada e de estado produz sempre as mesmas saídas. Segundo esta abordagem, uma aplicação distribuída é estruturada em termos de clientes e servidores. Um serviço tolerante a faltas neste modelo é visto como um replicação de Máquinas de Estados (um conjunto de servidores idênticos) onde máquinas (réplicas) são executadas em processadores distintos do sistema.

O protocolo de **coordenação de réplicas** neste serviço replicado garante que todas as réplicas recebem e processam a mesma seqüência de pedidos. Desta forma, o determinismo de réplicas é mantido se a entrega dos pedidos satisfaz as seguintes propriedades:

- **acordo:** todas as réplicas não faltosas de uma máquina de estado recebem todos os pedidos;
- **ordenação:** todas as réplicas não faltosas de uma máquina de estado processam os pedidos recebidos em uma mesma ordem relativa.

A ordenação pode ser relaxada se os pedidos são comutativos (por exemplo, dos pedidos de leitura).

Dispositivo de saída tolerante a faltas

Se as saídas das máquinas são enviadas para um único dispositivo de saída, os efeitos de falha do mesmo determina a falha de todo o serviço replicado. No texto original, a solução indicada como usual consiste em replicar o dispositivo de saída (replicação do mecanismo de votação). Neste caso, cada votador combina as saídas das máquinas de estado, produzindo os **sinais** para o cliente. Ou seja, qualquer leitura dos **dispositivos** consiste na obtenção destes **sinais** dos dispositivos replicados

A abordagem também oferece soluções para o caso em que as repostas produzidas pelas máquinas de estado são dirigidas diretamente ao cliente. Neste caso, para o cliente a replicação da máquina de estado pode não ser transparente, exigindo que o cliente aguarde:

- ***t*+1 *respostas idênticas***, se as máquinas de estado apresentam semântica de falhas arbitrárias;
- ***uma única resposta***, se as máquinas de estado possuem semânticas de falhas de *crash* ou por omissão.

Tolerância a clientes faltosos

Somente a estruturação do serviço em Máquinas de Estados replicadas não é suficiente para garantir a tolerância a *t* faltas no sistema [Schneider 90]: clientes faltosos podem enviar pedidos fora de especificação às máquinas de estado; como conseqüência estas últimas podem produzir saídas errôneas, tendo seus estados comprometidos na possível seqüência de novos pedidos. Duas abordagens podem ser utilizadas para isolar a máquina de estado das faltas do cliente:

- **replicação do cliente:** a idéia básica desta abordagem consiste em replicar o cliente e transformar os múltiplos pedidos enviados pelo conjunto de réplicas à máquina de estado em um único pedido. Por exemplo, num sistema tolerante a *t faltas*, se *2t+1* pedidos diferentes são recebidos, cada um contendo o valor de um sensor, então um único pedido contendo o valor médio destes valores pode ser obtido e processado pelas máquinas de estado;

- **programação defensiva:** em algumas circunstâncias onde a replicação do cliente não é possível, ou pela indisponibilidade de processadores ou pela inexistência de um meio aceitável para transformar vários pedidos em um único, a introdução de testes nos pedidos ou comandos aparece como sendo uma solução razoável.

  A programação defensiva também pode consistir na especificação de tempos máximos de acesso aos recursos de uma máquina de estado (*timeouts*). Desta forma, é possível a uma máquina de estado distinguir clientes em falha que não dão continuidade aos seus processamentos. No caso de detecção de clientes faltosos, a máquina de estado pode abortar o processamento do pedido atual, liberando os recursos a outros clientes.

3.3.2 Modelo Líder/Seguidores [Powell 91]

O modelo Líder/Seguidores (*Leader/Followers*) [Powell 91] emprega a estratégia de replicação semi-ativa, onde a réplica líder assume o papel de réplica privilegiada. Na proposta original, os clientes interagem com o grupo enviando seus pedidos de serviço, usando primitivas de comunicação de grupo. Estas comunicações de grupo podem ser confiáveis (todas as réplicas recebem), porém sem o uso de mecanismos de ordenação total. Estas interações também poderiam se dar de outra forma, por exemplo, usando comunicações ponto a ponto entre o cliente e o líder.

A réplica líder, de posse dos pedidos, retransmite os mesmos às demais réplicas (seguidoras), usando um protocolo de difusão confiável que deve garantir no mínimo uma ordenação FIFO no envio das mensagens-pedidos. Com isto, todas as réplicas recebem as mensagens pedidos e seguem a ordem de execução estabelecida pelo líder, garantindo as condições para a consistência das réplicas.

Como uma estratégia de replicação semi-ativa, todas as réplicas executam os pedidos de serviço, porém somente a réplica líder envia os resultados aos clientes. Na literatura existem duas variantes do modelo. Na primeira, antes de enviar a resposta ao cliente, a réplica líder se bloqueia aguardando as confirmações de fim de serviço das réplicas seguidoras; esta abordagem é denominada de **bloqueante**. Na segunda abordagem, esta identificada como **não bloqueante**, a réplica líder envia a resposta ao cliente, imediatamente, no término da sua execução. Embora a abordagem não bloqueante apresente melhor desempenho, não é possível nesta abordagem a detecção de falhas de seguidores; são necessários mecanismos e protocolos adicionais para tal.

O modelo líder/seguidor no sistema Delta-4 [Powell 91] mantém o determinismo de réplicas mesmo em sistemas com escalonamento preemptivo. O sistema Delta 4 permite que o processamento devido a uma mensagem (pedido) seja interrompido por uma mensagem mais urgente. A solução encontrada para que o determinismo de réplicas fosse mantido foi introduzir a noção de **pontos de preempção** que são locais nos códigos das operações executadas pelas réplicas em que a preempção pode ocorrer. Quando a réplica líder atinge um ponto de preempção, a fila de mensagem é observada para verificar a existência de um pedido mais prioritário que o atual. A existência desta tal mensagem determina a preempção do código atual. O líder neste ponto deve informar as seguidoras que atingiram também o ponto de preempção e ficaram bloqueadas esperando as decisões de escalonamento do líder.

3.3.3 Modelo Coordenador/Coordenados [Birman 87a]

O modelo de coordenador/coordenados (*coordinator-cohorts*) [Birman 87a] é uma técnica de replicação passiva. Nessa técnica, o papel de coordenador (réplica privilegiada) não é fixo: a cada pedido de serviço submetido é atribuído um coordenador. A seleção do coordenador é feita dinamicamente em todas as réplicas, na chegada de um pedido, através de um algoritmo pré estabelecido. As proposições para este algoritmo de seleção ou tentam melhorar o desempenho do sistema escolhendo para coordenador o nó mais próximo do cliente ou distribuem aleatoriamente esta coordenação entre as réplicas do serviço.

O modelo permite também que vários pedidos de serviços sejam atendidos concorrentemente. Desta forma, é possível que exista vários coordenadores atendendo a pedidos de serviços distintos num mesmo instante.

Para que as várias réplicas (servidores) possam indistintamente assumir o papel de coordenador, é necessário algumas premissas no modelo:

1- Todos os clientes devem submeter seus pedidos usando o protocolo de difusão atômica ABCAST do ISIS; isto determina o recebimento dos pedidos segundo uma ordem total nos servidores replicados;

2- A escolha do novo coordenador a cada pedido, exige que todas as réplicas tenham o mesmo conhecimento do *membership* do grupo. O protocolo GBCAST do ISIS fornece esta visão única em todos os servidores.

No final da execução do pedido o coordenador, como um modelo de replicação passiva, deve enviar informações de *checkpoints* as demais réplicas (réplicas passivas). Os *checkpointing* são enviados usando o protocolo de ordenação causal CBCAST do ISIS. Como os servidores determinam seus papeis ora sendo coordenadores ou coordenados (passivos), o início de uma execução de um pedido pode ter sido provocado pela chegada de um *checkpoint*. Então, a réplica que executa este processamento deve transmitir junto com o seu *checkpoint* informações da precedência causal do *checkpoint* anterior em relação ao seu processamento. Todas as réplicas deverão manter esta ordem entre os dois processamentos. O protocolo CBCAST serve de suporte para estas ordenações causais entre *checkpoints*.

Este modelo tolera faltas de *crash* e a detecção de falhas, principalmente a dos coordenadores, é feita usando o sistema ISIS. No suporte ISIS existe mecanismos de monitoração dos nós que se refletem nos *views* de *memberships* enviados pelo GBCAST. A partir destes *views* são detectados as falhas de coordenadores e as recuperações correspondentes são então executadas seguindo a técnica de replicação passiva descrita no item 3.2.1.

3.3.4 Replicação no sistema MARS [Grüsteidl 89]

O MARS (*MAintainable Real-time System*) [Kopetz 89] é um sistema para aplicações de controle de processos industriais ou ainda em sistema embarcados. O modelo de replicação MARS é o único exemplo na literatura de replicação ativa com execução síncrona que conhecemos.

A tolerância a faltas é construída neste sistema através de módulos auto-testáveis. Um conjunto destes módulos auto-testáveis formam uma FTU (*Fault Tolerance Unit*). O serviço

fornecido por uma FTU é correto se pelo menos um destes nós auto-testáveis fornece o serviço. Um serviço então é fornecido por uma replicação de componentes de software que se executam nestes arranjos de módulos auto-testáveis.

A comunicação entre FTUs é feita através de difusões de mensagens em barramentos redundantes O controle de acesso usado nestes barramentos é feito usando a técnica de TDMA (acesso por divisão de tempo), onde a cada nó é atribuído um *slot* fixo para suas transmissões. Na verdade cada módulo auto-testáveis ativo de uma FTU constitui um nó nestes barramentos.

Além das informações de aplicação, um nó transmite informações de controle que permitem o reconhecimento de todas as mensagens difundidas em *slots* que o antecedem em um ciclo de transmissão[2]. Estas informações de controle permite que se monte um protocolo de *membership* [Kim 92],[Kopetz 91].

Embora uma FTU possa ser composta de vários nós *fail-silent* (módulos auto-testáveis), o número de réplicas com direito a acesso ao suporte de comunicação (com *slot* próprio) é limitado a dois por FTU. No MARS, os módulos de uma FTU com direito a acesso aos barramentos são chamados de ***ativos*** e os demais de ***shadows***. Os *shadows* recebem as mesmas mensagens dos ativos, executam os mesmos serviços, porém não transmitem mensagens (não possuem *slots*). As informações de *membership* são usadas para detectar em cada dois ciclos de processamento os nós ativos faltosos. Os nós faltosos serão substituídos pelos *shadows* de suas FTUs, assumindo seus *slots* nos barramentos.

O sistema MARS é um exemplo de sistema síncrono, as atividades de processamento e as transmissões de mensagens são todas ativadas por relógio *(time-triggered).* Em tempo de projeto são produzidos os escalonamentos e definidas as escalas para todas as atividades no sistema. A existência de uma base de tempo global (relógios físicos sincronizados) e dos escalonamentos produzidos *off-line* determinam a garantia do atendimento das restrições de tempo impostas na execução de cada atividade prevista no sistema.

---

[2] Um ciclo de transmissão em um protocolo TDMA é a distância entre dois *slots* consecutivos de um mesmo nó.

O determinismo de réplicas é mantido no MARS então pela execução síncrona dos módulos auto-testáveis. As características de redundância e a atribuição de *slots* por nó nas difusões garantem as propriedades de acordo e de ordenação.

3.3.5 Replicação Ativa Competitiva e Round-Robin [Chérèque 92]

O sistema DELTA-4 [Chérèque 92, Powell 91] suporta dois tipos de replicação ativa: a competitiva e a *round-robin* (cíclica). Nestas duas abordagens todas as réplicas processam os pedidos de serviço, porém apenas uma envia o resultado ao cliente. Tanto a abordagem cíclica quanto a competitiva apresentam cada réplica possuindo um controlador associado, responsável pela recepção, difusão e comparação de mensagens, ficando a réplica correspondente dedicada ao processamento das requisições. Para garantir a consistência entre as réplicas, todas as mensagens trocadas entre os controladores são transmitidas através de um protocolo de difusão atômica.

Os modelos podem ser configurados para tolerar dois conjuntos de falhas [Chérèque 92]: **falhas de temporização**, envolvendo as semânticas de falha de *crash*, omissão e temporização por atraso e **falhas não controladas** (arbitrárias), que compreendem todo o espectro de falhas.

Replicação ativa competitiva

Nesta replicação, a característica principal é a competição entre as réplicas: somente a mais rápida reponde a requisição. Na replicação competitiva, sob hipóteses de faltas de temporização, um cliente difunde uma requisição aos controladores que a repassam às suas réplicas associadas para o processamento replicado da mesma

O controlador ao receber a mensagem resposta de sua réplica associada verifica se já possui esta mensagem enviada por outro controlador do grupo. Se não possuir, o controlador concatena à mensagem resposta um identificador e difunde o resultado desta concatenação no grupo de controladores; se o controlador receber primeiro a mensagem que acabara de difundir, descobre que sua réplica foi a mais rápida e assim é o responsável pelo envio da resposta ao cliente. Caso contrário, sua mensagem é descartada. Este algoritmo garante que apenas uma réplica responde ao cliente, pois todas as mensagens difundidas no grupo são

observadas em cada controlador na mesma ordem relativa (garantido pela protocolo de difusão atômica).

Sob hipóteses de faltas de temporização, a replicação competitiva privilegia a réplica mais rápida, podendo com isto levar a uma desincronização muito grande no conjunto de réplicas. Entretanto, um grau aceitável de desincronização pode ser mantido se periodicamente um *rendez-vous* é realizado, com os controladores difundindo os resultados de suas réplicas entre si e o último a difundir envia o resultado ao cliente. Este *rendez-vous* é limitado no tempo (*timeout*) de maneira a detectar falhas nos controladores.

Com a hipóteses de faltas arbitrárias, a mensagem de resposta deve ser validada antes de ser enviada ao cliente. A validação de um resultado envolve a comparação, sendo que para se tolerar t falhas é necessário que $t+1$ resultados sejam iguais de no mínimo $2t+1$ réplicas. O controlador ao receber a resposta de sua réplica associada usa uma técnica de assinatura sobre a resposta e difunde a mensagem resultante a todos os controladores do grupo. O controlador no processo de comparação que verifica que a $t+1$ *ésima* mensagem correta recebida em difusão é a sua, é o responsável pelo envio da resposta.

Replicação ativa cíclica

Na replicação ativa *round-robin* os controladores de réplicas são configurados em anel lógico com um *token* associado. Ao atender um pedido, a réplica (controlador) que contém o *token* envia a resposta ao cliente e transfere o *token* para o próximo controlador do anel lógico que será o responsável pelo envio da resposta ao pedido seguinte.

Em hipóteses de faltas de temporização, na replicação cíclica ativa, o próprio mecanismo de controle de *token* é usado para detectar as falhas nos controladores. Entretanto, a estrutura em anel é pouco flexível para permitir que as réplicas possam se associar e deixar o grupo dinamicamente.

Com hipóteses de faltas arbitrárias, a replicação *round-robin* faz também uso de assinaturas; quando a resposta a um pedido de processamento é recebida pelo controlador possuidor do *token*, é criado uma mensagem *turn* que vai sendo passada no anel lógico, agregando mensagens assinaturas. Esta mensagem circula até que seja atingida $t+1$ respostas corretas.

Como a mensagem *turn* transfere o *token*, então o controlador que verificou que sua mensagem completou o limite exigido (t+1) é o que deve enviar os resultados ao cliente.

Os controladores e as réplicas associadas nas descrições apresentadas acima foram considerados com acoplamento forte, ou seja, uma falha de *crash* afeta a ambos. Na literatura geral era assumido um acoplamento fraco o quê determinava mecanismos nos controladores para detectar a falhas nas réplicas associadas.

3.3.6 Modelo de *tríades (triads)* [Ezhilchelvan 89]

O modelo de *tríades* foi proposto em [Ezhilchelvan 89] com o sentido de explorar a capacidade de processamento de um sistema distribuído ou um sistema multiprocessador na concepção de programas distribuídos tolerantes a faltas, usando a replicação de seus componentes. O modelo é de replicação ativa e o grau de replicação de cada componente é três, formando então as *tríades*. O modelo foi inicialmente desenvolvido em um rede de *transputer* explorando algumas características da rede de interconexão nestes sistemas multiprocessadores.

A versão replicada de uma aplicação distribuída neste modelo consiste na replicação de todos os componentes simples c_i formando as *tríades* C_i , tal que C_i é composto por três componentes $C_i = \{c_{i1}, c_{i2}$ e $c_{i3}\}$, onde c_{i1}, c_{i2} e c_{i3} são réplicas de c_i. *Estas réplicas são* executadas nos processadores p_{j1}, p_{j2} e p_{j3}, respectivamente, onde o conjunto $P_j = \{p_{j1}, p_{j2}, p_{j3}\}$ define uma tríade de processadores. Um processador é interconectado a outros processadores no sistema através de 4 ligações bidirecionais. O mapeamento réplica/processador apresenta as seguintes propriedades:

- qualquer processador não faltoso de uma *tríade* está diretamente conectado a todos os processadores não faltosos da mesma *tríade.*
- duas tríades podem estar ligadas diretamente ou não. Neste último caso as mensagens enviadas a *tríade* de destinação passa obrigatoriamente por processadores intermediários. De qualquer forma, sempre existe um caminho correto conectando duas *tríades* de processadores corretos.

- mascaramento de processadores faltosos de uma *tríade* é feito através de voto majoritário dos resultados obtidos por cada processo réplica da *tríade*. A votação é feita em cada

processador do sistema por onde passa o resultado, ou seja, não só as mensagens produzidas na tríade do processador, mas também as que trafegam pela tríade passam por votações intermediárias. Na implementação do voto, estas mensagens são enviadas aos processadores parceiros na *tríade* usando ligações bidirecionais. Quando um processador recebe duas cópias iguais de uma mesma mensagem a votação é bem sucedida e a mensagem é dirigida através de comunicação ponto-a-ponto para a tríade subsequente na rede de processadores. As comunicações entre *tríades* envolve sempre três mensagens, saídas de um processo triplicado de votação pertencente a tríade anterior. A integridade das mensagens que trafegam no sistema é garantida através de assinaturas (hipóteses de faltas arbitrárias). A mensagem é descartada quando o mecanismo de autenticação de assinaturas detecta a alteração da assinatura.

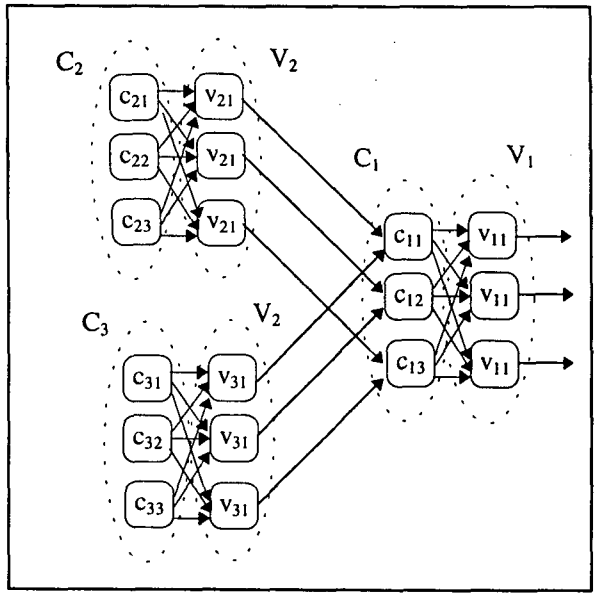


Figura 3.6: Conjunto de três *tríades*

A figura 3.6 ilustra um conjunto de três *tríades*. Na figura a comunicação das tríades C_2, e C_3, com C_1,, se dá concorrentemente envolvendo o envio das mensagens resultados dos votos V_2 (v_{21}, v_{22} e v_{23}) e V_3 (v_{31}, v_{32} e v_{33}), respectivamente. Se as mensagens saídas dos votos V_2 e V_3 se dão concomitantemente, então todas as réplicas da tríade C_1 (c_1, c_2, e c_3) devem selecionar a mesma mensagem para o consumo, mantendo o determinismo de réplicas. O problema de selecionar a mesma mensagem em todas as réplicas da *tríade* é resolvido através de um processo **ordenador** responsável por impor uma ordenação. A ordenação única de todas as mensagens destinadas as *tríades* é alcançada pelo processo ordenador através de um protocolo

de difusão atômica com tempo de terminação Δ. Cada ordenador nos processadores de uma *tríade* ao receber uma mensagem que tem por destino esta *tríade*, difunde a mensagem que chega aos ordenadores associados com a *tríade* na mesma ordem. As mensagens iguais são descartadas. Como os relógios são sincronizados, a mensagem difundida pelo ordenador na *tríade* no instante T de seu relógio local é recebida em todos os processos ordenadores da *tríade* no instante $T+\Delta$. A sincronização de relógio e o uso da difusão atômica síncrona garante a ordenação única das mensagens destinadas aos processos da *tríade*.

3.3.7 Replicação *Lazy* [Ladin 90]

A principio a replicação *Lazy* [Ladin 90] é uma estratégia semi-ativa. Este modelo foi definido para sistemas transacionais, projetado para se executar sobre o sistema Argus [Liskov 88]. A replicação *Lazy* está fundamentada sobre conceitos de objetos persistentes, memória estável e ações atômica. Embora não esteja dentro do contexto que nos fixamos inicialmente, a motivação em relação a este modelo está nas soluções diferentes das apresentadas anteriormente para a consistência de réplicas.

Os protocolos definidos neste modelo de replicação trabalham segundo a abordagem de ***consistência fraca*** que na verdade é uma negociação entre a consistência e a disponibilidade do serviço. A replicação *Lazy* e outros protocolos similares [Downing 90, Garcia-Molina 90, Alonso 90] exploram o conhecimento semântico da aplicação para relaxar a propriedade de ordenação das mensagens enviadas ao grupo, melhorando o desempenho da aplicação e forçando a ordenação estrita das operações somente quando for necessário.

Neste modelo um serviço é implementado por um conjunto de objetos (réplicas) distribuídos em diferentes processadores. O serviço fornece dois tipos de operações: *updates* que modificam o estado de uma réplica e *query* que lê variáveis do estado.

O cliente executa operações sobre uma réplica simples, mas o sistema garante que o serviço replicado permanecerá consistente, isto é, que todas as réplicas dentro do grupo eventualmente terão o mesmo estado. Os efeitos de pedidos que serão executados sobre uma réplica são propagados para outras réplicas através de mensagens chamadas de *gossips*.

O cliente pode impor algumas condições de ordem (ordem causal) entre as operações, transmitindo estas informações em um pedido aos servidores. Estas informações determinam quais operações devem ser executadas antes da operação pedida.

O sistema fornece uma ordenação total que tem como base os servidores. Esta ordenação se utiliza de um mecanismo de *timestamps*. A atribuição de *timestamps* é feita de forma centralizada usando um dos processos gerentes de réplicas (servidor) para tal.

O cliente ao apresentar um pedido de *update* traz consigo o *timestamp* do seu último acesso ao objeto. Caso o servidor contactado no seu controle de *timestamp* tenha um valor menor, isto indica que o objeto foi acerado anteriormente através de outro servidor e que o primeiro está com sua cópia do objeto inconsistente. Neste caso, o *update* deve ficar pendente até que os *updates* anteriores sejam recuperados pelo servidor. Todas as réplicas (servidores) mantém um *log* de operações de *updates* recebidos e que periodicamente são trocados na forma de informações enviadas em mensagens *gossips* entre as réplicas. Com este mecanismo as réplicas conseguem então atualizarem seus estados.

Este modelo é implementado sem o uso da noção de grupo ou de um suporte de comunicação de grupo na forma como tratamos até aqui neste capítulo. As soluções de coordenação são implementadas a nível de réplica.

3.3.8 Replicação no sistema Transis [Amir 95]

O uso da replicação no sistema Transis [Amir 95] tem como propósito principal também aumentar a disponibilidade de serviços de base de dados distribuída. O cliente solicita a execução de um serviço através de uma comunicação ponto a ponto ou de grupo. Os servidores replicados usam o suporte de comunicação TRANSIS, através de suas primitivas de comunicação de grupo para difundirem entre si os pedidos e obterem o acordo e a ordenação total dos pedidos concorrentes (protocolo de difusão atômica). O serviço uma vez executado tem os seus resultados enviados ao cliente através do servidor que inicialmente recebeu o pedido.

O suporte de comunicação de grupo no sistema Transis fornece os serviços de *memberships* e de difusão confiável de acordo com o modelo de **sincronização virtual estendida** [Amir 95]. As características do modelo de sincronização virtual estendida são as mesmas da

sincronização virtual proposta no sistema ISIS [Birman 87b], acrescentando hipóteses de particionamento do suporte de comunicação. Desta forma, como no sistema ISIS, todas as réplicas do grupo têm a mesma visão dos pedidos de operação e mudanças de configuração ocorridas no sistema., no mesmo instante lógico.

O sistema permite a continuidade dos serviços mesmo em situações de partição do suporte de comunicação. A estratégia escolhida para tratar estes casos é chamada de **votação linear dinâmica** [Paris 88] que define uma partição principal onde a maioria dos componentes do serviço replicado devem estar contidos. Os serviços devem ser retomados uma vez identificada esta partição, continuando normalmente. Na partição menor, novos pedidos podem ser aceitos porém só serão executados quando da restauração do sistema novamente.

3.3.9 Outros trabalhos

Alguns trabalhos na literatura propõem a adaptação do número de réplicas de uma replicação a certos requisitos de qualidade de serviço durante a evolução da aplicação em tempo de execução. A variação do número de réplicas normalmente é controlada por funções de gerenciamento. Em [Little 94], o programador especifica a qualidade de serviço desejada, através de medidas de desempenho e de disponibilidade, que o sistema tenta manter variando o número de réplicas dos serviços replicados. De maneira similar, em [Cristian 94] é proposto um serviço, com os mesmos propósitos, ou seja varia o número de réplicas para atender requisitos de disponibilidade de um serviço confiável. Com base nessas medidas e nas características do sistema, em ambos trabalhos, o serviço de gerenciamento calcula o número apropriado de réplicas, determinando novas configurações conforme a evolução do sistema e a ocorrência de falhas.

Normalmente, a probabilidade de um serviço replicado ser executado corretamente aumenta a medida em que o número de réplicas é maior. Num sistema distribuído, o aumento do grau de redundância em um serviço faz com que o desempenho do mesmo diminua. Em [Wang 95] é proposto um modelo matemático que tem como objetivo estabelecer o número ideal de réplicas que satisfaça a confiabilidade do serviço, sem degradar o comportamento do sistema do ponto de vista do desempenho.

3.3.10 Considerações sobre os modelos de replicação

Exceto pelos modelos de *tríades* e o usado no sistema MARS que fazem uso das respectivas arquiteturas para às quais foram concebidos, as técnicas estudadas neste item foram desenvolvidas para se executarem em equipamentos convencionais de propósito geral. A não ser pelas premissas sobre as semânticas de falhas nenhuma exigência especial é colocada sobre os elementos de processamento ou o suporte de comunicação (rede de comunicação) usados.

Estes modelos toleram basicamente faltas de hardware, dentro das limitações impostas nas suas hipóteses de faltas. Os modelos normalmente, explicitam o controle externo, ou seja, as interações entre as réplicas e as preocupações em manter a consistência dos estados. Os aspectos ligados às semânticas da aplicação são ignorados nestes modelos de replicação. Para tratar com faltas de valor ou mesmo faltas de software é necessário que se utilize nestes modelos de mecanismos especiais, aproximando estes modelos dos esquemas de tolerância a faltas tratados no capítulo anterior.

Na verdade, enxergamos os modelos de replicação e, mais precisamente, a noção de grupo como um suporte para a programação de mecanismos de tolerância a faltas em sistemas distribuídos. Estes modelos de replicação correspondem então a uma espécie de modelos de execução onde estruturamos os mecanismos de um esquema de tolerância a faltas. Muitos esquemas descritos no capítulo anterior podem ser estendidos para sistemas distribuídos, usando a noção de grupo. É neste sentido que trabalhamos nos próximos capítulos.

3.4 Conclusão

Neste capítulo foi introduzida a abstração de grupo. Esta abstração vem sendo utilizada atualmente como um meio natural de explorar a replicação de componentes físicos inerentes ao sistemas distribuídos. No capítulo nos restringimos a noção de grupo e o uso de replicação de processos ao contexto da tolerância a faltas.

Na seqüência características e conceitos envolvendo as estratégias de replicação foram examinadas.

O uso de técnicas de replicação ativas e semi-ativas em sistemas distribuídos exige que o determinismo de réplicas seja mantido. Este determinismo pode ser mantido através do

controle interno (proibição do uso de construções de linguagens não deterministas, exigir que as decisões sejam tomadas com base em informações locais, etc.) ou do controle externo que envolve a participação de todas as réplicas do grupo em problemas de consenso.

Vários modelos de replicação presentes na literatura foram examinados. Estes modelos em suas premissas de falha se restringem às interações do serviço replicado e o cliente ou nas comunicações entre réplicas. Nestas hipóteses de faltas, os modelos toleram basicamente faltas de hardware. Nestes modelos as semânticas de aplicação são ignoradas nestes modelos.

CAPÍTULO 4

UM ESQUEMA DE TOLERÂNCIA A FALTAS BASEADO EM MÚLTIPLAS REPLICAÇÕES DE PROCESSOS: ESQUEMA MR

Os modelos de replicação do capítulo anterior, diferentes dos esquemas do capítulo 2, se restringem a faltas de hardware ou quando muito em manter a **consistência interativa** [Lamport 82] em um sistema distribuído. Faltas de software que dão origem a semânticas de falha por valor não são tratadas nesses modelos de replicação.

Dentro de nossa visão, indicada no capítulo anterior, enxergamos as técnicas de replicação e a noção de grupo como suportes para a programação distribuída tolerante a faltas, ou seja, as aplicações distribuídas seriam estruturadas segundo esquemas de tolerância a faltas que fariam uso do conceito de grupo.

Qualquer um dos esquemas de tolerância a faltas descritos no capítulo 2 poderia ser estendido para sistemas distribuídos usando a noção de grupo e os modelos de replicação citados. Nesses esquemas, faltas de software ou mesmo faltas por valor são toleradas usando mecanismos especiais construídos, tomando como base a semântica da aplicação. O teste de aceitação e o ajustador são exemplos desses mecanismos.

Porém, os esquemas convencionais apresentam estruturas estáticas pouco flexíveis e algumas vezes dependentes de arquiteturas especiais. Em sistemas distribuídos, sem considerações especiais em relação aos elementos de processamento e tratando com aplicações evolutivas, com características de carga variável, fica difícil se justificar o uso desses esquemas mais tradicionais. Diante disso, encontramos a motivação para a proposição e o estudo de um novo esquema de tolerância a faltas. A princípio na proposição desse esquema nos fixamos em dois pontos:

- fazer uso extensivo dos recursos computacionais disponíveis nos sistemas distribuídos e paralelos, através da replicação massiva de processos;
- explorar atributos da aplicação, no sentido de que este esquema se adapte, melhorando assim a eficiência dos serviços tolerantes a faltas implementados segundo este esquema.

A flexibilidade de um esquema de tolerância a faltas em um sistema distribuído ou paralelo pode ser alcançada de duas maneiras. Considerando que a implementação de um serviço tolerante a faltas envolva a execução de N réplicas de um mesmo servidor distribuídas entre diferentes estações (elementos de processadores), a primeira abordagem explora os recursos computacionais disponíveis no sistema, definindo processamentos com números variáveis de réplicas. Esta abordagem é utilizada em [Little 94], [Cristian 94] e [Wang 95] (item 3.9, capítulo 3). A outra abordagem, adotada neste trabalho, consiste em explorar as características da aplicação.

Seguindo esta segunda abordagem, atualmente, existem na literatura algumas propostas interessantes de trabalhos que flexibilizam a execução de uma aplicação em tempo de execução. Em [Park 96] é proposto um conjunto de ferramentas que permite que um programa se adapte às alterações do sistema, no caso composto a partir de uma máquina paralela. O compilador gera várias versões parametrizadas do mesmo programa e o sistema em tempo de execução vai se utilizando dessas versões conforme as informações de desempenho disponíveis em momentos precisos.

O DARPA (*Defense Advanced Research Projects Agency*) lançou recentemente um novo programa, denominado de ***Chameleon***, com objetivo de desenvolver um sistema dinamicamente adaptável às condições do ambiente [Parker 96]. Segundo Parker, a

capacidade de um sistema se adaptar dinamicamente em tempo de execução é um dos atributos críticos dos futuros sistemas de defesa. Por exemplo, num sistema distribuído o envio de mensagens pode ser transferida de um sistema de comunicação corrente para um alternativo no instante em que os requisitos temporais ou confiabilidade (taxa de erros) não são mais satisfeitos.

O esquema MR (**Múltiplas Replicações**) é introduzido neste capítulo no sentido de permitir estruturas mais flexíveis, adaptáveis às condições evolutivas de um sistema distribuído. Este esquema de tolerância a faltas é formado a partir de múltiplas replicações de processos. As diferentes replicações podem ser usadas no sentido de atender à diversidade de projeto e a diversidade de dados. Os dados de entrada apresentam atributos que permitem escalonar replicações adequadas ao processamento dos mesmos. Esta característica de ativação de replicações a partir de atributos de dados de entrada definem o MR como um instrumental poderoso na construção de servidores tolerantes a faltas, com características de programas adaptáveis como os sugeridos em [Parker 96].

Neste capítulo é inicialmente apresentada uma introdução informal do esquema MR discutindo sua estrutura e os principais conceitos relacionados. Em seguida com base no modelo CSP [Hoare 85] e na Teoria de Processos Replicados [Mancini 88] vários aspectos dos mecanismos MR são explorados em detalhes. O esquema MR foi introduzido inicialmente como unidade para composição de Sistemas Distribuídos Replicados (SDRs); o capítulo encerra com a discussão da implementação de SDRs a partir de MRs, segundo alguns paradigmas de programação distribuída.

4.1 Esquema Múltiplas Replicações (MR)

O MR (**Múltiplas Replicações**) é um esquema de tolerância a faltas, formado a partir de múltiplas replicações de processos. As diferentes replicações podem ser usadas no sentido de atender à **diversidade de projeto** e à **diversidade de dados**. Os algoritmos base das replicações que compõem o esquema MR, são soluções diferentes de um mesmo problema, diferenciadas pelos graus de elaboração e de desempenho. Neste esquema, **ciclo de processamento** é definido como o período de tempo entre a chegada de dados de entrada e a saída dos resultados correspondentes. Em cada ciclo de processamento, no estilo *data-driven*

[Mori 86], uma das replicações do MR é ativada pelo dado de entrada para o processamento associado.

Se considerarmos um sistema distribuído onde nós (estações) são interconectados através de um suporte de comunicação e ainda processos P_A, P_B, P_C,... P_K, (processos base), executando, respectivamente, os algoritmos A, B, C, ..., K, teremos então, formadas a partir da distribuição de cópias de todos os processos base em *N* nós do sistema, as replicações P_A^N, P_B^N, P_C^N,..., P_K^N. Os dados de entrada (m_A, m_B, m_C, m_K) apresentam atributos de modo que a recepção, sobre o canal replicado C^N, de uma destas mensagens determina a execução correspondente no ciclo de processamento considerado. O escalonamento de uma das replicações do esquema MR com base na verificação de atributos de uma mensagem se dá no que chamamos de **teste de escalonamento**. Às saídas destas réplicas juntam-se mecanismos usuais de detecção e mascaramento de erros como teste de aceitação, voto majoritário ou ajustadores e temos o esquema MR completo.

No sentido do uso da diversidade de projeto, as saídas (resultados) de cada uma das réplicas são submetidas a um **teste de aceitação**. Com isto, a tolerância a faltas de software pode ser implementada, a partir da detecção de erro no teste de aceitação, com a desativação da replicação em andamento no ciclo, e subseqüente restauração do estado inicial do MR neste ciclo de processamento e o escalonamento de uma nova replicação. Este procedimento pode se repetir com diferentes replicações, adequadas às características do dado de entrada, até que se tenha um resultado que passe no teste de aceitação.

Os esquemas RB e DRB que fazem uso da diversidade de projeto, apresentam os diferentes algoritmos alternativos ordenados estaticamente. A ordenação nestes esquemas é uma relação pré estabelecida e independente dos dados de entrada, ou seja, os dados são submetidos sempre a mesma seqüência de algoritmos, até que um destes algoritmos forneça um resultado que passe pelo teste de aceitação.

No modelo MR os algoritmos base das replicações estão ordenados de acordo com as características dos dados de entrada que processam. Diferentes algoritmos podem atender o mesmo tipo de dados com mais ou menos precisão. Forma-se então o que chamamos de **classes de algoritmos**. Numa mesma classe, os algoritmos estão dispostos de acordo com o grau de precisão com que processam os dados (mensagens) referentes aquela classe: de

algoritmos mais especializados a algoritmos mais gerais. A ordenação total dos algoritmos segundo atributos, formando as classes é denominada de **hierarquia de especialização**. A figura 4.1 ilustra a formação de classes usando uma hierarquia de algoritmos.

Para um dado de entrada (mensagem), o teste de escalonamento é responsável pela determinação dinâmica da lista de algoritmos que fazem parte de sua classe de algoritmos e que portanto podem processá-lo. O primeiro algoritmo da lista é denominado **preferencial** (o mais preciso), cuja replicação correspondente será executada na recepção do dado, durante o ciclo de processamento do MR. Os outros algoritmos da lista serão alternativas ao preferencial: a lista é percorrida no sentido contrário ao da especialização, até que um dos algoritmos tenha seus resultados passando pelo teste de aceitação. Da figura 4.1, pode-se tirar que se o dado de entrada pertence à classe *k*, a lista de algoritmos, retornada pelo teste de escalonamento, contém os algoritmos *D*, *B* e *A*, sendo *D* o preferencial. Uma vez executado o algoritmo preferencial (*D*), os outros algoritmos da lista (*B* e *A*) somente são executados no caso da falha do algoritmo *D*. É importante salientar o comportamento dinâmico na seleção de algoritmos. A ordenação de algoritmos alternativos não é sempre a mesma. Os algoritmos usados no MR dependem das características dos dados de entrada. Na figura 4.1, por exemplo, um dado da classe *m* tem como lista de algoritmos *E* e *A*.

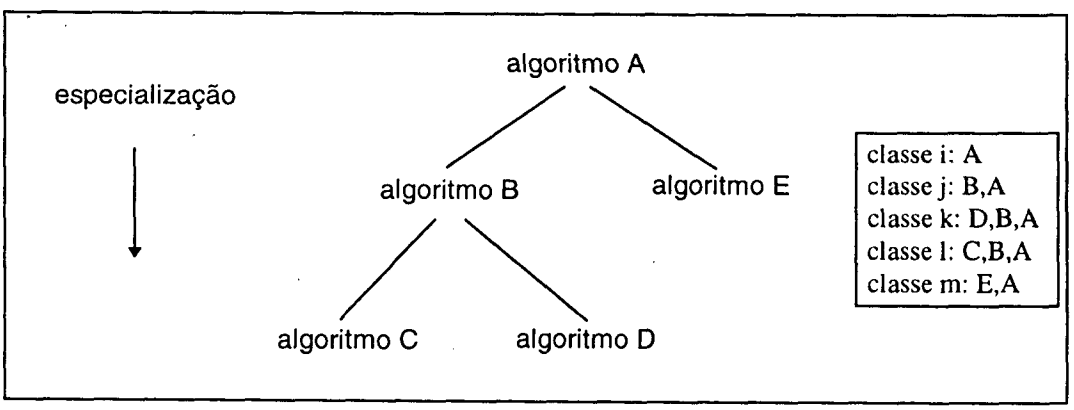


Figura 4.1: Hierarquia de especialização e classes de algoritmos

Os dados que formam o conjunto de entradas processáveis pela hierarquia de especialização em um MR, são logicamente equivalentes e podem ser interpretados como a reexpressão de uma mesma entrada. O modelo MR se aproxima do modelo *N-Copy* [Ammann 87], como técnica de diversidade de dados, onde a reexpressão do dado de entrada tem um algoritmo apropriado (ou uma classe no MR).

A execução das replicações que compõem o esquema MR pode se dar segundo os diferentes modelos de replicações citados no capítulo anterior: Coordenador/Coordenados [Birman 87], Líder/Seguidores [Powell 91] ou no modelo Máquina de Estados [Schneider 90]. A definição por um desses modelos de processamento de grupo determinará então a combinação de réplicas ativas e passivas no MR. Os mecanismos de detecção de erro a serem usados dependem além desses modelos de processamento de grupo, das hipóteses de faltas assumidas.

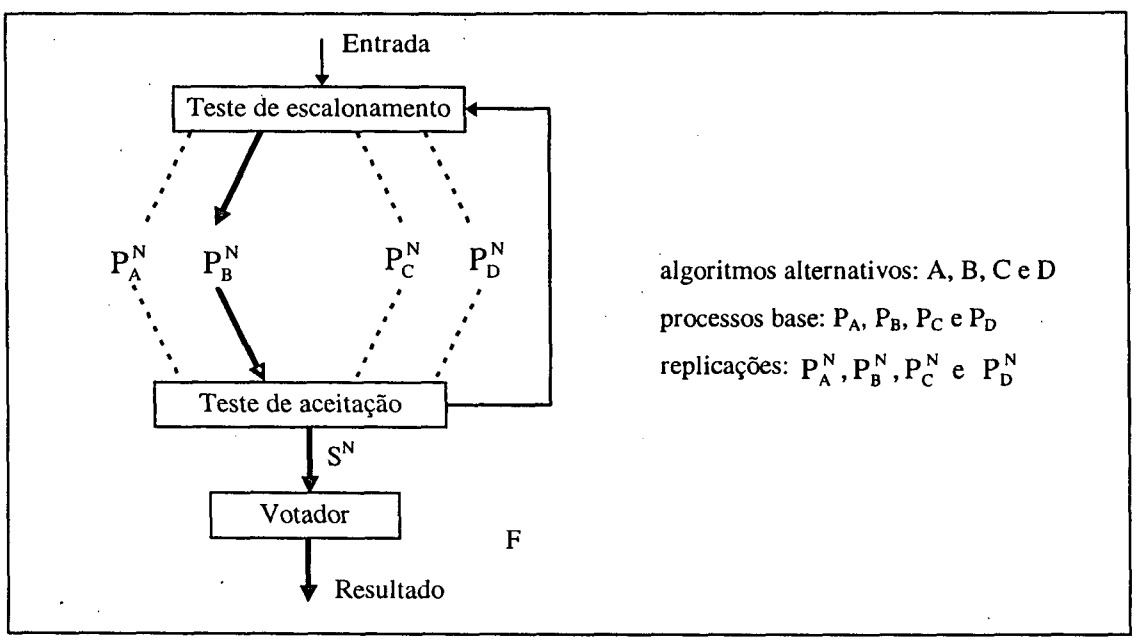


Figura 4.2: Implementação do MR segundo a abordagem de réplicas ativas

Na seqüência do texto, assumimos o modelo de processamento de grupo Máquina de Estados para as replicações do MR. Com isto teremos então réplicas ativas de um mesmo algoritmo base sendo escalonadas em diferentes elementos de processamento do sistema distribuído. Após o teste de aceitação, e no sentido de tolerar hipóteses mais abrangentes de faltas, as saídas produzidas são enviadas a um **votador** ou **ajustador**. Uma vez que as réplicas executam o mesmo algoritmo, a votação será exata, produzindo um único resultado numa comparação bit-a-bit. As premissas sobre semânticas de falhas dos componentes-réplicas assumidas nos modelos de execução do MR podem simplificar o votador ou simplesmente esvaziá-lo no seu significado.

A figura 4.2 ilustra o esquema MR, com o teste de escalonamento, as diferentes replicações e ainda os mecanismos de detecção e mascaramento de erros, na abordagem de execução

assumida acima. Esta figura especifica a replicação P_B^N como sendo escalonada a partir do dado de entrada.

Uma discussão mais ampla de modelos de execução envolvendo tipos de replicações, aspectos de terminação (síncrona, assíncrona e assíncrona temporizada) e as necessidades de comunicação nesses modelos é retomada no capítulo 6.

4.2 Descrição Formal do MR

Neste item, usando o Modelo de traços CSP [Hoare 85], procuramos mostrar a adequação do esquema MR às provas de correção da Teoria de Processos Replicados [Mancini 88]. Usando o mesmo formalismo apresentamos também a especificação de alguns mecanismos que fazem parte do MR. Um resumo do principais comandos CSP é apresentado no anexo A.

4.2.1 Representação CSP de um processo P

No modelo de traço CSP [Hoare 85], um processo P é representado pelo par $(\alpha P, \tau P)$, onde αP é o ***alfabeto*** do processo P e τP é o conjunto dos ***traços*** do processo. O *alfabeto* de um processo corresponde a um conjunto de eventos (ações) considerados relevantes para a descrição de P. Um *traço* do comportamento de P é uma seqüência finita de eventos executados por P e observáveis pelo ambiente durante um dado intervalo de tempo. O par $(\alpha P, \tau P)$ representando um processo P deve satisfazer às seguintes propriedades:

- αP é um conjunto não vazio;
- τP representa os traços do processo P; τP é um subconjunto da linguagem αP^*;
- $<> \in \tau P$ (o traço vazio pertence a αP^*);
- Se $t \in \tau P$ e t_1 é um prefixo de t, então $t_1 \in \tau P$.

Os processos no modelo CSP se comunicam, exclusivamente, por troca de mensagens enviadas através de ***canais síncronos***. O evento ***comunicação*** no modelo é descrito por *(c.m)*, onde c é o canal de comunicação e m o valor da mensagem transmitida. O evento $c!m$ representa o envio de uma mensagem de valor m através de um canal de saída c. Já o evento $c?m$ é complementar ao comando anterior, e representa a recepção da mensagem m no canal

de entrada c. O conjunto de canais de um processo P é representado pelo símbolo χP, onde $c \in \chi P$ e, como conseqüência, a ação $c.m$ pertence ao *alfabeto* αP.

4.2.2 *Processo I/O*

A noção de ***processo I/O*** (entrada/saída), generalizando a noção de *pipe*, foi introduzida em [Hoare 85]. Um processo P é denominado de ***I/O process*** quando o seu conjunto de canais χP pode ser particionado entre canais de entrada ιP e de saída ωP e é caracterizado por uma função monotônica[1]. Desta forma, o processo P satisfaz a especificação dada pela relação $t \lceil \omega P \leq f(t \lceil \iota P)$, ou seja:

$$P \textbf{ sat } t \lceil \omega P \leq f(t \lceil \iota P)$$

isto significa que toda observação possível do comportamento de P é descrita pela relação $t \lceil \omega P \leq f(t \lceil \iota P)$.

Definindo dois conjuntos de traços U e V, tais que $U \subseteq \tau P \lceil \iota P$ e $V \subseteq \tau P \lceil \omega P$, P é dito ser *input-guarded* se para todo $t \in \tau P$ com $t \lceil \iota P \in U$ implica que $t \lceil \omega P \in V$. Em [Koutny 91] um processo *input-guarded* P tem como notação $P \in IG(U,V)$ ou simplesmente $P \in IG$.

Além dos conjuntos αP e τP mais dois conjuntos caracterizam o comportamento de um processo P [Hoare 85]:

- o conjunto de falhas de P, $\varnothing P = (t,R)$ que descreve $R(R \subseteq \alpha P)$ como conjunto de possíveis *recusas* (*refusals*) de eventos oferecidos pelo ambiente externo ao processo P, depois de executado o traço t.
- o conjunto de divergência de $P(\partial P)$ descreve que se $t \in \partial P$, então o processo P deve divergir depois de t executado.

Este dois conjuntos incluem no modelo CSP o comportamento caótico ou seja, um processo pode executar qualquer ação e pode recusar qualquer conjunto de ações. A noção de processo guardado pode ser estendida a partir destas definições:

---

[1]Uma função monotônica é tal que $f(t) \leq f(u)$ para todo $t \leq u$.

$$P \in IG(U,V) \wedge t \in \partial P \Rightarrow t\lceil \iota P \notin U$$

Com base ainda nas definições acima, o conceito ***processos deterministas*** pode ser introduzido. P é um processo I/O determinista implementando a função f se [Koutny 91, Hoare 85]:

a) $\tau P = \{t \mid \forall\, u \leq t: u\lceil \omega P \leq f(u\lceil \iota P)\}$

b) se $(t, \{a\}) \in \varnothing P \Rightarrow t\,\hat{}\,\langle a \rangle \notin \tau P$

P como um processo I/O determinista (notado como $P = DIO(f)$) é *input-guarded* e em conseqüência está livre de divergência. Neste estudo os processos serão tidos como seguindo o modelo ***deterministic input/output process*** quando corretos.

4.2.3 Teoria de Processos replicados (Teoria de Mancini)

No modelo proposto em [Mancini 88] o alfabeto de um processo P é simplificado de maneira a representar apenas as comunicações, ou seja:

$$\alpha P = \{c.m \mid c \in \chi P\}$$

Como conseqüência, o processo P pode ser representado pelo par $(\chi P, \tau P)$, substituindo $(\alpha P, \tau P)$. Com base na representação acima, a N-replicação ($N > 1$) de um processo base P corresponde a um processo replicado (P^N), que apresenta N cópias dos canais de P, executadas de maneira consistente, em relação ao comportamento de P. A formalização da existência de N cópias dos canais de P é dada por:

$\mathbf{R_1}$ $\qquad\qquad \chi P^N = \chi P \times \{1,2,...,N\}$

ou seja, se P^N é uma replicação de P, então todo canal $c[n] \in \chi P^N$, onde $1 \leq n \leq N$, é uma versão ou cópia do canal base $c \in \chi P$. O conjunto de réplicas de um mesmo canal, isto é, $\{c[1],c[2],...,c[N]\}$ é representado por c^N e o conjunto de todos os canais replicados, ou seja, $c^N \cup d^N \cup \ldots$, é representado por C^N.

A noção de consistência entre P^N e P é satisfeita a partir da observação de seus traços. Considerando que o processo replicado P^N contém N cópias de P, cada traço de P^N pode ser visto como uma seqüência de eventos na forma de permutações $\{c[1].m, c[2].m,..., c[N].m\}$,

desde que $c.m$ seja um evento no traço do processo base P. Esta caracterização de eventos N-upla no traço de P^N, em geral pode ser impossível de se obter considerando a ocorrência de falhas em *links* ou nos componentes de P^N (P_n). Uma condição menos restritiva foi introduzida em [Mancini 88]: "se P^N é uma N-replicação de P e t^N é um traço de P^N, então deve ser possível extrair um traço de P a partir de t^N ". Neste sentido é assumida a existência da relação ***extract***:

$\mathbf{R_2}$ $\qquad t^N \in \tau P^N \wedge \textbf{\textit{extract}}\ (t^N, t) \Rightarrow t \in \tau P$

sendo que:

$\mathbf{E_1}$ $\qquad \forall t^N . \exists\, t\, .\, \textbf{\textit{extract}}\ (t^N, t)$

Considerando nas mensagens trocadas entre os processos a existência da função ***id:Msg→Idents*** que atribui a cada mensagem $(m \in Msg)$ identificadores únicos, teremos então:

$\mathbf{R_3}$
$$\alpha P = \{c.m \mid c \in \chi P \wedge m \in Msgs\}$$
$$\alpha P^N = \{c[n].m \mid (c \in \chi P) \wedge (1 \leq n \leq N) \wedge (m \in Msgs)\}$$
$$t \in \tau P \wedge t = \langle \ldots, c!m_i, \ldots, c!m_j, \ldots \rangle \Rightarrow id(m_i) \neq id(m_j)$$

Considerando a seqüência $t^N \lceil c^N \lceil (\textbf{\textit{id}}(x) = i)$, onde $t^N \lceil c^N$ é um traço resultante da projeção de t^N em c^N, eliminando os eventos que não ocorrem nos canais c^N. Por outro lado, o predicado $\textbf{\textit{id}}(x) = i$ elimina do traço $t^N \lceil c^N$ todos os elementos em que $\textbf{\textit{id}}(x) \neq i$. Ou seja, o resultado de $t^N \lceil c^N \lceil (\textbf{\textit{id}}(x) = i)$ é um traço constituído dos canais de c^N cujas mensagens apresentam o identificador i. Pode-se assumir então que se t é extraído de t^N, cada ação $c.m$ de t pode ser ***eleita*** a partir da seqüência $t^N \lceil c^N \lceil (\textbf{\textit{id}}(x) = i)$, onde $i = \textbf{\textit{id}}(m)$. Neste sentido é definida a função ***elect***, tal que:

$\mathbf{E_2}$ $\qquad \textbf{\textit{elect}}\ (t^N \lceil c^N \lceil (\textbf{\textit{id}}(x) = i)) = c.m \ \wedge (\textbf{\textit{id}}(m) = i) \vee \textbf{\textit{elect}}\ (t^N \lceil c^N \lceil (\textbf{\textit{id}}(x) = i)) =$

NIL

$$\textbf{E}_3 \qquad \begin{array}{l} \textit{extract}(t^N,t) \wedge e = \textit{elect}\,(t^N \lceil c^N \lceil (id(x) = i)) \Rightarrow \\ \textbf{se}\ e = NIL \quad \textbf{então}\ (t \lceil c \lceil (id(x) = i)) = <> \\ \qquad\qquad\qquad \textbf{senão}\ (t \lceil c \lceil (id(x) = i)) = <e> \vee\ t \lceil c \lceil (id(x) = i) = <> \end{array}$$

O segundo componente do laço **senão** da propriedade E_3 ($t \lceil c \lceil ((x) = i) = <>$) indica que a mensagem eleita não necessariamente é um elemento do traço extraído.

Para que a extração de um traço seja independente do contexto, no sentido que se t é extraído de t^N, a contribuição de $t^N \lceil c^N$ em t deve ser independente das ações eliminadas de t^N pela projeção em c^N, ou seja:

$$\textbf{E}_4 \qquad \textit{extract}(t^N,t) \Rightarrow \textit{extract}\,(t^N \lceil c^N,\ t \lceil c)$$

$$\textbf{E}_5 \qquad \textit{extract}(s^N \lceil c^N,t) \Rightarrow \exists u.\ (u \lceil c = t \wedge \textbf{extract}\,(s^N,\ u))$$

Para ilustrar o uso das funções ***extract*** e ***elect*** nós apresentaremos um exemplo de extração dos traços t^N de P^N de maneira que este último corresponda a uma replicação de P. Para isto, são impostas mais duas condições:

- A primeira delas assumi que todo o conjunto de identificadores de mensagens (*Idents*) é totalmente ordenado, e que a projeção de um traço extraído dos canais é ordenado pelos identificadores de mensagem, sem *gaps*:

$$\textbf{E}_6 \qquad \begin{array}{l} \textit{extract}(t^N,t) \Rightarrow \textit{no\_gaps}(t) \\ \text{onde, } \textit{no\_gaps}(t) \Leftrightarrow \forall c.\ \textit{no\_gaps}\,(t \lceil c) \end{array}$$

  ou seja, qualquer extração dá origem a um traço $u = <\ldots, c.m_i,\ c.m_j, \ldots>$, onde uma mensagem m_i terá um sucessor m_j cujo $id(m_j)$ sucede $id(m_i)$ na ordenação definida sobre *Idents*, como conseqüência $no\_gaps(u)$ é *true*;

- A segunda condição garante que todos os traços extraídos de t^N devem ser **máximos**, ou seja, as seqüências de eventos que caracterizam estes traços devem apresentar o maior número possível de eventos, no sentido que se um evento $c.m$ pode ser eleito, este deve estar incluído no traço extraído, contanto que a condição $\mathbf{E}_6$ não seja violada.

$\mathbf{E_7}$ $\qquad\qquad$ $extract(t^N, t) \wedge extract(t^N, t_1) \Rightarrow \forall c \,.\, t\lceil c = t_1\lceil c$

Exemplo 4.1:

Dado o traço $t^N = <c[1].m^{\alpha}, c[2].m^{\alpha}, c[1].m^{\beta}, c[2].m^{\beta}, c[1].m^{\gamma}, c[1].m^{\delta}, c[3].m^{\delta}>$ e assumindo que o número máximo de réplicas (N) é igual a 3 e as extrações são feitas de forma a se ter a maioria absoluta, dada por $\lfloor N/2 \rfloor + 1$. Considerando que m^{α}, m^{β}, m^{γ} e m^{δ} representam as mensagens de identificadores $id(m^{\alpha})=\alpha$, $id(m^{\beta}) = \beta$, $id(m^{\gamma}) = \gamma$ e $id(m^{\delta}) = \delta$, onde $\alpha < \beta < \gamma < \delta$, nós temos:

$$elect(t^N\lceil c^N\lceil(id(x) = \alpha)) = c.m^{\alpha} \qquad elect(t^N\lceil c^N\lceil(id(x) = \beta)) = c.m^{\beta}$$
$$elect(t^N\lceil c^N\lceil(id(x) = \gamma)) = NIL \qquad elect(t^N\lceil c^N\lceil(id(x) = \delta)) = c.m^{\delta}$$

Se t e t^N satisfazem a relação $extract(t^N,t)$, então t não pode conter mensagem de identificador γ, pois $elect(t^N\lceil c^N\lceil(id(x) = \gamma)) = NIL$ (E3); além disto t não pode conter $c.m^{\delta}$, senão a condição ***no_gaps*** (E6) é violada; conseqüentemente: $t\lceil c\lceil(id(x) = \delta) = <>$. Com isto:

$$t = <c.m^{\alpha}, c.m^{\beta}>$$

Com base nas condições e relações introduzidas acima, podemos concluir a definição de N-replicação de um processo base como [Mancini 88]:

Definição 4.1: Dadas as funções ***extract*** e ***elect*** e as relações E1-E7, um processo P^N é uma N-replicação de um processo base P, se e somente se as relações R_1, R_2 e R_3 são verdadeiras.

Até o presente momento a correção do processo P^N em relação a P foi considerada observando-se apenas o comportamento de P e P^N, isto é, foram abstraídas as estruturas internas de ambos os processos. Sendo assim, considerando que Ψ é uma especificação de um processo P, o critério de correção (consistência) extraído da relação $\mathbf{R_2}$, permite que se tenha:

se P **sat** $\Psi(t) \wedge t \in \tau P$
então P^N **sat** $\forall u.(extract(t^N,u)) \Rightarrow \Psi(u)$

que é denominado de **critério de correção global** em [Mancini 88].

4.2.4 Critério de correção de P^N como replicação de P num contexto de faltas

Na visão apresentada anteriormente o sistema distribuído pode ser replicado, através da replicação de seus componentes (processos I/O). O teorema que se segue endereça o problema de montar N cópias de um processo base P (processo I/O), que implementa uma N-replicação P^N de P mesmo em um contexto com presença de faltas.

Teorema T_1 [Mancini 88]: Se a maioria de cópias de P não se afastam do modelo ***deterministic I/O process*** (não faltosos), então P^N é uma N-replicação de P.

Os lemas e as provas associadas a este teorema são apresentados em [Mancini 88].

4.2.5 Extensões à teoria de Mancini: nossa proposta

A comunicação no modelo de Mancini é modelada, tendo como base o conceito de Rendez-vous, presente no CSP. Nesta situação, é difícil caracterizar as comunicações de grupo como alfabeto de P^N, usando canais ponto-a-ponto e em situação de ocorrência de falhas na comunicação. A solução proposta em [Mancini 88] foi reduzir as ações observáveis de P^N a $c[n].m$.

Na nossa proposta, assumimos que a comunicação entre processos (grupo de processos) é efetuada por meio de protocolos de difusão confiável. Sendo assim, é perfeitamente aceitável considerar que as ações $c[n].m$, ocorram simultaneamente em todas as réplicas P_n que compõem P^N. Desta forma é possível caracterizar $c^N.m$ (N-upla) como ação de P^N e portanto, é necessário a introdução das duas proposição abaixo:

Proposição 1: O alfabeto de processo replicado P^N apresenta como ações N-upla, tal que:

$$\alpha(P^N) = \{(c_{[1]}.m, c_{[2]}.m, c_{[3]}.m, \ldots, c_{[N]}.m) \mid \forall n.\ 1 \leq n \leq N \ \wedge \ c[n].m \in (\alpha P_n) \ \wedge \ m \in Msgs\}$$

Por simplificação as ações de P^N (N-upla) serão representadas por $c^N.m$ no restante deste texto.

Proposição 2: Os traços t^N de uma replicação constituirão seqüências destas N-upla, com $t^N = (t^N)_0 \ ^\wedge (t^N)'$, onde $(t^N)_0$ corresponde a N-upla *head* e $(t^N)'$ é o traço *trail* de t^N.

A redefinição do alfabeto de P^N leva então a uma reescrita de $\mathbf{R_3}$:

$$\mathbf{R_3} \quad \begin{array}{l} \alpha P = \{c.m \mid c \in \chi P \wedge m \in Msgs\} \\ \alpha P^N = \{c^N.m \mid \forall n.\ 1 \leq n \leq N \ \wedge \ c[n].m \in (\alpha P_n) \ \wedge \ m \in Msgs\} \\ t \in \tau P \ \wedge \ t = <\ldots c!m_i, \ldots, c!m_j, \ldots> \Rightarrow id(m_i) \neq id(m_j) \end{array}$$

Considerações:

- R_2 e as funções ***extract*** e ***elect*** não sofrem alterações com a redefinição de $\alpha(P^N)$. A priori, com as condições impostas pela redefinição do alfabeto de P^N, os resultados da função ***elect***, para qualquer identificador de mensagem, seriam sempre valores contidos em N-upla do alfabeto αP (no caso, eventos do tipo $c.m$) e a extração de $t \in \tau P$ a partir de $t^N \in \tau P^N$ seria máxima, com $\#t = \# t^N$.
- Esta redefinição de $\alpha(P^N)$ não está considerando situações de faltas e portanto, a N-upla apresenta todos os seus componentes com valor idêntico de mensagem (m). Esta situação será revista quando da modelagem do sistema de comunicação, onde réplicas podem se afastar do modelo I/O determinista (item 4.2.2), e então as funções ***elect*** e ***extract*** poderão assumir valores diferentes dos indicados nas considerações do parágrafo anterior ($\#t \neq \#t^N$).

4.2.6 Modelo MR em CSP

Com base nas definições anteriores o processo MR pode ser definido pela combinação *alternate* de processos prefixos apresentada abaixo:

$$\mathrm{MR} = (c^N ? m_1 \rightarrow P_1{}^N \mid c^N ? m_2 \rightarrow P_2{}^N \mid \ldots \mid c^N ? m_k \rightarrow P_k{}^N)$$

onde $P_1{}^N, P_2{}^N, \ldots P_k{}^N$ correspondem a k diferentes replicações que são ativadas a partir da recepção da mensagem m_i com $1 \leq i \leq k$, pelo teste de escalonamento. Numa forma mais sucinta o processo MR pode ser representado por:

$$c^N ? x: B \rightarrow P(x)^N$$

onde $B = \{m_1, m_2, \ldots m_k\}$ e $P(x)^N$ é uma expressão definindo um processo replicação para cada evento x diferente.

4.2.6.1 Alfabeto e traços de um processo MR

No modelo MR, o escalonamento inicial de uma replicação P_i^N depende da ocorrência da N-upla:

$$(c_{[1]} ? m_i, c_{[2]} ? m_i, \ldots, c_{[k]} ? m_i), \text{ ou simplesmente } c^N ? m_i$$

e os processos prefixos $c^N ? m_i \rightarrow P_i^N$, representam então neste caso, a seleção de uma replicação P_i^N.

Considerando que os processos $(c ? m \rightarrow P)$ e P são equivalentes [Hoare 85], teremos então $\alpha(P) = \alpha(c ? m \rightarrow P)$ e que os conjuntos de canais representados por $\chi(c ? m \rightarrow P)$ e χP são por sua vez idênticos. Como conseqüência, também considerando as extensões do item 4.2.5, os processos replicados $(c^N ? m \rightarrow P_i^N)$ e P_i^N manterão uma relação de equivalência:

$$\mathbf{E_8} \qquad \begin{array}{l} \alpha(c^N ? m_i \rightarrow P_i^N) = \alpha(P_i^N) \ \wedge \ c^N ? m_i \in \alpha(P_i^N) \\ \chi(c^N ? m_i \rightarrow P_i^N) = \chi(P_i^N) \\ \textbf{\textit{traços}}(c^N ? m_i \rightarrow P_i^N) \subseteq \textbf{\textit{traços}}(P_i^N) \end{array}$$

O processo CSP que dá o comportamento MR terá então:

$$\mathbf{E_9}\quad \begin{aligned} &\alpha(\mathrm{MR}) = \alpha(P_1^N) \cup \alpha(P_2^N) \cup \ldots \cup \alpha(P_k^N) \\ &\chi(\mathrm{MR}) = \chi(P_1^N) \cup \chi(P_2^N) \cup \ldots \cup \chi(P_k^N) \\ &\textit{traços}(\mathrm{MR}) = \textit{traços}(c^N ?\, m_1 \rightarrow P_1^N) \cup \textit{traços}(c^N ?\, m_2 \rightarrow P_2^N) \cup \ldots \\ &\qquad\qquad \cup \textit{traços}(c^N ?\, m_k \rightarrow P_k^N) \\ &\textit{traços}(\mathrm{MR}) \subseteq \textit{traços}(P_1^N) \cup \textit{traços}(P_2^N) \cup \ldots \cup \textit{traços}(P_k^N) \end{aligned}$$

as condições $\mathbf{E_9}$ podem ser verificadas a partir das definições do alfabeto e traços de processos *alternate* [Hoare 85].

4.2.6.2 O MR como um processo replicação

Pela teoria de Mancini um processo é uma replicação de um processo base se dado as funções ***extract*** e ***elect*** que satisfazem as condições E_1-E_7, as regras R_1, R_2 e R_3 são verdadeiras. Nesta situação podemos então introduzir o teorema abaixo:

Teorema T_2: Se os processos P_1^N, P_2^N,..., P_k^N são, respectivamente, replicações de P_1, P_2,..., P_k, então o processo MR é uma replicação que tem como processo base:

$$(c?m_1 \rightarrow P_1 \mid c?m_2 \rightarrow P_2 \mid \ldots \mid c?m_k \rightarrow P_k)$$

ou seja:

$$\begin{aligned} \mathrm{MR} &= (c^N?m_1 \rightarrow P_1^N \mid c^N?m_2 \rightarrow P_2^N \mid \ldots \mid c^N?m_k \rightarrow P_k^N) \\ &= (c?m_1 \rightarrow P_1 \mid c?m_2 \rightarrow P_2 \mid \ldots \mid c?m_k \rightarrow P_k)^N \end{aligned}$$

Prova: A prova do teorema T_2 é dividida em duas partes.

Parte 1: Provar que $(c^N?m_i \rightarrow P_i^N) = (c?m_i \rightarrow P_i)^N$.

Esta prova depende da verificação da definição 4.1 da teoria de Mancini; ou seja, depende basicamente da verificação das regras R_1, R_2 e R_3 para que $(c^N?m_i \rightarrow P_i^N)$ seja visto como replicação de $(c?m_i \rightarrow P_i)$. Assumindo também que P_i^N é uma replicação de P_i.

verificação de R_1:

Sabendo que:

$\chi P_i^N = \chi P_i \times \{1, 2, \ldots N\}$

$\chi(c^N?m_i \rightarrow P_i^N) = \chi(P_i^N)$ (condição E_8)

$\chi(c?m_i \rightarrow P_i) = \chi(P_i)$

teremos então:

$$\chi(c^N?m_i \rightarrow P_i^N) = \chi(c?m_i \rightarrow P_i) \times \{1,2,\ldots N\}$$

o que verifica a regra 1.

verificação de R_3:

Sabendo que P_i^N é a replicação de P_i e portanto satisfaz R_3 e que, da condição E_8 e [Hoare 85], tem-se:

$$\alpha(c?m_i \rightarrow P_i) = \alpha(P_i)$$
$$\textbf{traços}(c?m_i \rightarrow P_i) \subseteq \textbf{traços}(P_i)$$
$$\alpha(c^N?m_i \rightarrow P_i^N) = \alpha(P_i^N)$$
$$\textbf{traços}(c^N?m_i \rightarrow P_i^N) \subseteq \textbf{traços}(P_i^N)$$

com isto, então, $(c^N ? m_i \rightarrow P_i^N)$ também satisfaz R_3.

verificação de R_2:

Para que R_2 seja verificada é necessário que tenhamos:

$$t^N \in \textbf{traços}(c^N ? m_i \rightarrow P_i^N) \wedge \textbf{extract}(t^N, t) \Rightarrow t \in \textbf{traços}(c? m_i \rightarrow P_i)$$

assumindo as extensões propostas no item 4.2.5 e também de [Hoare 85]:

$$t^N = <c^N ? m_i> \hat{} (t^N)' \quad \wedge \quad t_i^N, (t^N)' \in \textbf{traços}(P_i^N)$$
$$t = <c ? m_i> \hat{} t' \quad \wedge \quad t, t' \in \textbf{traços}(P_i)$$

e sendo P_i^N replicação de P_i, teremos então, de R_2:

(a) $<c^N ? m_i> \hat{} (t^N)' \in \textbf{traços}(P_i^N) \wedge$

$\textbf{extract}(<c^N ? m_i> \hat{} (t^N)', <c ? m_i> \hat{} t') \Rightarrow \quad <c ? m_i> \hat{} t' \in \textbf{traços}(P_i)$

como:

$$\textbf{extract}(<c^N ? m_i> \hat{} (t^N)', <c ? m_i> \hat{} t') =$$
$$\textbf{extract}(<c^N ? m_i> \hat{} (t^N)' \lceil \chi(P_i^N), <c ? m_i> \hat{} t' \lceil \chi(P_i)) =$$
$$\textbf{extract}(<c^N ? m_i> \hat{} (t^N)' \lceil \chi(c^N ? m_i \rightarrow P_i^N), <c ? m_i> \hat{} t' \lceil \chi(c ? m_i \rightarrow P_i))$$

como $< c^N ?\ m_i > \wedge (t^N)' \in$ ***traços***$(c^N ?\ m_i \rightarrow P_i^N)$ e
$< c ?\ m_i > \wedge t' \in$ ***traços***$(c?\ m_i \rightarrow P_i)$ a expressão (a) pode ser rescrita na forma:

(b) $t^N \in$ ***traços***$(c^N ?\ m_i \rightarrow P_i^N) \wedge$ ***extract***$(t^N, t) \Rightarrow t \in$ ***traços***$(c?\ m_i \rightarrow P_i)$

o que verifica a condição de R_2, determinando que

$(c^N ? m_i \rightarrow P_i{}^N) = (c ? m_i \rightarrow P_i)^N$.

Parte 2: verificação da condição

(c) $(c^N ?\ m_1 \rightarrow P_i^N) \mid (c^N ?\ m_2 \rightarrow P_2^N) \mid \ldots \mid \mid (c^N ?\ m_k \rightarrow P_k^N) =$

$((c?\ m_1 \rightarrow P_i) \mid (c ?\ m_2 \rightarrow P_2) \mid \ldots \mid \mid (c?\ m_k \rightarrow P_k))^N$

Utilizando o mesmo procedimento de prova anterior, pode se verificar facilmente que as regras R_1, R_2 e R_3 são mantidas para os processos da expressão (c) o que implica que o processo MR, corresponde a replicação do processo base $(c?m_1 \rightarrow P_1 \mid c?m_2 \rightarrow P_2 \mid \ldots \mid c?m_k \rightarrow P_k)$.

Teorema T3: o processo MR = $(c^N ? m_1 \rightarrow P_1{}^N \mid c^N ? m_2 \rightarrow P_2{}^N \mid \ldots \mid c^N ? m_k \rightarrow P_k{}^N)$ é uma replicação do processo $(c?m_1 \rightarrow P_1 \mid c?m_2 \rightarrow P_2 \mid \ldots \mid c?m_k \rightarrow P_k)$ se a maioria das *N* réplicas do processo base é não faltosa.

Prova: Pelo teorema T_2 é possível concluir que o processo $(c^N ? m_1 \rightarrow P_1{}^N \mid c^N ? m_2 \rightarrow P_2{}^N \mid \ldots \mid c^N ? m_k \rightarrow P_k{}^N)$ é uma replicação de $(c?m_1 \rightarrow P_1 \mid c?m_2 \rightarrow P_2 \mid \ldots \mid c?m_k \rightarrow P_k)$, podemos então assumir os resultados do teorema 1 apresentado em [Mancini 88], onde a condição de replicação só é satisfeita na presença da maioria de processos bases corretos, assumindo o comportamento I/O determinista (não faltosos).

4.2.7 O processo base de um MR

O processo *alternate* $(c?m_1 \rightarrow P_1 \mid c?m_2 \rightarrow P_2 \mid \ldots \mid c?m_k \rightarrow P_k)$ corresponde a base de replicação MR. A hierarquia de especialização é implementada então através dos processos P_1, P_2,...,P_k, bases das k replicações envolvidas no MR. A integração de mecanismos de detecção de erros nestes processos e a ordenação dos mesmos segundo a noção de hierarquia de especialização, definida no item 4.1, permite que falhas em um processo $P_i[n]$ ative um processo sucessor na hierarquia, representado por P_{suc_Pi}, de maneira a fornecer o serviço num sentido menos preciso. Os processos $P_i[n]$ que compõem o *alternate* são dados pela expressão CSP:

$$P_i[n] = \langle salva\_estado \rangle;\ result := alg_i(m_i);\ MEC\_DET(result);\ e^N\ !\ result\_consenso \qquad (4.1)$$

que indica o salvamento de estado *(salva_estado)*, a execução do algoritmo de aplicação alg_i, os mecanismos de detecção de (*MEC_DET*) e o envio dos resultados de consenso pelo canal de difusão $(e^N!\ result\_consenso)$ como operações no processo $P_i[n]$. *MEC_DET(result)* agrupa as técnicas de detecção e recuperação de estados errôneos. Estes mecanismos são deixados a critério do programador de aplicação no sentido da escolha e na disposição dos mesmos no processo *MEC_DET(result)*, de maneira a atender as necessidades impostas pelo seu modelo de execução. A princípio o uso dos **testes de aceitação** [Anderson 81] se fazem necessários para a implementação da hierarquia de especialização.

A título de ilustração, tomando o modelo de réplicas ativas e sem privilégio (replicação ativa, capítulo 3), o uso de técnicas de voto neste caso se justifica e o processo *MEC_DET* assume a forma:

$$MEC\_DET(result) = ((\langle restaura\_estado \rangle;\ P_{suc\_Pi}) \lessdot ta(result) \gtrdot ((d^N!\ result) \parallel VOTADOR_n)) \qquad (4.2)$$

A função ***ta(result)*** é uma expressão booleana que sinaliza a consistência do parâmetro ***result*** em relação a critérios especificados pelo programador. O processo *VOTADOR* representa o mecanismo de voto. A opção de réplicas ativas e sem privilégio implica em determinadas semânticas de falha (capítulo 3), na implementação de voto replicado. O votador, a título de

ilustração, foi colocado dentro do processo *MEC_DET*, o que determina a necessidade de um endereço de grupo (*multicast*) exclusivo para o voto (d^N). Soluções menos dispendiosas em termos de comunicação têm o esquema do voto colocado ou no sistema de comunicação [Powell 91] ou na entrada dos processos receptores de resultados [Mancini 88]. Neste último caso os canais d^N (expressão 4.1) coincidirão como os canais e^N (expressão 4.2).

O processo $VOTADOR_n$ é o responsável pela obtenção do resultado de consenso (***result_consenso***) a partir do conjunto de valores de saídas das réplicas P_n. O limite inferior de valores (***lim_min***) define o número mínimo no qual é possível o consenso. Este limite depende das hipóteses de faltas assumidas para P_i^N:

- faltas de ***crash*** ou **omissão**, a função voto se resume a encontrar uma única mensagem diferente de ***Null*** no conjunto de mensagens m_i;
- faltas **arbitrárias e de valor**, a função voto é implementada a partir da comparação das m_i mensagens; o valor que se repetir neste conjunto em número maior ou igual a $\lfloor N/2 \rfloor + 1$ (*lim_min*) será o valor de consenso (voto majoritário).

O processo $VOTADOR_n$ é explicitado na expressão abaixo:

$$VOTADOR_n = (i := 0; \mu X(d^N ? m_i \rightarrow (i := i+1; (X \triangleleft (i \leq lim_{min}) \triangleright (result\_consenso := voto(m_1, m_2, \ldots m_i); (SKIP \triangleleft (result\_consenso \neq NULL) \triangleright (X \triangleleft (i < N) \triangleright SKIP))))))$$

onde, o primeiro teste indica se o limite mínimo necessário para a obtenção de um resultado de consenso foi atingido, o segundo teste verifica se o valor de consenso foi obtido e finalmente o último teste indica que todas as mensagens foram recebidas.

4.3 Composição de sistemas distribuídos usando MRs

A composição de Sistemas Distribuídos (SD) a partir de processos CSP não replicados não explora a separação física inerente aos sistemas distribuídos como forma de tolerar faltas em um ou mais componentes do sistema. Por outro lado, a concepção de Sistemas Distribuídos Replicados (SDR) consiste na composição de componentes replicações de forma que mesmo quando da ocorrência de faltas em um número limitado de componentes físicos ou lógicos o

sistema apresente o comportamento desejado. Recentemente, foi proposto em [Tully 90b] a concepção de SDR a partir de componentes replicações denominados de ***Tríades***. Um objetivo inicial neste trabalho era explorar a capacidade de processamento disponível em sistemas distribuídos ou sistemas paralelos (redes de *transputers*) e criar programas distribuídos usando replicações massivas de seus componentes.

As réplicas nestes componentes replicações poderiam se apresentar estruturadas segundo esquemas já conhecidos de tolerância a faltas. Nós nos deteremos na utilização de componentes MRs compondo SDRs. Um SDR, segundo esta visão, é então uma composição de processos replicações MR_1, MR_2,..., MR_i.

Ao considerarmos o alfabeto de um processo MR como *N*-upla (item 4.2.6.2), a caracterização destes eventos, usando o modelo comunicação do CSP fica limitada. Neste sentido é introduzido os processos de comunicação SC_{ij}; estes processos são os responsáveis pelo mapeamento de endereço de grupo entre o MR emissor e o receptor, de modo que um SDR toma a forma:

$$SDR = (MR_0 \,||\, SC_{01} \,||\, MR_1 \,||\, SC_{12} \,||\, ... \,||\, MR_{i-1} \,||\, MR_i)$$

A composição de um SDR a partir de quatro módulos MR é ilustrada na figura 4.3.

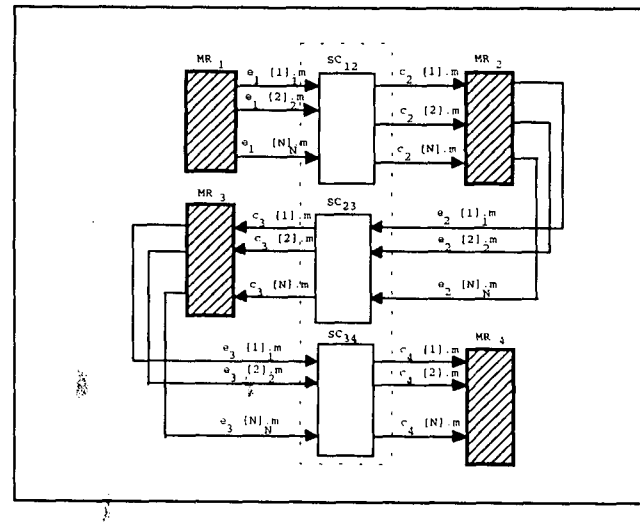


Figura 4.3: Exemplo de SDR (Sistema Distribuído Replicado) composto por MRs

4.3.1 O suporte de comunicação para os SDRs (processos SC_{ij})

No modelo de comunicação CSP, cada canal liga somente dois processos (o emissor e o receptor) e serve exclusivamente ao transporte unidirecional de mensagens. A comunicação entre dois processos replicações, usando este modelo, determinaria a necessidade de uma grande quantidade de canais CSP (comunicação ponto-a-ponto), o que implicaria também no gerenciamento a nível de réplicas da ordenação de mensagens e no tratamento de eventuais falhas nas comunicações. O uso de difusão confiável no lugar de comunicações ponto a ponto, limita estes aspectos aos processos **Sistema de Comunicações** *(SC_{ij})*, introduzidos no sentido de permitir a representação das necessidades de comunicação de grupo. No item 4.5 foram discutidas as extensões à teoria de Mancini, abrangendo este último modelo de comunicação.

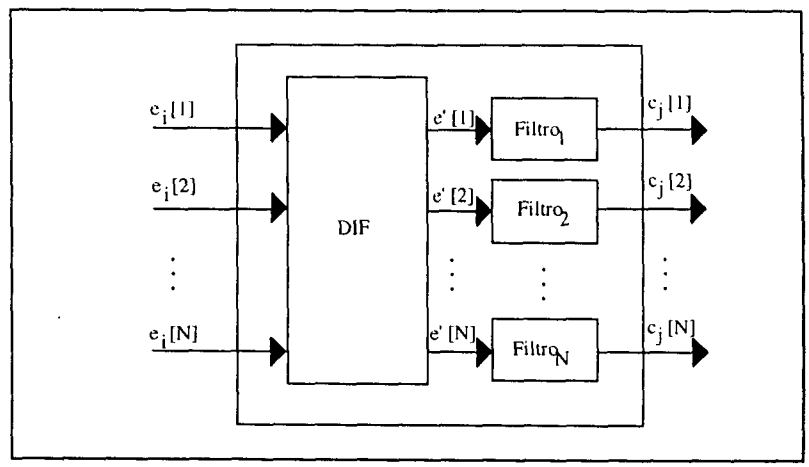


Figura 4.4: Representação de um processo **Sistema de Comunicação *(SC)***

A figura 4.4 descreve um processo *(SC_{ij})* que mapeia as saídas do grupo MR_i no endereço de grupo correspondente as entradas de MR_j. Os componentes de SC_{ij} são os processos ***DIF*** e ***FILTROS***:

$$SC_{ij} = (DIF \parallel Filtro_1 \parallel Filtro_2 \parallel ... \parallel Filtro_N) \setminus E' \quad \text{ou} \quad SC_{ij} = (DIF \parallel Filtro^N) \setminus E'$$

O processo *DIF* implementa a comunicação no estilo difusão (*multicast*) confiável, no endereço de grupo especificado em SC_{ij}, ou seja, uma mensagem emitida em um canal de entrada $e_i[n]$ deste processo é difundida através dos N canais de saída (canais E'):

$$DIF_i = \mu X\ (((e_i[1]\ ?\ x) \rightarrow (e'_1\ !\ x \rightarrow e'_2\ !\ x \rightarrow ... \rightarrow e'_N\ !\ x);X)\ |$$
$$((e_i[2]\ ?\ x) \rightarrow (e'_1\ !\ x \rightarrow e'_2\ !\ x \rightarrow ... \rightarrow e'_N\ !\ x);X)\ |$$
$$\vdots$$
$$((e_i[N]\ ?\ x) \rightarrow (e'_1\ !\ x \rightarrow e'_2\ !\ x \rightarrow ... \rightarrow e'_N\ !\ x);X));$$

A expressão acima, modelando o processo *DIF*, limita-se as relações de entrada e saída. A modelagem completa de um protocolo de difusão confiável é bem mais complexa e foge do escopo deste capítulo, entretanto, a título de ilustração, na figura 4.5 é mostrada a estrutura geral destes protocolos proposta em [Duenãs 94], onde são identificadas para estes protocolos as seguintes fases:

Fase 1: ***difusão*** da mensagem pela camada de difusão confiável (CD) usando os serviços de comunicação de mais baixo nível;

Fase 2: ***recepção*** da mensagem difundida na camada de difusão confiável nas estações;
verificação das propriedades de acordo, ordenação e terminação;

Fase 3: ***Aceitação*** (engajamento) da mensagem pela camada de difusão para envio a camada de aplicação.

Protocolo de Difusão Confiável :: X
Aplicação :: AP_j

Camada de Difusão Confiável na estação *i* :: *CD(i)*
Mensagem enviada pela Aplicação para difusão :: *msg*
Mensagem difundida pela Camada de Difusão Confiável :: *mdif*

X:: *((AP_j? *msg* *mdif* := *msg* + *mensagens de controle*;
(i:1..N) *CD (i)*; *CD (i)* ! *mdif*}) [ ]
((i:1..N) *CD (i)*; *CD (i)* ? *mdif* < *Ordena mdif no buffer de recepção* >) [ ]
(<*Existe mdif no buffer de recepção da CD(j)* >;
*(<*Protocolo de acordo*>; <*Reordena mdif no buffer de recepção* >)
*(<*Aceita mdif* >; < *Envia mdif aceita para a* AP_j)))

Figura 4.5: Modelo de um protocolo de difusão confiável

Dependendo do modelo de processamento de grupo usado, as propriedades de acordo, ordenação e terminação assumem características diversas, definindo diferentes tipos de protocolos de difusão confiável (item 3.1.2.1). Neste capítulo, tem sido usado como referência o modelo de processamento de réplicas ativas e sem privilégio. Então, neste caso, o protocolo deve assumir características de difusão atômica, onde a ordenação é do tipo total, o acordo usado dependerá das hipóteses de faltas e a terminação síncrona ou assíncrona será escolhida conforme as necessidades de sincronismo deste modelo de processamento.

Os processos *FILTRO*s que completam SC_{ij} são responsáveis pelo descarte de mensagens replicadas. Foi assumido da regra R_3, item 4.2.3, que todas as mensagens apresentam um identificador e que as mensagens que fazem parte da mesma *N*-upla (e_i^N ! *m)* têm o mesmo identificador retornado pela função ***id(m)***. Desta maneira, cada réplica $Filtro_n$ recebe (no máximo) *N* cópias de uma mensagem com o mesmo identificador, emitida no canal replicado e_i^N de SC_{ij} [2] . As regras impostas aos traços de um MR devem nortear o tráfego destas mensagens, ou seja, as mensagens são ordenadas segundo seus identificadores. A função de cada $FITRO_n$ é enviar uma só cópia de cada mensagem para a réplica *n* do processo replicação MR_j, destino da comunicação:

$$FILTRO_n = ult\_msg := 0;\ \mu X$$
$$((e'_n\ ?\ m) \rightarrow (X \not\leq (ult\_msg \leq idents(m) \wedge e'_n\ ?\ m \neq NULL) \not> (ult\_msg{:=}idents(m);\ c_j[n]\ !\ m;\ X))$$

Os processos filtros, da maneira como foram expressados, podem ser interpretados como exercendo a função do voto distribuído em hipóteses de falhas de omissão ou *crash*. A inclusão de mecanismos de voto no sistema de comunicação para modelos de grupo com réplicas ativas e sem privilégio retira a necessidade de um endereço de grupo exclusivo para o voto distribuído (SC_{ij}).

O voto em caso de faltas arbitrárias (voto majoritário) pode ser incluído nos processos Filtros. Nesta situação, para evitar a comparação bit-a-bit das mensagens de mesmo identificador até que a maioria seja alcançada, mecanismos de assinatura podem ser usados, limitando então estas comparações aos campos de identificação e a um certo número de bytes

---

[2] As regras impostas aos traços de um MR devem nortear o tráfego destas mensagens, ou seja, as mensagens são ordenadas segundo seus identificadores.

correspondente ao campo da assinatura. A assinatura é calculada no início da difusão. O esquema apresentado em [Davies 81], com o algoritmo DES [Denning 83] é uma possibilidade plausível; a assinatura produzida ocupa 8 bytes, que junto com o identificador da mensagem e o uso de CRC em protocolos de níveis mais baixos determinam o conjunto de informações e mecanismos de controle que garantem a integridade das mensagens que chegam nos processo FILTROS, durante a transmissões. O protocolo de difusão atômica proposto em [Cristian 85], na sua versão para faltas arbitrárias, envolve filtros com assinaturas criptografadas.

Os traços de saída de um MR ($\tau MR \lceil \omega MR$), como definidos no item 4.2.5, ficam difíceis de se garantir na presença de processos base faltosos. Considerando que os processos base (réplicas) que compõem o MR estão sincronizados dentro de um limite superior de defasagem de modo que se possa caracterizar como resultado de um ciclo de processamento a N-upla da forma $e^N\,!m$. A ocorrência de faltas descaracteriza esta situação.

No sentido de tratar com estas situações é definido um predicado ***non-faulty***(n) que deve sinalizar o comportamento do processo base de índice n e suas condições de saída. Então se esse predicado retorna *true* o processo base n é dito não faltoso. As mensagens produto de réplicas faltosas são expressadas pelo *token* NIL. Desta forma é possível N-upla no alfabeto de MR com componentes NIL. Diante disto, é necessário redefinir o alfabeto de MR na regra R3 do item 4.2.5:

$$\alpha(P^N) = \{(\overline{c}_{[1]}.m, \overline{c}_{[2]}.m, \ldots, \overline{c}_{[N]}.m) \mid \forall\, n.\ 1 \le n \le N \ \wedge$$

$$((\textbf{\textit{non\_faulty}} := \textit{true} \Rightarrow (\overline{c}_{[n]}.m = c_{[n]}.m) \wedge (c_{[n]}.m \in (\alpha P_n))$$

$$\wedge (m \in Msg) \vee (\textbf{\textit{non\_faulty}} := \textit{false} \Rightarrow \overline{c}_{[n]}.m = NIL)) \qquad (4.3)$$

onde, $\alpha(\mathrm{MR}) = \alpha(P_1{}^N) \cup \alpha(P_2{}^N) \cup \ldots \cup \alpha(P_k{}^N)$

No modelo, as mensagens NIL são assumidas como resultados do processamento de réplicas faltosas em qualquer situação de hipótese de faltas (omissão, *crash*, de temporização ou de valor). As faltas que possam ocorrer nos canais de entrada das réplicas (canais $c_i[n]$ da Figura 4.6) são incorporadas nas réplicas e portanto indicadas pelo predicado ***non-faulty*** na reexpressão de $\alpha(P^N)$ dada na equação (4.3). Do ponto de vista da replicação MR_j (receptora

da comunicação), devido ao uso de difusão atômica, é assumido então que as N-upla $(c_j^N ? m)$ são completas, sem componentes NIL.

Com base nas afirmativas e redefinições acima, chegamos a uma especificação do processo SC_{ij} como suporte de comunicação para réplicas ativas e sem privilégios, dada por:

$$SC_{ij} \ \textbf{Sat} \ \textit{elect}(t^N \lceil e_i^N \lceil \textit{ident}(x) = p) \ \wedge \textit{ident}(m) = p$$
$$\textbf{se} \ m = NIL \ \textbf{então} \ (c_{j[1]}.NIL, c_{j[2]}.NIL, \ldots, c_{j[N]}.NIL$$
$$\textbf{senão} \ (c_{j[1]}.m, c_{j[2]}.m, \ldots, c_{j[N]}.m)$$

ou seja, o processo SC_{ij} só produz uma N-upla $c_j^N ! m$ na situação em que a N-upla de saída do MR_i seja produto de réplicas ***non-faulty***, em número definido pelas hipóteses de faltas assumidas. Na condição de faltas arbitrárias é implementado nos filtros o voto majoritário que determina a necessidade de uma maioria de réplicas ***non-faulty*** para a produção de uma N-upla $c_j^N ! m$.

4.3.2 O MR como unidade de construção de SDRs

Ao tomarmos o MR como unidade de construção de um SDR (Sistema Distribuído Replicado), estaremos supondo que qualquer modelo de interação entre processos emissores e receptores possa ser montado. Na verdade algumas limitações são impostas aos modelos de processamento distribuídos quando expressados em termos de MRs.

Assumindo, por exemplo, o modelo **cliente/servidor**, pelas restrições impostas pelo CSP (comunicação unidirecional), necessitaríamos de dois endereços de grupo que permitiriam o fluxo de pedidos e de respostas nos dois sentidos.

O processo servidor atendendo mais de um cliente, necessitaria de mais de um canal (replicado) de entrada. Até o presente momento, por simplicidade, foi assumido que cada processo MR possui somente um canal de entrada. A adoção de mais canais (replicados) de entrada não implica em modificações nas definições de processos I/O. O teste de escalonamento pode então ser estendido como no MR servidor para representar duas portas de entrada (c^N e d^N):

$$(\ldots|(c^N ?\, m_i \rightarrow P_i^N \mid d^N ?\, m_i \rightarrow P_i^N)| \ldots| (c^N ?\, m_{i+1} \rightarrow P_{i+1}{}^N \mid d^N ?\, m_{i+1} \rightarrow P_{i+1}{}^N) \ldots)$$

No caso acima, as mesmas classes de algoritmos $(\ldots, P_i, P_{i+1}, \ldots)$ estão associadas às duas portas de entrada. Neste caso, uma mesma replicação pode ser ativada, indistintamente, por mensagens que ocorram ou num canal ou no outro. Porém, os servidores também podem atender portas com diferentes classes de algoritmos, e com isto teríamos:

$$(\ldots|c^N ?\, m_i \rightarrow P_i^N \mid c^N ?\, m_{i+1} \rightarrow P_{i+1}{}^N |\ldots|d^N ?\, m_j \rightarrow P_j^N \mid d^N ?\, m_{j+1} \rightarrow P_{j+1}{}^N \mid \ldots)$$

este servidor pode ser interpretado como um *alternate* de MRs.

Para expressar processos clientes em uma estrutura MR é necessário novas considerações. O processo cliente como iniciador de transação a priori não seria ativado por mensagem, o que descaracterizaria o teste de escalonamento nas entradas, e só uma classe de algoritmos pode ser associada a execução de um cliente. Uma maneira de se manter um cliente na forma de um MR é considerar a ativação do mesmo sempre por mensagens (correspondente a classe de algoritmos clientes). Esta mensagem teria o envio emulado a partir da ativação do MR cliente através dos *dispatchers* nos processadores que suportam as replicações destes clientes.

Mas o que parece uma restrição para a implantação de um MR cliente é o estilo de ***I/O process*** de suas réplicas. A definição de processo I/O, introduzida em 4.2.2, caracteriza o processamento que se inicia por operações de entrada e termina com operações de saída; não ocorrem interações fora dos pontos de entrada e de saída nestes processos. Se considerarmos o modelo cliente/servidor, verificamos que o cliente ao emitir um pedido ao servidor, tem a sua evolução subsequente dependente da mensagem de resposta. Uma maneira de conformar o recebimento de uma mensagem resposta ao estilo *I/O process* é considerar que esta mensagem escalona outra classe de algoritmo no MR cliente. A mensagem de resposta passaria então pelo teste de escalonamento ativando uma replicação que daria continuação ao processamento no cliente. A forma MR do cliente é dada então por:

$$(\ldots \mid c^N ?\, ativ \rightarrow P^N{}_{\text{inicio}} \mid d^N ?\, resp' \rightarrow P^N{}_{\text{resp}'} \mid \ldots \mid e^N ?\, resp'' \rightarrow P^N{}_{\text{resp}''} \mid \ldots)$$

o cliente tomaria a forma de um MR onde teríamos um canal (c^N) para ativação do cliente (ativação da replicação $P^N_{início}$) e um canal replicado (d^N) para a recepção das mensagens respostas provenientes do servidor que podem definir diferentes classes de algoritmos sobre este canal. Um MR cliente não necessariamente se limita a um só canal de ativação e também poderia ter outros canais replicados conectados a mais de um servidor o que então implicaria no teste de escalonamento envolvendo a exemplo do que já ocorria com MR servidor com vários canais (replicados) de entrada.

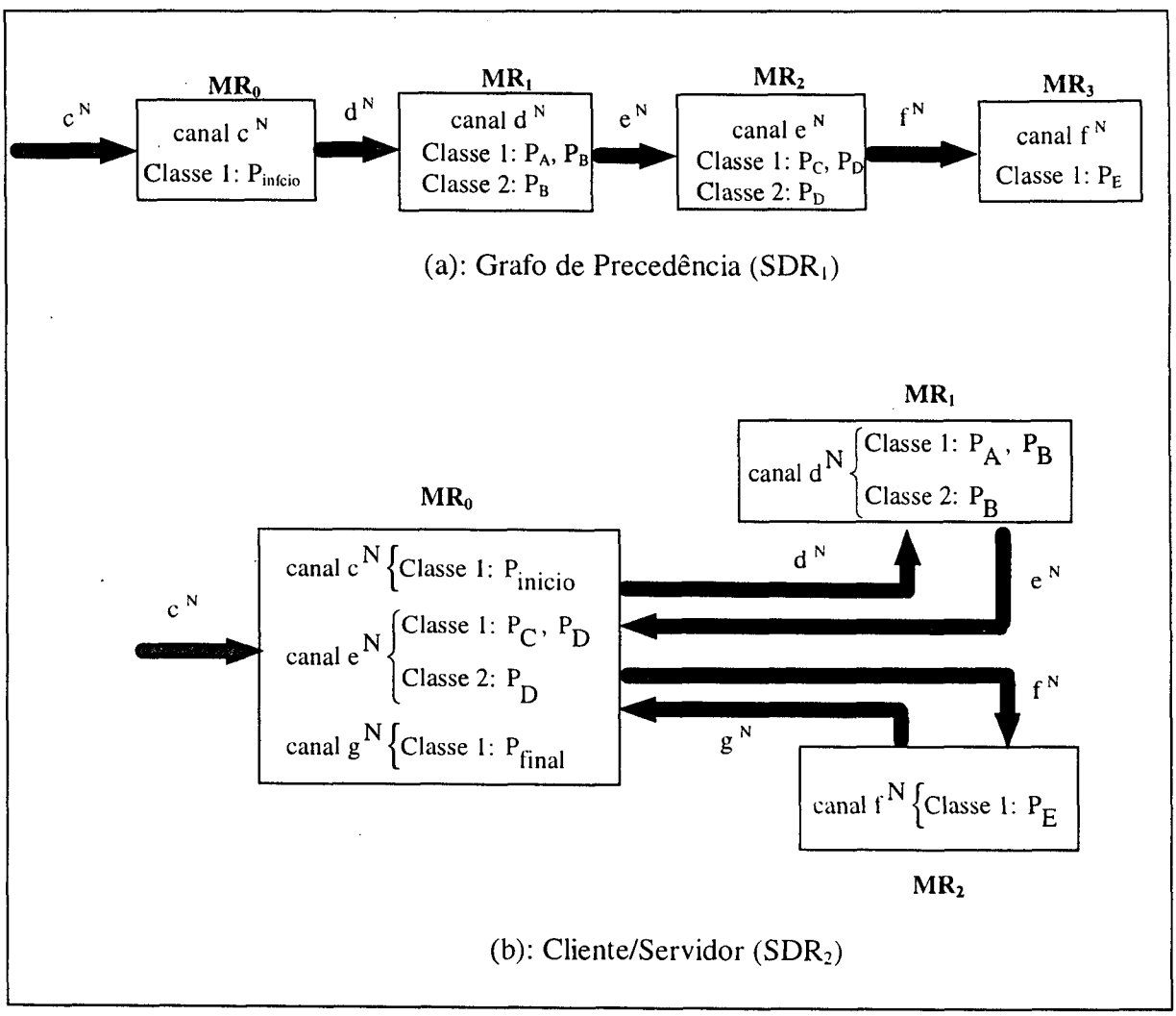


Figura 4.6: Execução de um SDR na forma de (a) grafo de precedência e (b) cliente/servidor

Um outro modelo de processamento em que o MR como unidade de processamento se adaptaria de maneira adequada seria na forma de grafos de precedência (ou modelo *data-driven* [Mori 86]). Estes grafos, acíclicos, determinam a ordem de ativações dos nós MRs.

Neste modelo a execução do grafo se repete ciclicamente, dando-se numa seqüência de MRs, da raiz até as folhas da árvore. O sistema MARS [Kopetz 89b] apresenta um modelo de processamento que se fundamenta em grafos de precedência. A figura 4.6.(a) apresenta um SDR na forma de ***grafos de precedência*** caracterizado por um só caminho; isto se deve ao fato que os MRs deste SDR apresentam só uma opção de canal (replicado) de entrada e uma de saída.

O nó raiz de um grafo, a exemplo do cliente no modelo cliente/servidor, deve possuir um canal de ativação com somente uma classe de algoritmos associada. A figura 4.6(a) mostra o grafo de precedência, indicando os algoritmos a serem executados no forma de replicações em cada nó MR e a disposição das classes destes algoritmos por canal replicado. Os mesmos algoritmos são agrupados na forma do modelo cliente/servidor na figura 4.6(b). Por questão de simplicidade em ambas figuras (4.6(a) e 4.6(b)) foram omitidas as representações dos processos SC_{ij}. As expressões de dois SDRs, compostos a partir dos mesmos algoritmos, nos dois modelos são indicados abaixo:

$$SDR_1 = (c^N ?\ ativ \rightarrow P_{início}{}^N) \parallel SC_{01} \parallel (d^N ?\ m_A \rightarrow P_A{}^N \mid d^N ?\ m_B \rightarrow P_B{}^N) \parallel SC_{12} \parallel (e^N ?\ m_C \rightarrow P_C{}^N \mid e^N ?\ m_D \rightarrow P_D{}^N) \parallel SC_{23} \parallel (f^N ?\ m_E \rightarrow P_E{}^N)$$

$$SDR_2 = (c^N ?\ ativ \rightarrow P_{início}{}^N \mid e^N ?\ m_C \rightarrow P_C{}^N \mid e^N ?\ m_D \rightarrow P_D{}^N \mid g^N ?\ m_{final} \rightarrow P_{final}{}^N) \parallel SC_{01} \parallel SC_{10} \parallel (d^N ?\ m_A \rightarrow P_A{}^N \mid d^N ?\ m_B \rightarrow P_B{}^N) \parallel SC_{02} \parallel SC_{20} \parallel (f^N ?\ m_E \rightarrow P_E{}^N)$$

A princípio podemos afirmar que a segunda expressão CSP estimula mais o paralelismo. O desempenho dos SDRs depende destes modelos de estruturação de seus processamentos; mas também é verdade que o desempenho depende igualmente dos modelos de execução destes programas distribuídos. No capítulo 6 é discutido um modelo de execução assíncrono temporizado para o esquema MR.

Muitas das restrições que aparecem na formalização do MR estão ligadas nos próprios conceitos da ferramenta utilizada, o CSP, e desaparecem ou são simplificados nas implementações.

A garantia de correção global nos SDRs, além dos aspectos de correção dos seus MRs, depende de interações livres de *deadlocks* entre seus MRs componentes. Em [Hoare 85] são examinados em vários modelos de processamento as condições de interações sem *deadlocks*. As mesmas condições podem ser aplicadas em composições de MRs.

4.4 Considerações sobre o MR em relação a outras abordagens de flexibilidade

Se comparamos a abordagem de flexibilização utilizada no MR com as apresentadas em [Cristian 94],[Little 94] ou mesmo no SCOP, veremos que essas últimas têm com como base a variação do número de réplicas. Essas variações no número de réplicas tem como objetivo manter requisitos de qualidade de serviço (desempenho, disponibilidade, etc.). Funções de gerenciamento estabelecem mudanças de configuração nos serviços confiáveis conforme as medidas de desempenho ou disponibilidade se afastem de requisitos estabelecidos.

Nesse trabalho, não tratamos com este tipo de flexibilidade. A técnica empregada no MR define classes de algoritmos a partir do teste de escalonamento. Por exemplo, se tivermos expectativas quanto ao desempenho do serviço replicado, ou se o processamento do mesmo estiver sujeito a requisitos temporais, o teste de escalonamento pode ser montado de modo a selecionar os algoritmos conforme medidas de desempenho na evolução do processamento no sistema. Se as medidas indicam um afastamento dos requisitos estabelecidos, os algoritmos selecionados serão os menos precisos e mais rápidos no lugar dos algoritmos mais lentos e mais elaborados. Ou seja, a técnica de flexibilidade usada no MR não altera o número de réplicas, mas sim a precisão dos resultados quando a qualidade desejada do serviço não é alcançada.

4.5 Conclusão

Neste capítulo foi proposto o esquema de tolerância a faltas denominado de Múltiplas Replicações ou simplesmente MR que explora o conceito de grupo e as próprias características de uma aplicação. Este esquema foi introduzido no sentido de permitir estruturas mais flexíveis, adaptáveis as condições de um sistema distribuído.

O MR é formado a partir de múltiplas replicações de processos que são usadas no esquema no sentido de atender os princípios tanto da diversidade de projeto quanto da diversidade de dados. Os dados de entrada apresentam atributos que permitem escalonar replicações adequadas ao processamento dos mesmos.

O esquema MR apresenta também um estrutura bastante flexível que permite a sua fácil adaptação diante da evolução de uma aplicação distribuída. Novos algoritmos e por conseqüência novas classes de algoritmos podem ser agregados na hierarquia de especialização existente, sem que isso represente a reprogramação total do esquema implementado. Essa característica é mais a ativação de réplicas a partir dos atributos dos dados de entrada definem o MR como um instrumento poderoso na construção de servidores tolerantes a faltas, com características de programas adaptáveis como os sugeridos em [Parker 96].

O MR foi apresentado nesse capítulo através do Modelo de Traços CSP [Hoare 85] e da Teoria de Processos Replicados [Mancini 88]. Algumas extensões foram propostas à teoria de Mancini com o intuito de simplificar a apresentação do MR. A composição de processos MR, formando, um Sistema Distribuído Replicado (SDR) foi também explorada nesse texto.

Nos próximos capítulos o modelo proposto é comparado com outros esquemas no sentido de explicitar suas eventuais qualidades. Modelos de execução para o MR são também objeto de discussão na seqüência desse texto.

CAPÍTULO 5

ANÁLISE DE PERFORMABILIDADE DO ESQUEMA MR

No segundo capítulo foram apresentados vários esquemas de tolerância a faltas. Muitos destes esquemas têm sido objeto de avaliação do ponto de vista da Segurança de Funcionamento em vários estudos apresentados na literatura. Estes trabalhos de avaliação ou tratam da introdução de uma nova abordagem ou metodologia de avaliação, normalmente testada numa análise comparativa de vários esquemas de tolerância a faltas conhecidos [Tai 93a][Laprie 90][Scoth 87] ou, ao contrário, descrevem o uso de uma abordagem já conhecida no sentido de justificar a proposição de um novo esquema.

Inúmeras técnicas de avaliação têm sido introduzidas no sentido da validação da Segurança de Funcionamento de esquemas de tolerância a faltas. Em [Nacamura 94], uma abordagem usando simulação numérica é apresentada no sentido de um estudo comparativo, que permitiu quantificar a eficiência do MR em relação a esquemas como o RB, PNV, CRB e outros.

Uma outra abordagem, conhecida como análise de performabilidade, vêm sendo bastante difundida nos últimos anos. A **performabilidade** (*performability*) [Meyer 80] é uma composição de grandezas que quantificam o Desempenho e a Segurança de Funcionamento. A análise de performabilidade vêm sendo empregada para diversas finalidades: em [Tai 93a] e [Chiaradonna 94], esta metodologia de análise é usada na comparação de vários esquemas de tolerância a faltas; em [Reibman 90] a performabilidade é um parâmetro de referência para medir o efeito da introdução de mecanismos de tolerância a faltas no desempenho de aplicações distribuídas; em [Meer 93] é proposta uma abordagem de análise de performabilidade para avaliação de sistemas reconfiguráveis.

O objetivo deste capítulo é apresentar um estudo comparativo do esquema MR com outros esquemas da literatura. Este estudo tem como base modelos analíticos que permitem o cálculo da performabilidade dos esquemas considerados.

5.1 Medidas e modelos de avaliação

O aumento cada vez maior da complexidade no desenvolvimento de sistemas computacionais tem feito com que a análise e a avaliação do projeto destes sistemas sejam feitas através de modelos. Um **modelo** corresponde a uma abstração do comportamento de um sistema que é construído a partir das hipóteses de funcionamento (operação) do próprio sistema. Estas hipóteses são representadas através de relações matemáticas ou lógicas que se forem bastante simples, podem ser resolvidas analiticamente. A dificuldade de se obter uma solução analítica aumenta quanto maior for o realismo introduzido na descrição do sistema. Em situações de complexidade muito grande, a simulação surge como alternativa viável na avaliação de um sistema. Quando a análise do modelo, usando soluções analíticas ou simulações não é suficiente ainda para garantir ou obter propriedades desejadas, resultados suplementares podem ser obtidos a partir de testes realizados usando protótipos do sistema.

De uma maneira geral, as metodologias usadas em processos de avaliação de um sistema, através de modelos, correspondem a um processo iterativo que envolve etapas como [Reibman 91]:

(1) Escolher as medidas a serem avaliadas com base nos objetivos gerais do sistema;
(2) Construir o modelo;
(3) Avaliar o sistema através do modelo;
(4) Refinar o modelo até que este seja preciso;
(5) Usar os resultados para melhorar o projeto e facilitar a tomada de decisões;
(6) Atualizar o modelo refletindo as evoluções do projeto.

No estudo a ser apresentado neste capítulo, a metodologia usada no processo de avaliação do esquema MR se concentra nas primeiras três etapas citadas acima.

5.1.1 Medidas da Segurança de Funcionamento e do Desempenho

Normalmente, as propriedades de Segurança de Funcionamento de um sistema são especificadas através das medidas de confiabilidade (*reliability*) e de disponibilidade (*availability*) [Laprie 92]. A **confiabilidade** de um sistema é a medida que representa a probabilidade do sistema operar adequadamente sem falha, até um determinado instante *t*. Neste caso, a confiabilidade do sistema pode ser expressa através de uma função *R(t)*. Uma outra forma de expressar a confiabilidade de um sistema é através do **Tempo Médio Para Falha** (MTTF - *Mean Time To Failure*) que corresponde ao tempo médio esperado até a ocorrência de uma falha do sistema.

A **disponibilidade instantânea** (*instantaneous availability*) é a probabilidade do sistema estar operacional num dado instante *t*, e normalmente é expressa através de uma função *A(t)*. A disponibilidade tem um sentido prático quando analisada por longos períodos ($t \rightarrow \infty$) e neste caso é denominada como **disponibilidade estacionária** (*steady state availability*) ou, simplesmente como **disponibilidade**. De maneira informal, a disponibilidade de um sistema pode ser expressa pela fração de tempo em que o serviço do mesmo é operacional, durante o período em que o sistema foi especificado para ser operacional. A disponibilidade é usada freqüentemente como medida de referência na avaliação de sistemas reparáveis, cujos serviços se caracterizam por transições entre os estados **serviço não operacional** e **serviço operacional** diante das ocorrências de eventos **falha** e **recuperação**, respectivamente.

As técnicas de tolerância a faltas são baseadas no uso de redundâncias de software e/ou de hardware ocasionando assim um *overhead* extra durante a operação do sistema. Além disto, nestes sistemas a falha de um componente pode não causar a falha total do sistema, mas reduzir a capacidade de processamento ou desempenho destes serviços (operação degradada). As medidas de desempenho têm por objetivo quantificar a qualidade do serviço do ponto de vista temporal. As medidas de desempenho, normalmente, utilizadas são: o tempo médio de execução de um serviço (*average response time*), a taxa efetiva de execução do serviço (*throughputs*) e a taxa de utilização do serviço (*utilization*) [Menascé 94].

O **tempo médio de execução de um serviço** é uma medida tomada do ponto de vista do cliente e corresponde ao tempo médio que o cliente espera pela execução do serviço. O ***throughput*** também é uma medida de desempenho que quantifica a qualidade do serviço

fornecido pelo sistema; por exemplo, o *throughput* de um sistema pode ser medido a partir do valor médio de execuções do serviço por unidade de tempo. A **taxa de utilização do serviço** é a relação entre a soma dos tempos de execução do serviço em um período considerado, sobre a duração deste período.

A medida de performabilidade apresentada em [Meyer 80] combina quantificações de Segurança de Funcionamento e de Desempenho. Na interpretação apresentada em [Tai 93a], a performabilidade de um sistema corresponde ao número médio de execuções corretas do serviço (sistema), durante um período de tempo *t* com a condição de que não ocorra nenhuma falha catastrófica.

5.1.2 Tipos de modelos

Os modelos de avaliação de sistemas tolerantes a faltas podem ser classificados segundo as medidas utilizadas para quantificá-los ou ainda pelas formas de representação usadas nestes modelos.

Modelos classificados segundo as medidas utilizadas

As abordagens que utilizam a confiabilidade como parâmetro de referência de análise do sistema são denominadas de **modelos de Confiabilidade**. Por outro lado, as abordagens que analisam a disponibilidade são denominadas de **modelo de Disponibilidade**. O termo **modelo de Segurança de Funcionamento** é usado para referenciar genericamente as duas abordagens citadas.

As técnicas que analisam o desempenho de um sistema, segundo uma das medidas apresentadas no item 5.1.1, são denominadas de **modelos de desempenho**. Já as abordagens que analisam a combinação das medidas de desempenho com a de Segurança de Funcionamento são denominados de **modelos de Performabilidade**.

Modelos classificados segundo a forma de representação

No que se refere principalmente a Segurança de Funcionamento, as formas mais usuais de se representar sistemas (modelos de avaliação) são os grafos de confiabilidade, árvores de falhas e modelos *state-space*.

O grafo de confiabilidade é uma das formas mais simples de modelar o comportamento do sistema e é utilizado especificamente na determinação da confiabilidade do sistema modelado. Os grafos de confiabilidade são usados para modelar a interconexão física ou a dependência de falhas em sistemas. Os modelos de interconexão física de sistemas são construídos a partir de grafos de confiabilidade probabilistas, onde os nós representam os componentes do sistema e os arcos as ligações entre estes componentes. A probabilidade de falha pode ser designada aos nós e/ou aos arcos. Este tipo de modelo é usado, normalmente, na resolução de problemas relacionados com a confiabilidade de redes de comunicação. A figura 5.1(b) apresenta o grafo de conexão física de um sistema TMR.

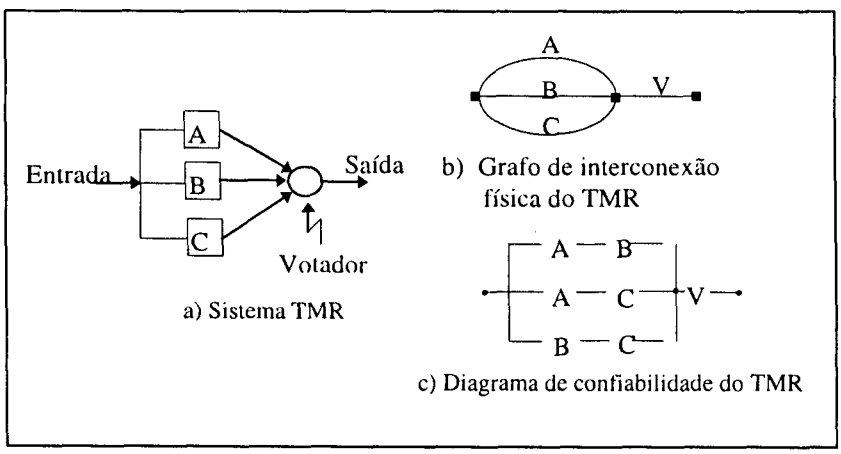


Figura 5.1: Grafo de confiabilidade de um sistema TMR

Por outro lado, a modelagem de dependência de falhas, também conhecida, como grafos de sucesso ou diagrama de confiabilidade, determina a dependência operacional em conjuntos de componentes. Cada caminho no grafo descreve uma seqüência de dependências entre componentes de um sistema. Num diagrama de confiabilidade, dependendo do sistema, um componente pode se apresentar em mais de um caminho. A figura 5.1(c) mostra um diagrama de confiabilidade de um sistema TMR.

Uma outra forma de modelar um sistema, usada também para o cálculo da confiabilidade do mesmo, é a técnica de **árvore de falhas**. Nesta abordagem os eventos que provocam uma falha do sistema são enumerados. Por exemplo, a árvore de falhas do sistema composto por dois processadores (p_1 e p_2) em série e mais uma fonte de alimentação (A) é mostrada na figura 5.2(b). O serviço do sistema é falho quando um dos processadores ou a fonte de alimentação falha. A árvore de falha é bastante utilizada na análise de sistemas de hardware

pela simplicidade de representação do modelo. Entretanto, a principal limitação desta técnica está na dificuldade de expressar situações mais complexas, tais como, falhas de modo comum ou interdependência de faltas. Este modelo como muitos baseados em análise combinatória não representam em detalhes muitos dos aspectos envolvendo a Segurança de Funcionamento o que pode influenciar fortemente o cálculo da confiabilidade ou a cobertura de faltas, por exemplo.

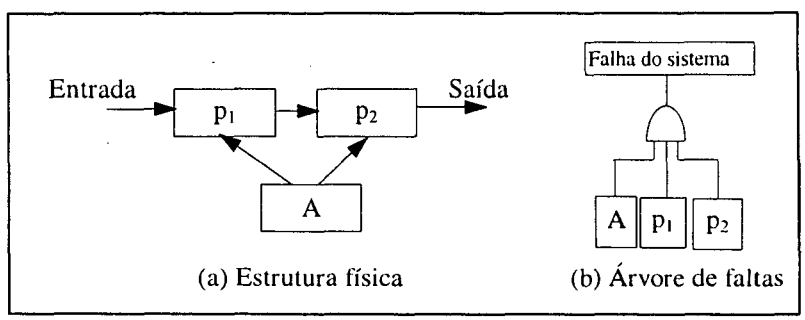


Figura 5.2: Árvore de falta de um sistema em série

Uma forma genérica de modelar a Segurança de Funcionamento, o Desempenho e a Performabilidade de um sistema é através de ***modelos state-space***. O principal exemplo de modelos *state-space* é a cadeia de Markov. Neste caso, um conjunto de variáveis aleatórias, denominadas $\{X(t) \mid t \in T\}$, define um processo estocástico. Considerando que $X(t)$ representa estados de um sistema onde t assume instantes pertencentes ao período discreto ou contínuo T, a variável $P\{X(t)\}$ corresponde a probabilidade do sistema encontra-se num dos estados possíveis $X(t)$. Os estados de um processo estocástico representa uma cadeia (processo) de Markov, se todas as condições abaixo são satisfeitas:

(1) Existe um número finito de estados;
(2) A probabilidade condicional do sistema atingir um estado futuro é determinado somente pelo estado presente, independente de qualquer estado anterior;
(3) A probabilidade da transição de um estado para outro é independente do tempo;
(4) Para todos os estados do sistema é associado, inicialmente, um conjunto de probabilidades.

Normalmente, os modelos *state-space* são utilizados na descrição de sistemas tolerância a faltas por permitirem representar o reparo e a reconfiguração do sistema quando da ocorrência de uma falha e a cobertura envolvida nos mecanismos de detecção de erros presentes nestes

sistemas. Além disto, a utilização de ferramentas, tais como SURF [Costes 81], SHARPE [Sahner 86], etc, facilitam a análise de modelos *state-space*. As descrições destas ferramentas e das técnicas utilizadas na análise do modelo de Segurança de Funcionamento estão fora do escopo deste trabalho, podendo ser encontradas em [Johnson 88] e [Geist 90].

5.2 Trabalhos relacionados

A literatura técnica apresenta vários trabalhos sobre modelagem e avaliação de esquemas de tolerância a faltas. Entre estes trabalhos, nós nos resumimos a uns poucos que reputamos como merecedores de citação neste seção.

Em [Scoth 87], as confiabilidades de vários esquemas são examinados a partir da construção de grafos de confiabilidade. Os modelos foram construídos com base no número de componentes operacionais (*up*) de cada esquema. Nas representações destes esquemas, os estados estão associados com as execuções dos componentes e as transições estão relacionadas com as probabilidades do sistema passar de um estado para outro. Neste trabalho são construídos modelos de confiabilidade para os esquemas Bloco de Recuperação (RB), Programação N-Versões (PNV) e Bloco de Recuperação em Consenso (CRB). Diferentes hipóteses de faltas foram assumidas para cada esquema, por exemplo, presença ou não de faltas relacionadas entre as versões de algoritmos de aplicação dos esquemas. Nestes estudos, RB e o CRB apresentam melhor confiabilidade que o PNV. Além disto, o esquema CRB, por fazer uso de um teste de aceitação em situações quando não é possível obter um resultado de consenso na votação, é mais confiável do que o RB.

Em [Laprie 90] são analisadas várias arquiteturas (RB/1/1, PNV/1/1, NCSP/1/1, etc.) que são construídas a partir dos esquemas de tolerância a faltas RB, PNV e NCSP. Estas arquiteturas foram analisadas a partir de um modelo genérico de comportamento composto por apenas 4 estados que representam: o estado inicial, o estado degradado após uma falha de hardware, estado com ocorrência de erro detectado e estado com ocorrência de um erro não detectado. As transições deste modelo genérico estão associadas com as taxas de falhas que correspondem a probabilidade do sistema passar de um estado para outro. A principal conclusão neste trabalho é que as taxas de falhas dos algoritmos (versões) influenciam de maneira idêntica a confiabilidade das três arquiteturas analisadas no trabalho.

Em [Hudak 93] um trabalho completo de avaliação e comparação de técnicas de software tolerante a faltas é apresentado. Os esquemas analisados são: Algorítmica Tolerante a Faltas (AFT: *Algoritmic Fault-Tolerance*), Detecção de Erros Concorrentes (CED: *Concurrent Error-Detection*), Programação N-Versões (PNV) e Bloco de Recuperação (RB). A Algorítmica Tolerante a Faltas corresponde a construção de algoritmos de aplicação com mecanismos de tolerância a faltas embutidos nos mesmos. A Detecção de Erro Concorrente corresponde a técnica onde erros são detectados a partir de mecanismos que se executam concorrentemente à aplicação. Técnicas de *watchdog* são exemplos desta detecção de erros concorrente.

A abordagem de avaliação usada em [Hudak 93] é baseada em modelos *state-space*. Na metodologia usada, foram definidos experimentos no sentido de levantar parâmetros como probabilidade de detecção de erros produzidos por faltas de hardware, probabilidade de detecção de erros produzidos por faltas de software, probabilidade de recuperação após a detecção de erro, etc. Os experimentos foram constituídos, nas implementações a partir de uma especificação de aplicação, dos quatro esquemas citados acima. Os parâmetros foram obtidos nos experimentos por meio de testes (injeção de faltas).

1 2 3 4 5

melhor pior

	PNV	RB	CED	AFT
Detecção de erros	5	2	4	1
Recuperação de erros	1	4	---	2
Aborto	--	3	1	3
Correção	1	5	2	2
Disponibilidade	1	4	4	3
MTTF	1	4	4	4

Tabela 5.1: Comparação entre esquemas de tolerância a faltas

Um modelo de Markov Discreto genérico é usado para a partir dos parâmetros de cada experimento gerar os modelos de cada esquema em estudo. Com base nestes modelos, as eficiências dos modelos implementados, segundo parâmetros de disponibilidade, MTTF e

correção[1], recuperação de erro, etc, foram comparadas entre si e também com resultados dos experimentos realizados. Algumas conclusões obtidas neste trabalho são explicitadas na tabela 5.1. Os dados contidos na tabela indicam que a detecção de erros do RB é melhor que a do PNV, por exemplo. Por outro lado, o mecanismo de recuperação do esquema PNV é melhor que o dos outros esquemas analisados.

Em [Tai 93a] e [Chiaradonna 94] os esquemas de tolerância a faltas são modelados usando os conceitos de performabilidade no sentido da avaliação e comparação dos mesmos. Estes estudos serão retomados na seqüência deste capítulo.

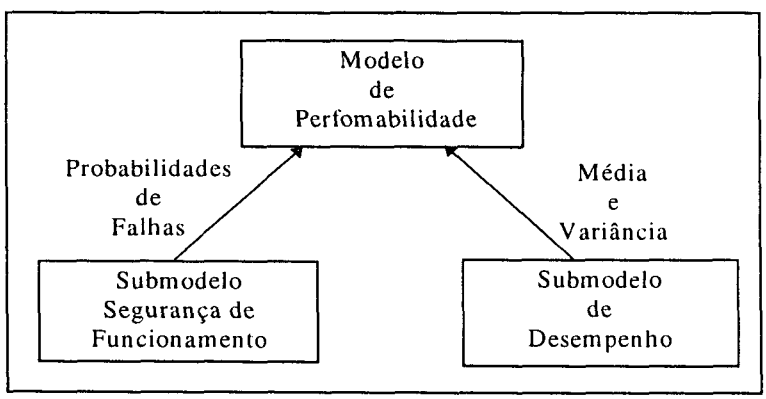


Figura 5.3: Estrutura hierárquica na análise de performabilidade

5.3 Análise de performabilidade

A performabilidade de um sistema é calculada a partir de seu Modelo de Performabilidade, cujos parâmetros que representam as propriedades de Segurança de Funcionamento e de Desempenho do sistema são obtidos, respectivamente, através dos submodelos de Segurança de Funcionamento e de Desempenho. A figura 5.1 ilustra a estrutura hierárquica existente entre os submodelos de Desempenho e de Segurança de Funcionamento com o modelo de Performabilidade.

---

[1] A correção é definida como uma medida em [Hudak 93] e corresponde a probabilidade de que um resultado incorreto não ser produzido em uma ativação de serviço:

correção = $(1 - P_f)$

onde P_f = **Pr** {transição para um estado *Falha do Serviço*}

A construção de modelos de performabilidade para esquemas de tolerância a faltas, segundo a abordagem proposta em [Tai 93a], é feita de maneira sistematizada e envolve a construção dos submodelos de Segurança de Funcionamento e de Desempenho. Estes submodelos são construídos a partir da descrição funcional do sistema (ou esquema) a ser analisado. Para isto, assume-se que o sistema apresente uma natureza iterativa, ou seja, cada execução do serviço do sistema corresponde a uma iteração. O tempo total de análise, denominado de missão, corresponde a uma série de execuções (iterações) sucessivas do sistema. No início de cada iteração (execução) o sistema recebe uma entrada e produz um resultado no final da iteração que é função apenas desta entrada. Do ponto de vista temporal, cada resultado deve ser fornecido dentro de um *deadline*. As violações deste *deadline* são detectadas através de um *watchdog*. Os resultados gerados dentro de *deadlines* estabelecidos são considerados corretos do ponto de vista temporal, e são analisados no domínio dos valores:

- **execução correta:** quando o resultado correto é aceito pelos mecanismos de detecção erro;
- **execução incorreta com faiha benigna por erro de valor:** o resultado correto não é aceito pelos mecanismos de detecção de erro;
- **execução incorreta com falha catastrófica:** quando um resultado errôneo é aceito pelos mecanismos de detecção de erro. O resultado errôneo é então assumido como correto.

Os resultados não fornecidos dentro dos *deadlines* esperados caracterizam erros de temporização. Neste caso, o sistema apresenta uma **execução incorreta com falha benigna por erro de temporização**. As execuções incorretas com falhas benignas (por erro de valor ou de temporização) determinam por parte do serviço ou o não envio de resultado ou o envio de um resultado *default*.

5.3.1 Submodelo de Segurança de Funcionamento

O submodelo de Segurança de Funcionamento do sistema em análise é modelado por um processo de Markov de estado finito e a evento discreto. A construção deste modelo é feita com base nas hipóteses de faltas assumidas para o sistema. Neste modelo, os estados estão

relacionados a execuções de componentes do sistema e as transições representam as probabilidades do sistema passar de um estado para outro.

Através do submodelo de Segurança de Funcionamento é possível obter as probabilidades de execução correta (p_c), de execução incorreta com falha benigna por valores (p_{fbv}) e de execução incorreta com falha catastrófica (p_{fc}). Os valores destas probabilidades são passados ao modelo de Performabilidade para a análise de performabilidade do sistema a ser avaliado.

5.3.2 Submodelo de Desempenho

O submodelo de Desempenho é construído com o objetivo de se calcular o tempo médio de uma execução (uma iteração) do serviço do sistema. O tempo de duração de uma iteração é um evento aleatório, representado pela variável *y*, que varia com o número de componentes do sistema que tomam parte do processamento em uma iteração e do tempo de execução de cada componente.

Duas funções densidade de probabilidade (*fdps*) são introduzidas para representar a distribuição dos valores da variável aleatória *y* associada ao tempo de execução do sistema (ou do serviço do sistema). A primeira é a $f_{sist}(y)$ que corresponde a *fdp* do tempo de execução do sistema sem considerar o *deadline*. A segunda função densidade de probabilidade da variável *y*, representada por $f(y)$, é associada aos valores de tempo de execução do sistema considerando o *deadline* τ ($\tau > 0$). A função $f(y)$ é calculada a partir da função densidade $f_{sist}(y)$ e do *deadline* τ estabelecido para uma iteração (execução).

Assumindo as variáveis aleatórias que representam os tempos de execução dos componentes do sistema como apresentando uma distribuição exponencial, as *fdps* destas variáveis e também $f_{sist}(y)$ são então funções exponenciais. Se a execução do sistema (iteração) termina antes do *deadline* estabelecido ($y < \tau$), a função densidade de probabilidade $f(y)$ é igual a $f_{sist}(y)$. Entretanto, se $y > \tau$, $f(y)$ corresponde a um impulso (δ) em $y=\tau$ com amplitude $1-\int_0^{\tau} f_{sist}(y)dy$, uma vez que $\int_{-\infty}^{\infty} f(y)dy = 1$ para qualquer *fdp*. A amplitude do impulso corresponde a **probabilidade de execuções incorretas com falhas benignas por erro de temporização** (p_{fbt}). A partir destes resultados, a função densidade de probabilidade $f(y)$ é calculada pela equação:

$$f(y) = \begin{cases} f_{sist}(y) & y < \tau \\ p_{fbt}\delta(y-\tau) & y \geq \tau \end{cases} \quad (5.1)$$

A média (μ) e variância (σ^2) do tempo de duração de uma iteração (execução do serviço do sistema) são calculadas com base na função densidade de probabilidade *f(y)*. Os valores obtidos de μ e de σ^2 no submodelo de Desempenho são passados a camada superior para o cálculo da performabilidade.

5.3.3 Modelo de Performabilidade

A performabilidade de um sistema é calculada com base na duração de uma missão, sendo que neste cálculo os seguintes fatores são considerados:

- as probabilidades relacionadas com a execução do serviço do sistema: a probabilidade de execução correta (p_c), as probabilidades de execução incorreta por falha benigna por erro de valor e de temporização (p_{fbv} e p_{fbt}) e a probabilidade de execução incorreta por falha catastrófica (p_{fc});
- o número de iterações do sistema (execuções) durante a missão.

Nós próximos itens serão apresentados os procedimentos que permitem calcular a performabilidade de um sistema com base nos parâmetros obtidos nos submodelos de Segurança de Funcionamento e de Desempenho.

5.3.3.1 Variável de performabilidade *(M_t)*

A performabilidade de um sistema numa missão de duração *t* corresponde ao número total de execuções corretas do sistema com a condição de que não ocorra nenhuma execução incorreta com falha catastrófica durante a missão. A ocorrência de uma falha catastrófica na execução do serviço faz com que todo o benefício obtido anteriormente na missão de um sistema seja perdido.

A performabilidade de um sistema é representada por uma variável aleatória notada como M_t, que contabiliza o número de execuções corretas do sistema numa missão, na condição que nenhuma falha catastrófica ocorra. O valor da variável M_t é incrementado de uma unidade a cada iteração correta ($M_t = M_t + 1$), permanece o mesmo quando a execução é incorreta com

falha benigna (por erro de valor e de temporização) e passa a ser zero ($M_t = 0$) no caso de uma execução incorreta com falha catastrófica. As variáveis aleatórias que descrevem os processos bases e servem de suporte para o cálculo de M_t são:

I_t : número de execuções corretas no intervalo de $[0,t]$;

D_t : número de execuções incorretas com falhas benignas no intervalo de $[0,t]$;

N_t : número de execuções incorretas com falhas catastróficas no intervalo de $[0,t]$.

As variáveis I_t, D_t e N_t são simples contadores usados na determinação de M_t . Por exemplo, se N_t é positivo significa que ocorreu no mínimo uma iteração incorreta com falha catastrófica durante a missão, sendo assim a variável de performabilidade assume $M_t = 0$, embora I_t seja positivo. A variável de performabilidade M_t é expressa com base nos valores de I_t e N_t da seguinte forma:

$$M_t = \begin{cases} I_t & se\ N_t = 0 \\ 0 & se\ N_t \neq 0 \end{cases}$$

A performabilidade de um sistema é calculada pelo valor médio da variável M_t e é representada por $E[M_t]$ [Tai 93a].

5.3.3.2 Número total de iterações durante uma missão (K_t)

Durante uma missão várias execuções do sistema são realizadas, sendo que o tempo de execução do sistema é um evento aleatório *(y)*. Considerando que uma iteração (execução) começa imediatamente após o término da anterior, os instantes de início de cada execução do sistema durante a missão podem por sua vez, ser representados como variáveis aleatórias independentes. Desta forma, a seqüência de iterações do sistema é modelada através de um ***processo de renewal*** [Trivedi 82], e representado por:

$$K = \{K_t \mid t \geq 0\}$$

onde, K_t corresponde ao número total de iterações ocorridas no intervalo de $[0,t]$.

Com base na definição da variável de performabilidade M_t o valor de K_t em qualquer instante t é igual a soma das variáveis I_t, D_t e N_t.

5.3.3.3 Cálculo da performabilidade *(E[M$_t$])*

Considerando que a duração t de uma missão, normalmente definida em horas, é muito maior que o tempo médio (μ) de uma execução, geralmente na faixa de *milisegundos*, e que os valores de μ e σ^2 da distribuição dos tempos de execução do sistema são finitos, pelo teorema do limite central [Meyer 83] a distribuição da variável K_t se aproxima da distribuição normal quando $t \rightarrow \infty$. Desta forma, com base nos valores de tempo médio (μ) e de variância (σ^2) de uma iteração do sistema obtido pelo submodelo de Desempenho é possível calcular os valores de média e variância de K_t [Pargen 62]:

$$E[K_t] = \frac{t}{\mu} \qquad Var[K_t] = \frac{t\sigma^2}{\mu^3} \qquad (5.2)$$

A figura 5.4 ilustra a estrutura do modelo de performabilidade de um sistema. Os dados internos ao retângulo correspondem ao valor médio de K_t e a sua variância que foram calculados a partir dos valores de μ e σ^2 obtidos no submodelo de Desempenho. As linhas que ligam a saída à entrada do retângulo representam os diferentes resultados que podem ser obtidos numa execução do sistema e estão relacionados com as probabilidades de execução correta (p_c), de execução incorreta com falha benigna (p_{fb}) e de execução incorreta com falha catastrófica (p_{fc}). A probabilidade p_{fb} corresponde a soma das probabilidades p_{fbv} e p_{fbt} que são, respectivamente, as probabilidades de execução incorreta com falha benigna por erro de valor e por erro de temporização.

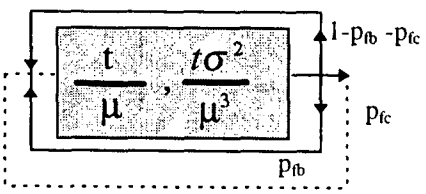


Figura 5.4: Modelo de Performabilidade de um sistema

O valor da variável M_t corresponde ao número total de execuções (iterações) que retorna a entrada pelo caminho ($1-p_{fb}-p_{fc}$) da figura 5.4, com a condição que nenhuma iteração retorne pelo caminho p_{fc} durante a missão. Desta forma, a função geratriz de momento da variável M_t corresponde a [Tai 91]:

$$E[e^{sM_t}] = E[(1-(1-p_{fc})^{K_t} + (p_b + (1-p_{fb}-p_{fc})e^s)^{K_t}] \qquad (5.3)$$

$$E[M_t] = \left.\frac{dE[e^{M_t s}]}{ds}\right|_{s=0} = \frac{1 - p_{fb} - p_{fc}}{1 - p_{fc}} E[K_t (1 - p_{fc})^{K_t}] \qquad (5.4)$$

Como K_t apresenta uma distribuição normal com média (t/μ) e variância $(t\sigma^2/\mu^3)$, temos que:

$$\lim_{t \to \infty} \text{Prob}\left\{ \frac{K_t - t/\mu}{\sqrt{t\sigma^2/\mu^3}} < x \right\} = \varphi(x) \qquad (5.5)$$

onde, $\varphi(x)$ é a *fdp* da distribuição normal. Se considerarmos que $\hat{\mu} = 1/\mu$, $\hat{\sigma} = \sqrt{t\sigma^2/\mu^3}$ e $\alpha = \log(1 - p_{fc})$, nós temos que:

$$\begin{aligned} E[K_t (1 - p_{fc})^{K_t}] &= E[(\hat{\sigma}X + \hat{\mu}) e^{\alpha(\hat{\sigma}X + \hat{\mu})}) \\ &= e^{\alpha\hat{\mu}} (\hat{\sigma} E[X e^{\alpha\hat{\sigma}X}] + \hat{\mu} E[e^{\alpha\hat{\sigma}X}]) \\ &= e^{\alpha\hat{\mu}} \left(\int_{-\hat{\mu}/\hat{\sigma}}^{\infty} (\hat{\sigma}x + \hat{\mu}) e^{\alpha\hat{\sigma}X} \varphi(x)\, dx \right) \end{aligned} \qquad (5.6)$$

Substituindo o resultado obtido na equação (5.6) na (5.4), nós temos a equação geral que permite calcular a performabilidade de um sistema:

$$E[M_t] = \frac{1 - p_{fb} - p_{fc}}{1 - p_{fc}} e^{\alpha\hat{\mu}} \left(\int_{-\hat{\mu}/\hat{\sigma}}^{\infty} (\hat{\sigma}x + \hat{\mu}) e^{\alpha\hat{\sigma}X} \varphi(x)\, dx \right) \qquad (5.7)$$

5.4 Modelagem de performabilidade do esquema MR

O MR é avaliado neste item usando um modelo de execução constituído de réplicas ativas sem privilégio (replicação ativa). A descrição do MR pode ser feita, como visto no capítulo anterior, em dois níveis: a nível de réplica e a nível de replicação. As figuras 5.5 e 5.6 sintetizam as estruturas básicas do MR que serão usadas no processo de modelagem.

Em cada iteração do esquema uma classe de algoritmos de aplicação na hierarquia de especialização é selecionada no teste de escalonamento *(TE)* a partir dos atributos dos dados de entrada. Por simplificação na modelagem, assumimos uma classe selecionada constituída por três algoritmos (figura 5.5). Uma vez definido os algoritmos que fazem parte da classe do dado de entrada, a funcionalidade de uma réplica é equivalente a do esquema de Bloco de

Recuperação (RB). Inicialmente, o algoritmo principal *(pref)* é executado, sendo o resultado obtido submetido ao teste de aceitação que pode: aceitar um resultado correto (saída 1), rejeitar um resultado correto ou incorreto (caminho interno) ou aceitar um resultado incorreto (saída 2). O algoritmo *back1* somente é executado se o resultado gerado pelo algoritmo *pref* é rejeitado. Neste caso, ocorre o *rollback*, isto é, o restabelecimento do estado inicial da réplica antes da execução de *back1*. Da mesma forma que para o algoritmo *pref*, o teste de aceitação pode aceitar ou rejeitar o resultado obtido por *back1*. Se o resultado gerado pelo algoritmo *back1* é rejeitado, ocorre um novo *rollback* e o algoritmo *back2* é executado. Diferentemente dos casos anteriores, se o resultado gerado pelo *back2* não é aceito, independentemente deste resultado ser correto ou não, o sistema ou envia um valor *default* ou não envia nenhum resultado. Os resultados obtidos em cada réplica são enviados ao votador (figura 5.6). No modelo é assumido o grau de replicação como sendo três (3).

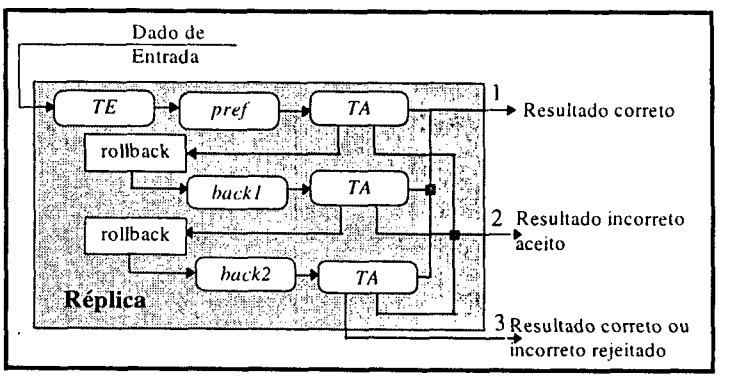


Figura 5.5: Estrutura interna de uma réplica

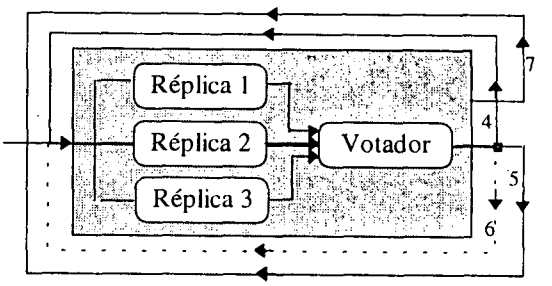


Figura 5.6: Estrutura de replicação do MR

5.4.1 Hipóteses de faltas para o esquema MR

Num ambiente distribuído, onde as réplicas são executadas em diferentes elementos de processamento (estações), além das faltas de projeto (software), existe a possibilidade de

faltas transitórias e permanentes de hardware. Neste trabalho de modelagem, em relação a faltas de hardware nos limitamos a faltas transitórias por serem de mais difícil trato. Os sintomas das faltas citadas no MR são distintos:

- as faltas de projeto se manifestam igualmente em todos os elementos de processamento (em todas as réplicas) em que se executa a replicação;
- as faltas de hardware (transitórias) podem se manifestar em subconjuntos dos elementos de processamento (subconjunto de réplicas) de uma replicação.

As faltas de hardware são tratadas na modelagem como faltas de hardware independentes. As faltas de software no esquema MR podem se manifestar nos algoritmos alternativos de aplicação, no teste de escalonamento, no teste de aceitação e no votador. Em relação ao teste de aceitação o que se pode dizer é que a cobertura do mesmo especifica o seu grau de eficácia na detecção de erros; na prática quanto maior o grau de cobertura mais complexo é o teste. O grau de cobertura assumido no modelo permite que o teste de aceitação deixe passar um resultado incorreto ou o teste rejeite um resultado correto. No que se refere ao teste de escalonamento o comportamento falha é representado pela seleção incorreta da classe (ou conjunto) de algoritmos alternativos.

O procedimento de votação (voto majoritário) no esquema MR pode apresentar dois tipos de falhas: não obter um resultado de consenso mesmo quando existe uma maioria ou ainda obter um resultado de consenso em situação que não existe maioria.

As faltas de software são tratadas como faltas independentes e relacionadas. As falhas de modo comum resultam de faltas que estão diretamente ligadas à aplicação, ou seja, são faltas produzidas nos algoritmos alternativos, no teste de aceitação e no teste de escalonamento. As combinações possíveis de faltas de software relacionadas são entre os algoritmos (alternativos) de aplicação, entre estes algoritmos e o teste de aceitação e entre o teste de escalonamento e o teste de aceitação. Não assumimos a ocorrência de faltas relacionadas entre os algoritmos alternativos e o teste de escalonamento porque na falha do teste de escalonamento um outro conjunto de algoritmos é selecionado.

Os tipos faltas considerados na análise do esquema MR composto por três réplicas iguais e três algoritmos internos de aplicação são apresentados na tabela 5.2. As probabilidades

associadas mostradas neste tabela serão parâmetros de entrada para o submodelo de Segurança de Funcionamento.

Tipos de faltas		Probabilidades
Falta(s) de software independente(s)	no *TE* (seleciona uma classe incorreta)	q_e
	no *TE* (seleciona classe incorreta com algoritmo da classe correta)	q_{ec}
	nos algoritmos alternativos *(pref, back 1* e *back 2)*	q_p, q_{b1}, q_{b2}
	no *TA* (resultado correto rejeitado ou resultado incorreto aceito)	q_a
	no votador (não reconhece maioria existente)	q_{d1}
	no votador (reconhece maioria inexistente)	q_{d2}
Faltas de software relacionadas	entre *TE* e *TA*	q_{ea}
	entre os algoritmos alternativos *(pref* e *back1, pref, back1 e back2* e entre *back1 e back2)*	q_{pb1}, q_{pb1b2}, q_{b1b2}, q_{pb2}
	entre *pref e TA*, entre *back1 e TA* e entre *back2 e TA*	q_{pa}, q_{b1a}, q_{b2a}
Falta de hardware independente	em um elemento de processamento	q_{ih}

Tabela 5.2: Tipos de faltas e probabilidades associadas

As faltas de software e de hardware assumidas neste item estão associadas a semânticas de falha por erro de valor e por erro de temporização. Ou seja, estas hipóteses se limitam a faltas de temporização e faltas por valor (ver item 2.3.2)[2]. As faltas de temporização se reduzem às faltas por atraso e de omissão. As faltas de *crash* não são consideradas porque foram desconsideradas as faltas de hardware permanentes.

No modelo de Performabilidade, os resultados obtidos de cada réplica são enviados ao votador (figura 5.6). Ao assumimos faltas de valor, a votação majoritária se faz necessária. Como as três réplicas são iguais nas suas estruturas internas, em não ocorrendo faltas transitórias de hardware, os resultados devem ser idênticos. Dependendo da votação a percepção da execução do MR se dará como:

---

[2] A ocorrência de erros de valor não necessariamente envolve o comportamento bizantino ou arbitrário. Nós no capítulo 2 a exemplo de [Little 91] e [Bookam 95] fazemos uma distinção entre faltas por valor e faltas arbitrárias. Sem a possibilidade de admitir a classe de faltas por valor fica difícil justificar nos esquemas de tolerância a faltas do capítulo 2 a existência de mecanismos de detecção de erro (como o teste de aceitação, por exemplo) fora do contexto de faltas bizantinas.

- **execução correta:** a maioria dos resultados fornecidos pelas réplicas são corretos (saída 4, figura 5.6);
- **execução incorreta com falha benigna por erro de valor:** não é obtida uma maioria a partir dos resultados fornecidos pelas réplicas (saída 5, figura 5.6);
- **execução incorreta com falha catastrófica:** as réplicas fornecem resultados incorretos e estes resultados formam uma maioria (saída 6 figura 5.6);
- **execução incorreta com falha benigna por erro de temporização:** a execução do esquema excede ao *deadline* estabelecido (saída 7 figura 5.6). Neste caso, a execução corrente é abortada e a próxima começa imediatamente.

O submodelo de Segurança de Funcionamento deve atribuir probabilidades (p_c, p_{fb}, p_{fc}) a estes tipos de execução que serão utilizadas mais tarde na análise de performabilidade do MR.

5.4.2 Submodelo de Segurança de Funcionamento do esquema MR

A construção do submodelo de Segurança de Funcionamento do MR é feita com base na estrutura do modelo (figura 5.5 e 5.6) e nas hipóteses de faltas consideradas no item anterior. Além disto, nós assumimos que todas as réplicas corretas executam os mesmos algoritmos de uma classe numa mesma seqüência e que as faltas de hardware somente se manifestam após a execução da réplica, isto é, antes da votação. Esta última hipótese reduz o número de estados e transições do submodelo de Segurança de Funcionamento, tornando assim a representação do submodelo mais concisa.

Seguindo a mesma sistemática usada na descrição do esquema MR a construção do submodelo de Segurança de Funcionamento é feita em duas etapas:

Etapa 1: O submodelo de Segurança de Funcionamento a nível réplica é construído, considerando apenas faltas de software;

Etapa 2: O submodelo de Segurança de Funcionamento a nível replicação é construído a partir dos estados terminais do submodelo concebido na etapa anterior. A ocorrência de falhas de hardware também é considerada nesta etapa

O submodelo de Segurança de Funcionamento do esquema MR consiste na composição dos submodelos construídos nas etapas 1 e 2.

5.4.2.1 Submodelo de Segurança de Funcionamento a nível réplica

O submodelo de Segurança de Funcionamento de uma réplica considerando apenas a ocorrência de falhas de software é mostrado na figura 5.7. Neste submodelo as transições estão relacionadas com as probabilidades de ocorrências de faltas independentes e relacionadas de software que fazem parte dos tipos de faltas apresentados no item 5.4.1. Os estados do submodelo da figura 5.7 são descritos na tabela 5.3.

As transições p_p, p_{b1} e p_{b2} correspondem, respectivamente, as probabilidades dos algoritmos *pref*, *back1* e *back2* serem executados corretamente e são calculadas diretamente a partir dos valores de outras probabilidades no próprio submodelo de Segurança de Funcionamento, considerando que a soma das probabilidades de todas as transições de saída de cada estado é igual a 1.

Considerando o estado P na figura 5.7, a ocorrência de faltas relacionadas e faltas independentes permitem diferentes cenários de transições:

- faltas relacionadas entre o preferencial e o teste de aceitação (transição q_{pa}) faz com que um resultado incorreto seja aceito pelo teste de aceitação. Esta transição leva o sistema do estado P para V_3;
- faltas relacionadas entre o teste de escalonamento e o teste de aceitação (q_{ea}) faz com que um resultado incorreto obtido por um algoritmo que não pertença a hierarquia correta (probabilidade $(1-p_x)$) seja aceito pelo teste de aceitação, levando o sistema do estado P a V_3;
- faltas independentes de *pref* (transição q_p) ou faltas relacionadas entre as alternativas *pref* e *back1* (transição q_{pb1}) ou entre *pref* e *back2* (q_{b1b2}) ou entre *pref, back1 e back2* (q_{pb1b2}) faz com que o sistema passe do estado P a P^I (resultado incorreto obtido pelo preferencial), sendo que se o teste de aceitação rejeita o resultado (transição $1-q_a$) o algoritmo *back1* é executado (estado B_1), senão o resultado incorreto é aceito pelo teste de aceitação (estado V_3).

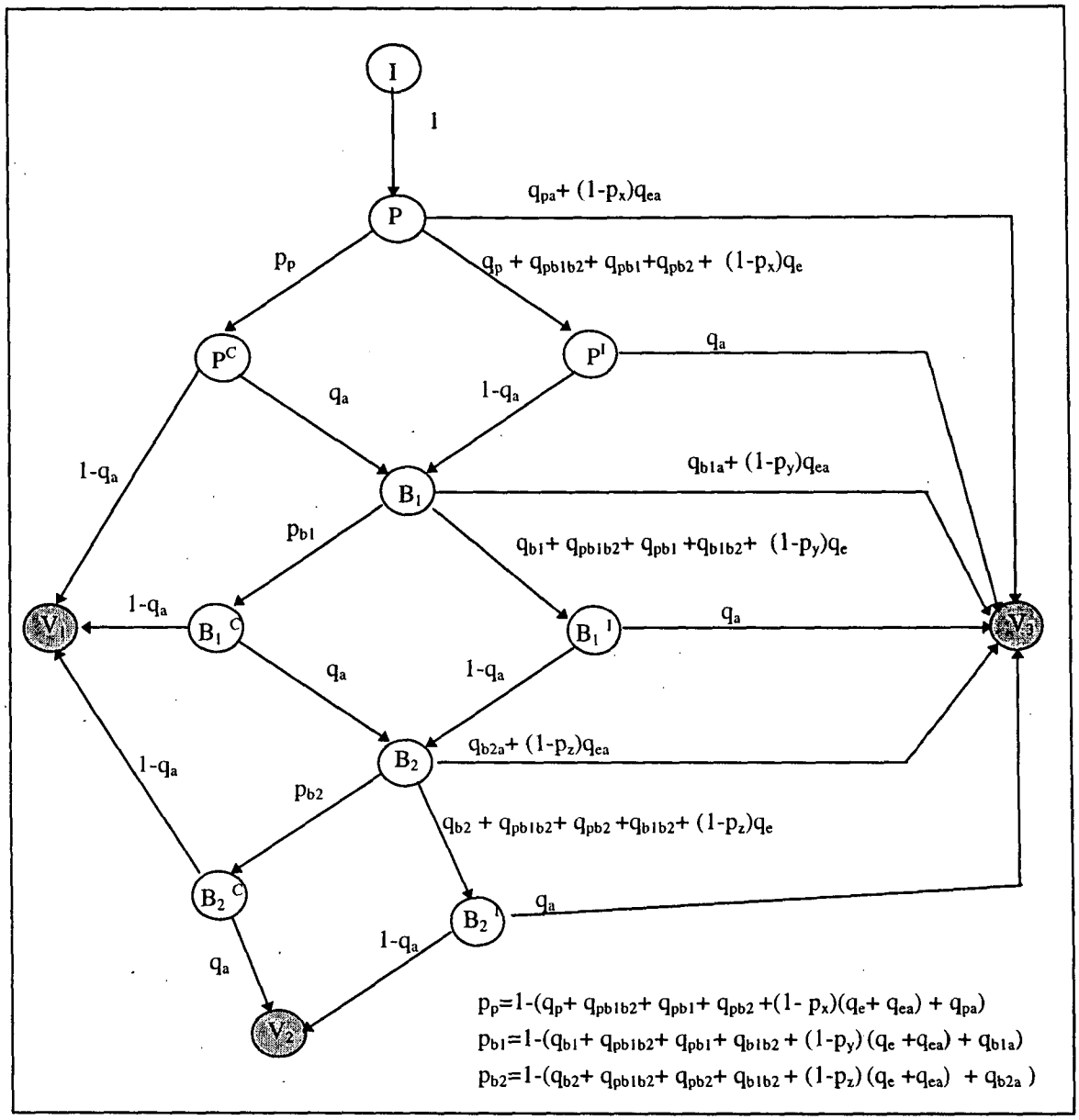


Figura 5.7: Submodelo de Segurança de Funcionamento de uma réplica

Na figura 5.7 a execução correta de um algoritmo preferencial que faz parte da hierarquia leva o sistema do estado P para P^C. A probabilidade do algoritmo preferencial gerar um resultado correto p_p está associada a esta transição. A execução correta do teste de aceitação (transição $1-q_a$) leva o sistema do estado P^C para o estado V_1 (resultado correto aceito pelo TA). A transição q_a corresponde a rejeição de um resultado correto pelo teste de aceitação, levando ao estado B_1, onde o sistema executa o algoritmo *back1*.

O mesmo raciocínio usado na descrição da execução do algoritmo preferencial permite entender o submodelo de Segurança de Funcionamento quando da execução dos algoritmos

back1 e *back2*. Quando o algoritmo *back2* fornece um resultado (correto ou incorreto) e o teste de aceitação rejeita este resultado a execução da réplica termina no estado V_2 (resultado correto ou incorreto rejeitado pelo teste de aceitação).

Estados	Descrição
I	Estado inicial.
P	Execução do algoritmo *pref.*
P^C	Execução correta do algoritmo *pref.*
P^I	Execução incorreta do algoritmo *pref.*
B_i	Execução do *i-esimo* algoritmo de *backup (back1 e back2).*
B_i^C	Execução correta do *i-esimo* algoritmo de *backup (back1 e back2).*
B_i^I	Execução incorreta do *i-esimo* algoritmo *backup (back1 e back2).*
V_1	Resultado correto aceito pelo TA.
V_2	Resultado correto ou incorreto rejeitado pelo TA.
V_3	Resultado incorreto aceito pelo TA.

Tabela 5.3: Estados do submodelo de Segurança de Funcionamento do MR

O submodelo de Segurança de Funcionamento a nível de replicação é construído a partir dos estados terminais (V_1, V_2 e V_3) do submodelo da figura 5.7. Desta forma, o submodelo de Segurança de Funcionamento a nível de réplica é construído com o objetivo de encontrar as expressões que permitem calcular as probabilidades de um resultado correto ser aceito (estado V_1), de um resultado correto ou incorreto ser rejeitado (estado V_2) e de um resultado incorreto ser aceito (estado V_3). Estas probabilidades são calculadas a partir da soma de todas as probabilidades associadas aos caminhos que ligam o estado inicial (I) aos respectivos estados V_1, V_2 e V_3. A probabilidade de um estado é calculada através da multiplicação das probabilidades associadas as transições entre estes dois estados. Desta forma, nós temos:

- Probabilidade da réplica aceitar um resultado correto (p_{v1}):

$$p_{v1} = (1-q_a)(p_p + p_{B1}p_{b1} + p_{B2}p_{b2}) \quad (5.8)$$

- Probabilidade da réplica rejeitar um resultado correto ou incorreto (p_{v2}) :

$$p_{v2} = p_{B2}[p_{b2}q_a + (q_{b2} + q_{pb1b2} + q_{pb2} + q_{b1b2} + (1-p_z)q_e)(1-q_a)] \quad (5.9)$$

- Probabilidade da réplica aceitar um resultado incorreto (p_{v3}):

$$p_{v3} = 1 - p_{v1} - p_{v2} \quad (5.10)$$

onde, as probabilidades p_{B1} e p_{B2} estão associadas a uma seleção correta das alternativas de aplicação e correspondem aos estados B_1 (execução do algoritmo *back1*) e B_2 (execução do algoritmo *back2*), respectivamente:

$$p_{B1} = p_p\, q_a + (q_p + q_{pb1b2} + q_{pb1} + q_{pb2} + (1-p_x)q_e)\,(1-q_a)$$

$$p_{B2} = p_{B1}\,(p_{b1}\, q_a + (q_{b1} + q_{pb1b2} + q_{pb1} + q_{b1b2} + (1-p_y)\, q_e)\,(1-q_a))$$

sendo que p_x, p_y e p_z correspondem, respectivamente, as probabilidades de seleção de um algoritmo *pref*, *back1* e *back2* que pertença a hierarquia de classe correta mesmo quando da ocorrência de falhas no teste de escalonamento.

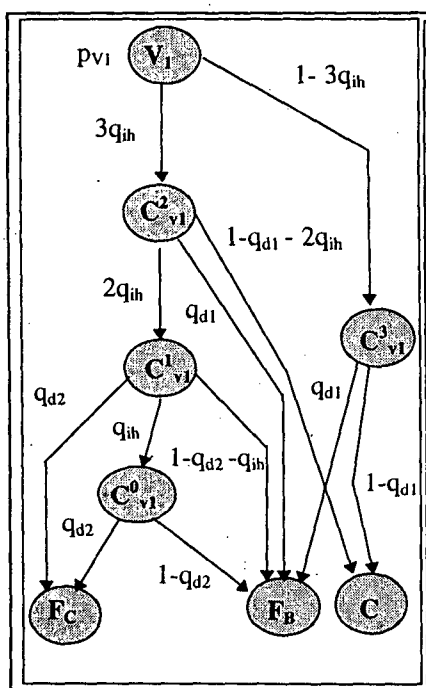


Figura 5.8(a)

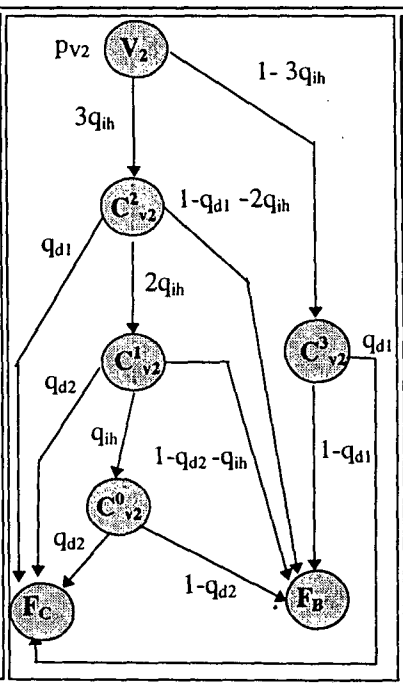


Figura 5.8(b)

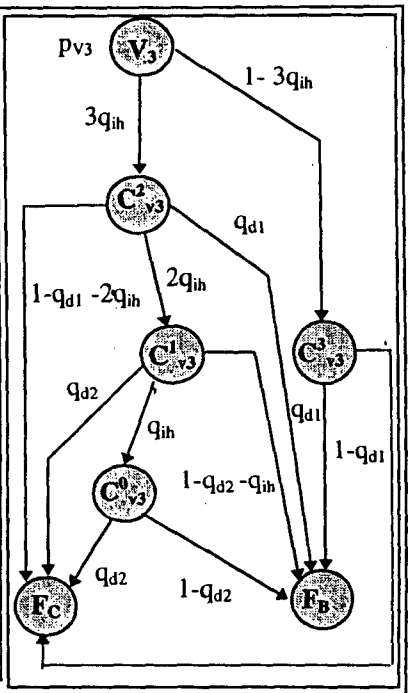


Figura 5.8(c)

Estados	Descrição
V_1	Resultado correto aceito pelo TA
V_2	Resultado correto ou incorreto rejeitado pelo TA
V_3	Resultado incorreto aceito pelo TA
C^x_y	Estado com origem y (V_1,V_2,V_3), sendo x o número de réplica que não tem seu resultado alterado por falhas de hardware
C	Iteração correta
F_B	Iteração incorreta com falha benigna
F_C	Iteração incorreta com falha catastrófica

Figura 5.8: Grafos de estados do MR considerando apenas a votação e faltas de hardware

As expressões acima são resultados intermediários que posteriormente serão utilizados no cálculo das probabilidades de execução incorreta com falha catastrófica (p_{fc}), de execução incorreta com falha benigna por erro de valor (p_{fbv}); referentes as execuções do MR.

5.4.2.2 Submodelo de Segurança de Funcionamento a nível replicação

A caracterização de uma iteração como correta ou incorreta (benigna por erro de valor e catastrófica) é feita com base nos resultados fornecidos pelas réplicas. Estes resultados estão relacionados aos estados V_1 (resultado correto), V_2 (resultado correto ou incorreto rejeitado pelo teste de aceitação) e V_3 (resultado incorreto aceito pelo teste de aceitação) da figura 5.7.

A possibilidade de ocorrência de faltas de hardware nas estações onde as réplicas são executadas, faz com que, mesmo quando os resultados submetidos a votação são corretos o sistema (MR) possa apresentar uma falha catastrófica ou benigna por erro de valor na presença de duas ou mais falhas de hardware. A figura 5.8 ilustra os três subgrafos do esquema MR a nível de replicação:

- A figura 5.8(a) corresponde ao caso em que os resultados obtidos pelas réplicas são corretos e foram validados pelo teste de aceitação. Se não ocorrer nenhuma falha de software no votador ou de hardware a iteração é dita correta. Na ocorrência de duas falhas de hardware a iteração pode ser incorreta com falha benigna por erro de valor (estado F_B). A iteração pode ser incorreta com falha catastrófica (estado F_C) quando o votador apresentando falha chega a um consenso a partir de resultados incorretos;

- A figura 5.8(b) ilustra o caso onde os resultados fornecidos pelas réplicas foram rejeitados pelo teste de aceitação (estado V_2). A partir do estado V_2 é impossível que a iteração corrente seja correta. Se não ocorrem falhas de software no votador ou de hardware na maioria das réplicas, a iteração deve alcançar o estado F_B onde a mesma é classificada como incorreta com falha benigna por erro de valor. Quando da ocorrência de falhas de hardware ou de software no votador, o estado F_C pode ser alcançado e a iteração é classificada como incorreta com falha catastrófica;

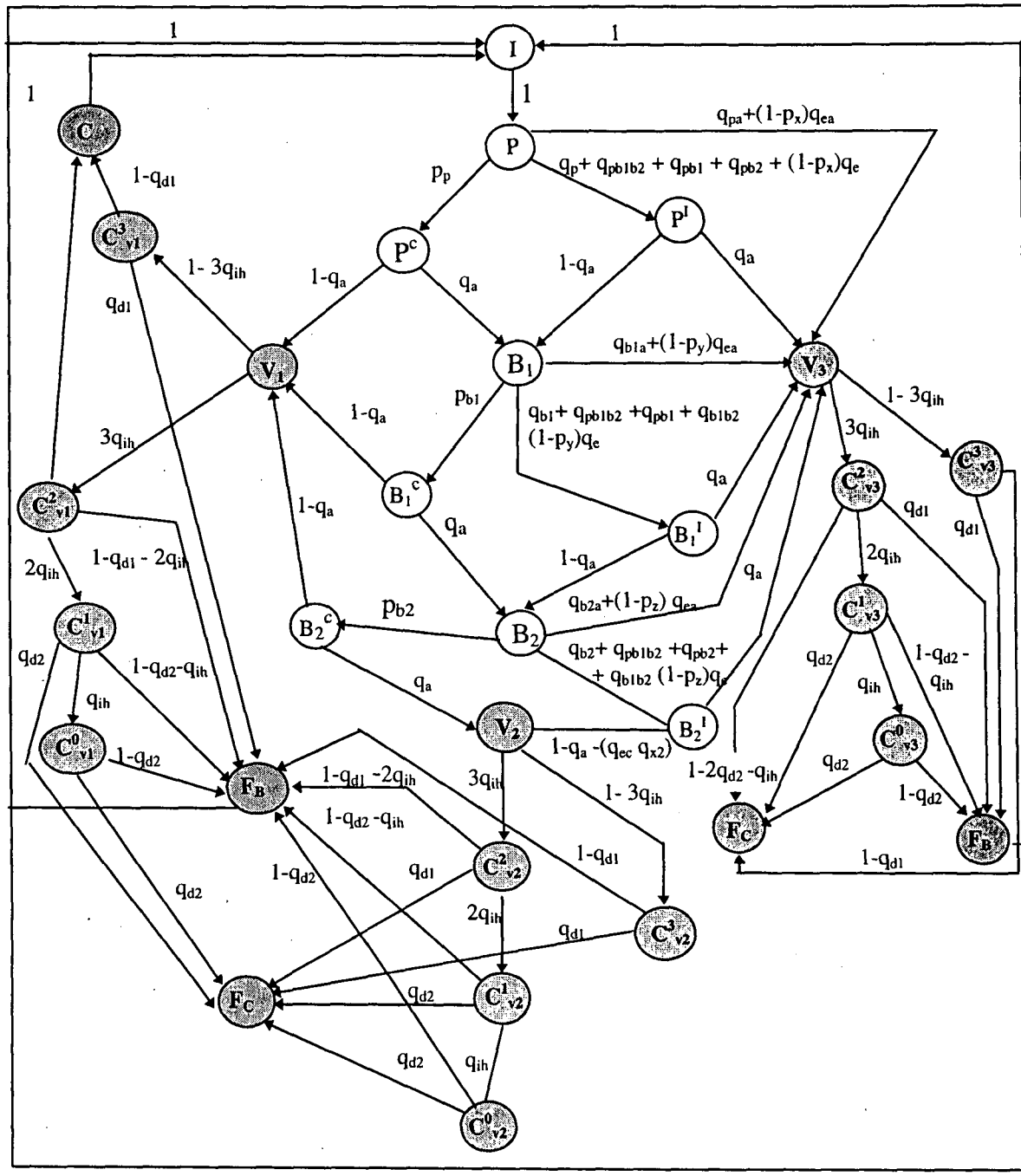


Figura 5.9: Submodelo de Segurança de Funcionamento do MR

- A figura 5.8(c) representa o caso em que os resultados fornecidos pelas réplicas foram validados pelo teste de aceitação. Se nenhuma falha de hardware ou de software no votador ocorre, o grafo evolui para o estado F_C onde a iteração é classificada como incorreta com falha catastrófica. Se, em caso contrário, ocorrem falhas de hardware e no votador, o estado F_B pode ser alcançado com falha benigna por erro de valor.

A figura 5.9 é uma combinação das figuras 5.7 e 5.8, e corresponde ao submodelo completo de Segurança de Funcionamento para o esquema do esquema MR.

As expressões que permitem calcular as probabilidades de interações incorretas com falha benigna por erro de valor (p_{fbv}) e incorretas com falha catastrófica (p_{fc}) são calculadas a partir do diagrama da figura 5.9:

- Probabilidade de falhas benignas devido a erros de valores (p_{fbv}):

$$\begin{aligned} p_{bv} = {} & p_{v1}[(1-3q_{ih})q_{d1} + 3q_{ih}((1-q_{d1} - 2q_{ih}) + 2q_{ih}((1-q_{d2}-q_{ih}) + q_{ih}(1-q_2)))] + \\ & p_{v2}[(1-3q_{ih})(1-q_{d1}) + 3_{ih}((1- q_{d1} - 2q_{ih}) + 2q_{ih}((1- q_{d1} - q_{ih}) + q_{ih}(1-qd_2)))] \\ & p_{v3}[(1-3q_{ih})q_{d1} + 3_{qih}(q_{d1} + 2q_{ih}(q_{ih}(1-q_{d2}) + (1-q_{d2}-q_{ih})))] \end{aligned} \quad (5.11)$$

- Probabilidade de falhas catastróficas (p_{fc}):

$$\begin{aligned} p_{fc} = {} & p_{v1}[3q_{ih}\,2\,q_{ih}\,q_{d2}\,(1+q_{ih}q_{d2})] + \\ & p_{v2}[(1-3q_{ih})q_{d1} + 3q_{ih}(q_{d1}+2q_{ih}q_{d2}(1-q_{ih}))] + \\ & p_{v3}[(1-3q_{ih})(1-q_{d1}) + 3q_{ih}((1-q_{d1}-2q_{ih})+ 2q_{ih}q_{d2}(1-q_{ih}))] \end{aligned} \quad (5.12)$$

5.4.3 Submodelo de Desempenho do esquema MR

Como citado anteriormente, o submodelo de Desempenho é um processo de *renewal*, representado por:

$$K = \{K_t \mid t \geq 0\}$$

No cálculo do valor de K_t nós consideramos que o tempo de execução dos componentes do esquema (teste de escalonamento, algoritmos de aplicação, teste de aceitação, etc.) são independentes e apresentam uma distribuição exponencial.

A figura 5.10 apresenta a estrutura interna de um serviço implementado a partir do esquema MR contendo três réplicas com três alternativas de algoritmos de aplicação. Os parâmetros entre parênteses correspondem ao valor médio do tempo de execução de cada componente. Os parâmetros p_1, p_2 e p_3 correspondem às probabilidades do resultado ser fornecido pelos algoritmos *pref*, *back1* e *back2*, respectivamente. Estes valores de probabilidade são calculados a partir do submodelo de Segurança de Funcionamento do esquema MR (figura 5.9).

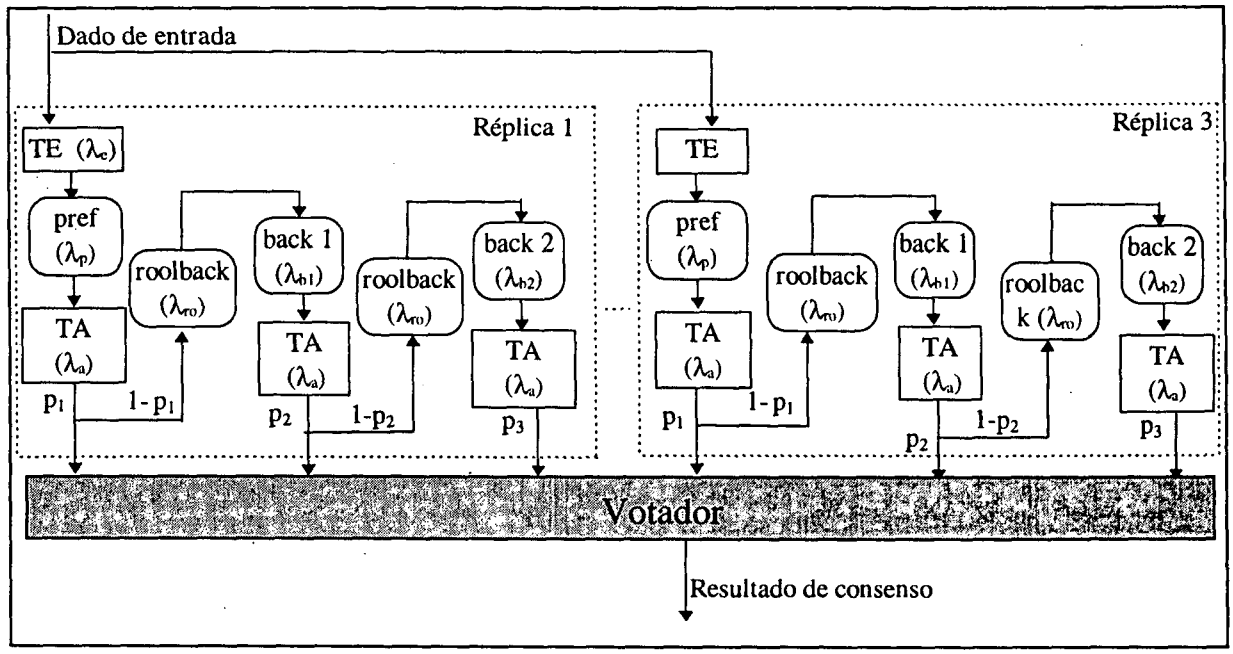


Figura 5.10: Submodelo de Desempenho do MR

A função densidade de probabilidade de uma iteração do esquema MR, representada por $f_{sist}^{MR}(y)$, é calculada através das transformadas inversas de Laplace[3] do produto da transformada de Laplace do tempo de execução das três réplicas em paralelo ($F_{RE}(s)$), com a transformada de Laplace da *fdp* do tempo de execução do votador ($F_{VO}(s)$), isto é:

$$f_{sist}^{MR}(y) = L^{-1}\,[F_{RE}(s) * F_{VO}(s)] \qquad (5.13)$$

Do ponto de vista das execuções das réplicas, nós assumimos que as réplicas são executadas paralelamente e fornecem os resultados ao votador no mesmo instante. Desta forma, o tempo de execução das réplicas em paralelo, representado pela variável aleatória y_{RE}, é igual a:

$$y_{RE} = y_{R1} = y_{R2} = y_{R3}$$

onde, y_{R1}, y_{R2} e y_{R3} são as variáveis aleatórias que representam, respectivamente, o tempo de execução das réplicas 1, 2 e 3.

A transformada de Laplace do tempo de execução das réplicas em paralelo, representada por $F_{RE}(s)$, é calculada a partir da transformadas de Laplace das *fdps* dos tempos de execução do

---

[3] A função densidade de probabilidade $f_{sist}^{MR}(y)$ poderia ser obtida no domínio do tempo através do cálculo das integrais e convoluções das pdfs dos componentes do esquema. Como nós assumimos que as pdf são exponenciais o cálculo da $f_{sist}^{MR}(y)$ através das transformadas de Laplace é muito mais simples e será sempre utilizada neste trabalho quando for possível.

teste de escalonamento $F_{te}(s)$), dos algoritmos *pref* ($F_p(s)$), *back1* ($F_{b1}(s)$), *back2* ($F_{b2}(s)$), do teste de aceitação ($F_{ta}(s)$) e do *rollback* ($F_{ro}(s)$), pela seguinte expressão:

$$F_{RE}(s) = p_1 [F_{te}(s)F_p(s)F_{ta}(s)] + p_2 [F_{te}(s)F_p(s)F_{ro}(s)F_{b1}(s)\, F_a^2(s)] + p_3 [F_{te}(s)F_p(s)F_{ro}^2(s)F_{b1}(s)Fb2(s)\, F_a^3(s)] \quad (5.14)$$

onde, p_1 é a probabilidade do algoritmo *pref* fornecer o resultado, p_2 é a probabilidade do algoritmo *back1* fornecer o resultado e p_3 é a probabilidade do resultado ser fornecido por *back2*. Os valores e p_1, p_2, p_3 são obtidos diretamente do submodelo de Segurança de Funcionamento do esquema MR (figura 5.9):

$$p_1 = [p_p(1-q_a) + q_a (q_p + q_{pb1b2} + q_{pb1} + q_{pb2} + (1-p_x)(q_e + q_{ea})) + q_{pa}]$$

$$p_2 = p_{B1} [p_{b1} (1-q_a) + q_a (q_{b1} + q_{pb1b2} + q_{pb1} + q_{b1b2}) + q_{b1a}]$$

$$p_3 = 1 - (p_1 + p_2)$$

5.4.4 Cálculo da Performabilidade (E[M_t]) do esquema MR

A performabilidade do esquema MR é calculada a partir da substituição, na equação 5.7, dos valores de probabilidade de falhas catastróficas por erros de valor (p_{fc}) e falhas benignas por erros de valor (p_{fbv}) obtidos no submodelo de Segurança de Funcionamento (equações 5.11 e 5.12) e do tempo médio e variância de execução (iteração) do MR obtidos no submodelo de Desempenho através da equação 5.13.

5.5 Estudo comparativo baseado na performabilidade de esquemas de tolerância a faltas

Neste item é apresentado um estudo comparativo dos principais esquemas de tolerância a faltas. Este estudo utiliza como medida de referência a performabilidade. Entretanto, além da performabilidade de cada esquema, também são analisados os resultados obtidos nos submodelos de Segurança de Funcionamento e de Desempenho, permitindo assim verificar as vantagens e deficiências de cada esquema. Fazem parte deste estudo, além do MR, os esquemas: RB, PNV, PNV-TB, PNV-TA, SCOP e CRB. Os Modelos de Performabilidade dos esquemas analisados, com exceção do MR que foi apresentado no item anterior, são

descritos em detalhes no Apêndice B. Na seqüência são discutidos os valores atribuídos aos parâmetros nos processos de modelagem e os casos de estudo considerados nos experimentos.

5.5.1 Escolha dos parâmetros para Modelos de Performabilidade

O número limitado de trabalhos que abordam a confiabilidade de componentes internos de esquemas de tolerância a faltas, como por exemplo a probabilidade de faltas no teste de aceitação e/ou do votador (ou ajustador), é o principal obstáculo encontrado na avaliação destes esquemas. Além disto, os poucos trabalhos existentes estão relacionados a aplicações específicas. Normalmente, esta limitação é contornada variando estes parâmetros no modelo e analisando a posteriori o comportamento do sistema, de modo a estabelecer faixas para as confiabilidades dos componentes.

O nosso procedimento neste texto é utilizar estudos que levantam estas faixas de probabilidades de faltas de componentes. Neste sentido, em relação a um esquema assumimos as probabilidades de faltas e os tempos médios de execução de seus componentes de acordo com as seguintes hipóteses:

Submodelo de Segurança de Funcionamento:

⇒ **Faltas independentes**

- **Alternativas de aplicação:** Segundo a classificação de [Buther 93] probabilidades de faltas acima de 10^{-3} são consideradas de baixa confiabilidade, faixas de probabilidades entre 10^{-3} a 10^{-7} representam confiabilidades moderadas e probabilidade de faltas abaixo de 10^{-7}, por sua vez, representam confiabilidades altas. Neste trabalho assumimos que as probabilidades de faltas independentes nas alternativas de aplicação estão situadas na faixa de confiabilidade moderada.

- **Teste de aceitação:** Neste estudo assume-se que a probabilidade de faltas independentes no teste de aceitação é inferior a das alternativas de aplicação. Além disto, devido a complexidade na elaboração do teste de aceitação, a probabilidade de faltas no teste de aceitação é maior do que nos mecanismos de votação e/ou de ajuste. Esta consideração está de acordo com a análise apresentada em [Laprie 90].

- **Votador ou Ajustador:** Da mesma forma que [Tai 93a, Chiaradonna 94], nós assumimos que para qualquer tipo de votação (exata ou inexata) a probabilidade de faltas independentes no votador ou ajustador é a mesma. Desta forma, para esquemas que fazem uso da votação inexata, os resultados obtidos correspondem a melhor situação, uma vez que, segundo [Laprie 90], a confiabilidade das funções de ajuste como votação inexata é menor ou igual; na melhor situação, a funções com votação exata.

⇒ **Faltas relacionadas**

- **Entre uma das alternativas de aplicação e o teste de aceitação:** As probabilidades de faltas relacionadas envolvendo uma das alternativas de aplicação e o teste de aceitação são pequenas. Neste trabalho, assumimos valores para estas probabilidades inferiores as probabilidades de faltas atribuídas aos algoritmos alternativos de aplicação. Como base nos valores apresentados em [Tai 93a, Chiaradonna 94, Laprie 90], estas probabilidades são assumidas com valores na ordem de 10^{-9}.

- **Entre mais de dois componentes:** As probabilidades de faltas relacionadas envolvendo mais de dois componentes de um esquema de tolerância a faltas são reduzidas, estando na ordem de 10^{-10}.

Submodelo de Desempenho:

- **Tempo médio de execução dos algoritmos alternativos de aplicação:** o desenvolvimento dos algoritmos alternativos de aplicação através da diversidade de projeto faz com que os tempos médios de execução destes algoritmos variem moderadamente [Tai 93a, Chiaradonna 94];

- **Tempo médio de execução do teste de aceitação:** devido a complexidade da elaboração do teste de aceitação e a sua dependência em relação à aplicação, assume-se que o tempo médio de execução do teste de aceitação é da mesma ordem que o tempo médio de uma alternativa de aplicação;

- **Tempo médio de execução do votador ou ajustador:** Devido a simplicidade das funções de votação (normalmente, baseadas em comparações) e das funções de ajuste (cálculo do valor médio, mediano, etc.), são atribuídos aos mesmos valores de tempo médio de execução menores do que do teste de aceitação. Nesta análise, os custos da comunicação que estão relacionados com os envios e recepções de mensagens, não são considerados por estarem diretamente associados ao modelo de execução;

As hipóteses associadas com a escolha dos valores de probabilidade de faltas e de tempo médio de execução do teste de escalonamento por serem específicas ao esquema MR são descritas no item 5.5.2 (caso 1).

5.5.2 Estudos de casos

Em [Tai 93a] foi analisada a influência de faltas relacionadas nos esquemas PNV e RB. Posteriormente, análise semelhante incluindo o esquema SCOP e os esquemas PNV-TB e PNV-TA, foram realizadas, respectivamente, em [Chiaradonna 94] e [Tai 93b]. O estudo comparativo desenvolvido no âmbito deste trabalho é dividido nos seguintes casos:

- **Caso 1:** cálculo da performabilidade do esquema MR e análise da influência do mecanismo de seleção dinâmica (teste de escalonamento) de classes de alternativas na performabilidade do esquema;

- **Caso 2:** comparação das performabilidades dos esquemas de tolerância a faltas, considerando a presença de faltas relacionadas nos algoritmos alternativos da aplicação e nos mecanismos de detecção e mascaramento de erros;

- **Caso 3:** comparação das performabilidades dos esquemas de tolerância a faltas considerando faltas independentes nos algoritmos alternativos de aplicação.

Os cálculos de performabilidade de cada esquema, nos casos a seguir, tratam de uma missão com duração de 10 horas, sendo que o *deadline* máximo de uma iteração é de 30 *ms*. Nos experimentos citados, ao contrário de [Tai 93a, Tai 93b, Chiaradonna 94], são consideradas as alternativas do esquema RB como ordenadas em ordem decrescente de tempos médios de

execução. Essa ordenação ocorre normalmente na prática devido ao fato que o algoritmo principal do esquema RB é o mais completo e portanto mais complexo entre as alternativas em um bloco de recuperação. As tabelas 5.4 e 5.5 apresentam, respectivamente, os valores de parâmetros dos submodelos de Segurança de Funcionamento e de Desempenho considerados nas hipóteses do item 5.5.1.

Probabilidade de faltas		**Caso 1**	**Caso 2**	**Caso 3**
faltas independentes	nas alternativas de aplicação[4]	Variável	2.91×10^{-3}	Variável de 10^{-3} a 9×10^{-3}
	no teste de aceitação	$q_a = 10^{-6}$		
	no ajustador	$q_{d1} = q_{d2} = 10^{-9}$		
faltas relacionadas	- entre duas alternativas de aplicação; - entre uma alternativa **e** o mecanismo de detecção de erro; - entre os testes de escalonamento e de aceitação	Variável	Variável de 0 a 180 $\times 10^{-9}$	80×10^{-9}
	em três alternativas de aplicação	10^{-10}		

Tabela 5.4: Valores utilizados no submodelo de Segurança de Funcionamento

	Taxa de execução[5] *(1/ms)*
Esquema RB	- principal: 1/8 - secundário: 1/5 - teste de aceitação: 1/5 - *rollback*: 0.28
Esquemas PNV, PNV-TB, e SCOP	- versão 1: 1/5 - versão 2: 1/6 - versão 3: 1/8 - ajustador: 2.0
Esquemas CRB e PNV-TA	- versão 1: 1/5 - versão 2: 1/6 - versão 3: 1/8 - ajustador: 2.0 - teste de aceitação: 1/5
***Deadline* máximo de uma iteração**	t = 30 *ms*

Tabela 5.5: Valores assumidos de taxas de execução usadas nos submodelos de Desempenho para os componentes de esquemas de tolerância a faltas

A tabela 5.4 mostra que nos três casos citados são assumidas diferentes situações de faltas independentes e relacionadas. No caso 2, embora a ênfase seja faltas relacionadas, é assumida

---

[4] Os valores de probabilidade usados aqui foram obtidos e usados em [Laprie 90].

[5] O valor médio do tempo de execução de uma alternativa corresponde ao inverso da taxa de execução de uma iteração.

uma probabilidade fixa de faltas independentes. O mesmo ocorre em relação ao caso 3 onde, ao contrário do caso anterior, a ênfase são as faltas independentes e as fixas são as relacionadas.

Caso 1: Cálculo da performabilidade do esquema MR e análise do mecanismo de seleção dinâmica de classes no esquema MR

O mecanismo de teste de escalonamento introduzido no esquema MR tem o objetivo de flexibilizar a execução das iterações, selecionando alternativas que são mais apropriadas ao processamento de determinados dados de entrada. Este caso de estudo tem o objetivo de calcular a performabilidade do esquema MR e analisar o mecanismo de seleção dinâmica de classes na hierarquia de algoritmos alternativos do esquema.

Devido a simplicidade do teste de escalonamento, nós consideramos que o tempo médio de execução do teste de escalonamento no esquema MR eqüivale a 10% do tempo médio de execução de uma alternativa.

Por hipótese, a probabilidade de faltas independentes no teste de escalonamento é da mesma ordem que a do teste de aceitação. Uma falha no teste de escalonamento faz com que a classe de alternativas selecionadas seja incorreta. Entretanto, em algumas situações é possível que esta classe incorreta contenha algumas alternativas da classe correta. Neste caso, o algoritmo alternativo pode fornecer um resultado correto, mesmo quando da seleção de uma classe incorreta.

Se considerarmos uma hierarquia montada a partir de n algoritmos e se não houver nenhuma restrição quanto aos arranjos possíveis com estes algoritmos, o número máximo de classes que se formariam a partir de n, representado por N, é dado por:

$$N = \sum_{r=1}^{n} A\binom{n}{r}$$

É claro que se olharmos situações concretas, grande parte dos possíveis arranjos que se formam de um conjunto de algoritmos são inviáveis como classes de uma hierarquia.

Nas análises que realizamos, no sentido de simplificar os nossos modelos e também de manter o MR em condições similares às dos outros esquemas, nós assumimos que as classes contém 3

algoritmos e são formadas a partir de 7 algoritmos, reduzindo o número de classes a $A\binom{7}{3}$. Os tempos médios de execução destes algoritmos foram considerados como: 5 *ms* (alg_1), 5,5 *ms* (alg_2), 6 *ms* (alg_3), 6.5 *ms* (alg_4), 7 *ms* (alg_5), 7.5 *ms* (alg_6), 8 *ms* (alg_7).

Se assumimos o número total de classes, ou seja, todos os arranjos para $n = 7$ e $r = 3$, teríamos uma hierarquia com 840 classes o que tornaria inviável ainda o processo de modelagem do MR. Diante disto, assumimos a hierarquia em nossos experimentos como sendo a dada pela tabela 5.6. Nesta tabela, cada algoritmo alternativo é referenciado apenas pelo seu tempo médio de execução.

A tabela 5.6 poderia ter sido montada com arranjos que refletissem o algoritmo mais rápido na posição de preferencial. Outras escolhas poderiam ser feitas no sentido de determinar o melhor desempenho do conjunto. No sentido de uma melhor isenção nos nossos trabalhos de medidas, assumimos que todos os algoritmos, mesmos os mais lentos, ocorrem na posição de preferencial com a mesma distribuição. O mesmo se sucede nas posições de *back1* e *back2*, ou seja, todos os algoritmos se repetem de forma idêntica nestas posições[6].

Hierarquias	*pref*	*back1*	*back2*
K	5,0 *ms*	5,5 *ms*	6,0 *ms*
L	5.5 *ms*	6,0 *ms*	6,5 *ms*
M	6,0 *ms*	6,5 *ms*	7,0 *ms*
N	6,5 *ms*	7,0 *ms*	7,5 *ms*
O	7,0 *ms*	7,5 *ms*	8,0 *ms*
P	7,5 *ms*	8,0 *ms*	5,0 *ms*
Q	8,0 *ms*	5,0 *ms*	5,5 *ms*

Tabela 5.6: Hierarquias de classes consideradas neste experimento

As probabilidades p_x, p_y e p_z (introduzidas no item 5.4.2.1) correspondem, respectivamente, as probabilidades dos algoritmos *pref*, *back1* e *back2* da classe selecionada, quando da ocorrência de falha no teste de escalonamento, pertencerem a classe correta. No experimento realizado foi assumido $p_x = 3/7$, $p_y = 2/6$ e $p_z = 1/5$. Estes valores podem ser tirados facilmente da tabela 5.6.

---

[6] Estas simplificações não comprometem as análises dos resultados, pois realizamos experimentos com um número menor de algoritmos alternativos (3), considerando todos os arranjos possíveis $N = \sum_{r=1}^{3} A\binom{3}{r}$ e observamos tanto no caso das simplificações citadas acima como nas hierarquias obtidas a partir de poucos algoritmos que os valores obtidos de performabilidade foram bem similares.

Com base nos parâmetros acima e valores descritos acima, a performabilidade do esquema MR foi calculada a partir das equações apresentadas no item 5.4. As transformadas de Laplace, as transformadas inversas de Laplace e as derivadas e integrais das funções densidade de probabilidade foram calculadas através do software ***Mathematica***[7]. Os valores de performabilidade obtidos para a hierarquia de classes da tabela 5.6 são apresentados nas tabelas 5.7 e 5.8:

- os valores de performabilidade apresentados na tabela 5.7 foram calculados a partir da variação de probabilidades de faltas relacionadas. As faltas relacionadas consideradas neste experimento são entre o teste de aceitação e as alternativas de aplicação *pref* (q_{pa}), *back1* (q_{b1a}), e *back 2* (q_{b2a}) e entre o teste de escalonamento e o teste de aceitação (q_{ea});
- os valores de performabilidade apresentados na tabela 5.8 foram calculados a partir da variação das probabilidades independentes nos algoritmos de aplicação *pref* (q_p), *back1* (q_{b1}) e *back2* (q_{b2}).

Considerando que a distribuição dos dados de entrada no MR é uniforme, isto é, que o teste de escalonamento seleciona as classes com igual probabilidade, a performabilidade do esquema MR é dada pelos valores médios das performabilidades das classes da hierarquia. Estes valores médios serão utilizados nos casos subsequentes.

Hierarquias de classes	Performabilidade ($E[M_t]$) x 10^6									
	Probabilidades de faltas relacionadas (x 10^{-9})									
	0	20	40	60	80	100	120	140	160	180
K	3,3843	3,0351	2,7220	2,4411	2,1893	1,9634	1,7608	1,5792	1,4163	1,2701
L	3,2235	2,9043	2,6167	2,3576	2,1241	1,9138	1,7243	1,5535	1,3997	1,2611
M	3,0752	2,7822	2,5172	2,2774	2,0604	1,8641	1,6865	1,5258	1,3804	1,2489
N	2,9393	2,6692	2,4239	2,2012	1,9989	1,8152	1,6484	1,4970	1,3594	1,2345
O	2,8135	2,5636	2,3359	2,1284	1,9393	1,7671	1,6101	1,4671	1,3368	1,2181
P	2,6980	2,4658	2,2537	2,0597	1,8825	1,7205	1,5725	1,4372	1,3135	1,2005
Q	2,5906	2,3742	2,1759	1,9942	1,8277	1,6750	1,5351	1,4069	1,2894	1,1817
Valor Médio	**2,9606**	**2,6849**	**2,4350**	**2,2085**	**2,0032**	**1,8170**	**1,6482**	**1,4952**	**1,3565**	**1,2307**

Tabela 5.7: Performabilidade do esquema MR a partir da variação de faltas relacionadas

[7] *Mathematica* é um trademark registrado da Wolfram Research Inc.

Performabilidade ($E[M_t]$) x 10^6										
Hierarquias de classes	Probabilidades de faltas independentes (x 10^{-3})									
	0	1	2	3	4	5	6	7	8	9
K	2,1986	2,1954	2,1922	2,1890	2,1858	2,1825	2,1791	2,1757	2,1724	2,1689
L	2,1330	2,1300	2,1269	2,1238	2,1207	2,1176	2,1144	2,1113	2,1080	2,1048
M	2,0692	2,0662	2,0632	2,0601	2,0570	2,0539	2,0508	2,0477	2,0445	2,0413
N	2,0073	2,0044	2,0016	1,9987	1,9957	1,9928	1,9898	1,9868	1,9838	1,9808
O	1,9475	1,9447	1,9419	1,9391	1,9363	1,9334	1,9305	1,9276	1,9247	1,9218
P	1,8898	1,8873	1,8848	1,8823	1,8797	1,8771	1,8745	1,8719	1,8693	1,8666
Q	1,8344	1,8321	1,8298	1,8274	1,8251	1,8227	1,8204	1,8180	1,8155	1,8131
Valor Médio	**2,0114**	**2,0086**	**2,0058**	**2,0029**	**2,0000**	**1,9971**	**1,9942**	**1,9913**	**1,9883**	**1,9854**

Tabela 5.8: Performabilidade do esquema MR a partir da variação de faltas independentes

Com base nos valores apresentados nas tabelas 5.7 e 5.8 é possível constatar que a performabilidade do esquema MR diminui quando as classes selecionadas apresentam, em suas ordenações de algoritmos, os mais lentos nas posições de preferencial e *back1*. Além disto, numa mesma hierarquia a performabilidade do esquema diminui a medida em que as probabilidades de faltas relacionadas e independentes aumentam.

Caso 2: Performabilidade e faltas relacionadas

Este segundo caso de estudo compara a influência da ativação de faltas relacionadas nas performabilidades dos esquemas de tolerância a faltas. Este experimento nada mais é do que a repetição dos estudos apresentados em [Tai 93a] e [Chiaradonna 94] onde adicionamos os modelos dos esquemas CRB e MR. As faltas relacionadas consideradas neste estudo são:

- no esquema RB: faltas relacionadas entre a alternativa principal e o teste de aceitação (q_{pa}) e entre a alternativa secundária e o teste de aceitação (q_{sa});
- nos esquemas PNV, PNV-TB, PNV-TA e SCOP: faltas relacionadas envolvendo duas alternativas (q_{2v});
- no esquema CRB: faltas relacionadas entre algoritmos alternativos (q_{2v}) e faltas entre o teste de aceitação e uma das alternativas V1 (q_{pa}), V2 (q_{sa}) e V3 (q_{ta}), respectivamente;
- no esquema MR: faltas relacionadas entre o teste de aceitação e as alternativas de aplicação *pref* (q_{pa}), *back1* (q_{b1a}) e *back 2* (q_{b2a}) e entre o teste de escalonamento e o teste de aceitação (q_{ea}).

Prob. de faltas relacionadas (10^{-9})	Performabilidade ($E[M_t]$ x 10^6) dos esquemas de tolerância a faltas							
	RB	PNV	PNV-TB$^{(+)}$	PNV-TB$^{(-)}$	PNV-TA	CRB	SCOP	MR
0	2,6757	2,8838	3,2150	4,0820	1,9666	2,8838	4,0660	2,9606
20	2,5261	2,4103	2,6338	3,1869		2,4103	3,4496	2,6849
40	2,3848	2,0146	2,1576	2,4880		2,0145	2,9266	2,4350
60	2,2515	1,6838	1,7675	1,9424		1,6838	2,4829	2,2085
80	2,1256	1,4073	1,4480	1,5164		1,4073	2,1065	2,0032
100	2,0067	1,1763	1,1862	1,1839		1,1762	1,7872	1,8170
120	1,8945	0,9831	0,9717	0,9243		0,9315	1,5162	1,6482
140	1,7886	0,8217	0,7961	0,7216		0,8217	1,2864	1,4952
160	1,6886	0,6868	0,6521	0, 5633		0,6868	1,0914	1,3565
180	1,5941	0,5740	0,5342	0, 4398		0,5740	0,9259	1,2307

Tabela 5.9: Performabilidade ($E[M_t]$) dos esquemas de tolerância a faltas (Caso 2)

As tabelas 5.9 e 5.10 apresentam, respectivamente, a performabilidade ($E[M_t]$) e a probabilidade de falhas catastróficas (p_{fc}) dos esquemas de tolerância a faltas para este caso de estudo. A tabela 5.11 mostra as probabilidades de falhas benignas por erro de temporização (p_{fbt}), falhas benignas por erro de valor (p_{fbv}) e o tempo médio de execução (μ) dos esquemas analisados neste caso de estudo. Os valores da tabela 5.11 são independentes da variação da probabilidade de faltas relacionadas; isto é, permanecem constantes durante toda análise.

Prob. de faltas relacionadas (10^{-9})	Probabilidade de falhas catastróficas (p_{fc})							
	RB (10^{-9})	PNV (10^{-9})	PNV-TB$^{(+)}$ (10^{-9})	PNV-TB$^{(-)}$ (10^{-9})	PNV-TA (10^{-15})	CRB (10^{-9})	SCOP (10^{-9})	MR (10^{-9})
0	2,9194	0,0001	0,0001	0,001	0,001	0,00011	0,00005	29,195
20	22,977	60,1	60,105	60,105	60,1	60,113	40,005	43,036
40	43,036	120,1	120,106	120,106	120,1	120,113	80,005	83,153
60	63,094	180,1	180,106	180,106	180,1	180,113	120,006	123,27
80	83,152	240,1	240,106	240,106	240,1	240,114	160,006	163,38
100	100,321	300,1	300,106	300,106	300,1	300,114	200,006	203,50
120	123,269	360,1	360,106	360,106	360,1	360,114	240,006	243,62
140	143,327	420,1	420,106	420,106	420,1	420,114	280,006	283,73
160	163,386	480,1	480,106	480,106	480,1	480,115	320,006	323,85
180	183,444	540,1	540,106	540,106	540,1	540,115	360,006	363,97

Tabela 5.10: Probabilidade de falhas catastróficas (p_{fc}) (Caso 2)

Observando a tabela 5.9 é possível verificar que o esquema RB apresenta melhor performabilidade do que o PNV. Além disto, a medida em que a probabilidade de faltas relacionadas aumenta, a supremacia do esquema RB vai se acentuando em relação ao PNV. A superioridade do esquema RB em relação ao PNV é justificada através das tabelas 5.10 e 5.11: embora o tempo de execução do esquema RB de 12,5497 *ms* seja um pouco superior ao do PNV (12,035 *ms*), a probabilidade de falhas catastróficas (p_{fc}) (tabela 5.10) decorrentes de duas faltas relacionadas no esquema PNV é quase três vezes maior do que no esquema RB.

	RB	PNV	PNV-TB$^{(+)}$	PNV-TB $^{(-)}$	PNV-TA	CRB	SCOP	MR
p_{fbv} (10^{-6})	8,46	25,373	25,372	25,2724	26,5735	6,3431	16,8495	0,0025
p_{fbt}	0,0593	0,0349	0,0326	0,0101	0,0948	0,0349	0,0106	0,0396
μ	12,5497	12,035	10,8319	8,72543	16,5693	12,0435	8,75917	11,7536

Tabela 5.11: Prob. de falhas benignas por erro de valor (p_{fbv}) e de temporização (p_{fbt}) e tempo médio de execução (μ) dos esquemas analisados (Caso 2)

O esquema PNV-TB apresenta melhor performabilidade do que o PNV para pequenos valores de probabilidade (ver tabela 5.9). No Apêndice B, duas hipóteses foram assumidas na modelagem do esquema PNV-TB. Uma delas, denominada de otimista (PNV-TB$^{(-)}$), pressupõe que as duas versões mais rápidas forneçam os resultados para o primeiro ajustador. Na outra, identificada como pessimista (PNV-TB$^{(+)}$), os resultados são fornecidos ao primeiro ajustador pelas versões mais lentas. Como era de se esperar, a performabilidade do modelo PNV-TB$^{(-)}$ é melhor do que o PNV-TB$^{(+)}$ em razão do tempo médio de execução do primeiro ser menor. Nos experimentos realizados, o tempo médio de execução no PNV-TB$^{(-)}$ foi de 8,72543 *ms* enquanto o PNV-TB$^{(+)}$ apresentou seu tempo médio de execução como 10,8319 *ms*. Por sua vez, estes valores são inferiores ao tempo médio de uma iteração no esquema PNV (12,035 *ms*) (ver tabela 5.11).

A medida que a probabilidade de faltas relacionadas vai aumentado, o efeito do melhor desempenho na performabilidade do esquema PNV-TB (modelos PNV-TB$^{(-)}$ e PNV-TB$^{(+)}$) começa a diminuir em relação ao PNV, isto porque num dado momento a probabilidade de duas faltas relacionadas causar uma falha catastrófica começa a ter um peso maior no cálculo da performabilidade do que a melhora obtida com o desempenho.

A performabilidade do esquema PNV-TA é praticamente constante (1,96664 x 10^6). Um fato interessante é que a probabilidade de falhas catastróficas neste esquema é bastante pequena (da ordem de 10^{-15}, enquanto que nos outros esquemas é da ordem de 10^{-9}). Entretanto, o PNV-TA entre os esquemas analisados é o que apresenta o pior desempenho (tempo médio de 16,5693 *ms*). Este fraco desempenho é decorrente do teste de aceitação ser sempre executado após o ajustador, ou seja, independente de existir um resultado de consenso ou não o teste verifica se o resultado é correto ou não.

A performabilidade do esquema CRB é praticamente igual ao do esquema PNV. A justificativa para este fato é a baixa probabilidade de falhas benignas por erro de valores (6,34 x 10^{-6}), ou seja, considerando os atuais valores de probabilidade de faltas dificilmente um

resultado é submetido ao teste de aceitação após a execução do ajustador. A semelhança do CRB com o PNV também pode ser observada pelos tempos médios de execução de uma iteração em ambos (12,0435 *ms* no CRB e 12,035 *ms* no PNV).

O esquema SCOP apresenta melhor performabilidade do que os outros esquemas baseados no PNV (PNV-TB, PNV-TA, PNV-TA e CRB), com exceção de situações onde a probabilidade de faltas relacionadas é grande ($> 90 \times 10^{-9}$). A performabilidade do esquema PNV-TA, nestes casos, é melhor que a do SCOP. Pode se concluir que para estes valores de probabilidade de faltas relacionadas, o elevado tempo médio de uma iteração do esquema PNV-TA deixa de ser um problema; a redução da probabilidade de faltas catastróficas decorrente da execução do teste de aceitação passa a pesar decisivamente para que o PNV-TA apresente uma melhor performabilidade em relação a outros esquemas baseados no PNV. Outra observação importante é que, embora o modelo PNV-TB$^{(-)}$ seja semelhante ao SCOP, a não execução simultânea da terceira versão neste último esquema faz com que a probabilidade de falhas catastróficas no mesmo seja aproximadamente 2/3 da apresentada pelo PNV-TB$^{(-)}$. Estes valores de probabilidade tendem a se aproximar quando, devido a faltas independentes das versões (ver a análise do item 5.5.4), os resultados de consenso passam a ser obtidos no segundo ajustador do esquema SCOP.

De uma maneira geral o esquema MR apresenta uma performabilidade melhor do que os esquemas PNV, PNV-TB, CRB. Esta superioridade é resultante de uma menor probabilidade de falhas catastróficas e também do valor médio de tempo médio de uma execução (ver tabelas 5.10 e 5.11). Por outro lado, em relação aos esquemas RB, PNV-TA e SCOP os valores da tabela 5.9 devem ser analisados em determinadas faixas.

Para valores de probabilidades de faltas relacionadas pequenos ($< 60 \times 10^{-9}$) a performabilidade do esquema MR é melhor do que a do RB. A partir de um valor, situado entre 40 e 60×10^{-9}, a possibilidade de faltas relacionadas entre o teste de escalonamento e o teste de aceitação (q_{ea}) passa a influenciar negativamente no esquema, anulando a vantagem do melhor desempenho do esquema MR ($\mu = 11{,}7536$) em relação ao RB ($\mu = 12{,}5497$).

A performabilidade do MR em relação ao esquema SCOP é pior para valores de probabilidades de faltas relacionadas menores. Entretanto, para valores acima de 80×10^{-9} a performabilidade do esquema MR passa a ser melhor do que a do esquema SCOP.

Observando a tabela 5.10 é possível verificar que as probabilidades de falhas catastróficas do esquema MR não são muito diferentes do esquema SCOP, desta forma a justificativa encontrada para o fato do MR apresentar melhor performabilidade do que o SCOP para valores de probabilidade de faltas relacionadas maiores é que a partir de um determinado valor, a vantagem de selecionar dinamicamente as classes da hierarquia melhora a performabilidade média do esquema MR, de maneira que este valor médio é superior ao obtido no esquema SCOP. Observando a tabela 5.7 (performabilidade das classes do esquema MR) e a performabilidade do esquema SCOP na tabela 5.9, é possível constatar que a medida em que a probabilidade de faltas relacionadas aumenta, o número de classes do esquema MR que apresenta performabilidade melhor do que a do SCOP também aumenta.

A performabilidade do esquema MR é melhor do que o esquema PNV-TA para valores de probabilidade de faltas relacionadas inferiores a 100×10^{-9}. Acima deste valor, como já citado na análise do esquema PNV-TA, o elevado tempo médio de uma iteração devido a execução do votador e do teste de aceitação deixa de influenciar negativamente no valor de performabilidade do esquema e o benefício da redução da probabilidade de falhas catastróficas faz com que o PNV-TA apresente uma performabilidade melhor do que o MR.

Caso 3: Performabilidade e faltas independentes

Este caso de estudo analisa a influência de faltas independentes das alternativas de aplicação na performabilidade de cada esquema. Na tabela 5.12 são apresentados os valores de performabilidade dos esquemas analisados. Observando esta tabela é possível verificar que para os valores de probabilidade de faltas e de taxa de execução (inverso do tempo médio de execução) considerados, o esquema SCOP apresenta melhor performabilidade do que os demais esquemas.

Além disto, a performabilidade do esquema SCOP se mantém quase constante porque na maioria das vezes os resultados são fornecidos pelo primeiro ajustador. A fim de observar o comportamento do SCOP quando da execução do segundo ajustador nós variamos a probabilidade de faltas independentes de 10^{-2} a 9×10^{-2}, e obtivemos os resultados apresentados na tabela 5.16, onde p_1 é a probabilidade da saída do esquema ser fornecido pelo primeiro ajustador e μ o tempo médio de execução de uma iteração. Com este experimento é possível verificar melhor uma das principais características do esquema SCOP: a medida que a

probabilidade de faltas independentes aumenta a probabilidade da saída ser fornecida pelo segundo ajustador é maior, aumentando assim o tempo médio de execução da iteração. No limite, isto é, quando todas saídas são fornecidas pelo segundo ajustador, o desempenho (tempo médio de iteração) do esquema SCOP é pior do que o do esquema PNV, pois ao contrário do PNV, as versões não são executadas paralelamente.

Prob. de faltas independentes (10^{-3})	Performabilidade ($E[M_t]$) x 10^6							
	RB	PNV	PNV-TB$^{(+)}$	PNV-TB$^{(-)}$	PNV-TA	CRB	SCOP	MR
0	2,1508		1,4482	1,5166	1,9663	1,4067	2,1113	2,0114
1	2,0652		1,4481	1,5166	1,9662	1,4066	2,1096	2,0086
2	1,9826		1,4481	1,5165	1,9661	1,4064	2,1079	2,0058
3	1,9029		1,4480	1,5164	1,9660	1,4062	2,1062	2,0029
4	1,8259	1,4072	1,4479	1,5163	1,9659	1,4060	2,1045	2,0000
5	1,7516		1,4478	1,5162	1,9658	1,4058	2,1028	1,9971
6	1,6799		1,4477	1,5161	1,9656	1,4056	2,1011	1,9942
7	1,6108		1,4476	1,5160	1,9655	1,4054	2,0993	1,9913
8	1,5441		1,4474	1,5159	1,9653	1,4051	2,0976	1,9883
9	1,4798		1,4473	1,5157	1,9652	1,4048	2,0959	1,9854

Tabela 5.12: Performabilidade ($E[M_t]$) dos esquemas de tolerância a faltas (Caso 3)

Prob. de faltas independentes (10^{-3})	Tempo Médio x 10^{-3} (μ)							
	RB	PNV	PNV-TB $^{(+)}$	PNV-TB $^{(-)}$	PNV-TA	CRB	SCOP	MR
0	12,5190		10,8273	8,71491	16,5683		8,71643	11,7197
1	12,6246		10,8289	8,71853	16,5681		8,73167	11,7313
2	12,7302		10,8305	8,72214	16,5678		8,74687	11,7430
3	12,8359		10,8320	8,72576	16,5675		8,76204	11,7547
4	12,9415	12,0435	10,8336	8,72938	16,5672	12,0435	8,77718	11,7663
5	13,0471		10,8352	8,73299	16,5668		8,79229	11,7780
6	13,1528		10,8368	8,73661	16,5664		8,80737	11,7897
7	13,2584		10,8384	8,74022	16,5660		8,82242	11,8015
8	13,3641		10,8400	8,74384	16,5655		8,83744	11,8132
9	13,4697		10,8416	8,74745	16,5651		8,85243	11,8249

Tabela 5.13: Tempo médio de execução (μ) dos esquemas analisados (Caso 3)

No esquema RB, o acréscimo na probabilidade de faltas independentes corresponde na diminuição na performabilidade do esquema. As razões são as seguintes:

- o tempo médio de uma iteração (tabela 5.13) do esquema RB aumenta, pois a medida que o valor da probabilidade de faltas independentes vai se elevando, a freqüência da execução de todas as alternativas de aplicação por iteração também aumenta;

- o crescimento do número de resultados incorretos submetidos ao teste de aceitação, determina um aumento no número de falhas catastróficas, isto é, maior é o número de resultados incorretos que o teste de aceitação aceita como corretos;

Prob. de faltas independentes (10^{-3})	Probabilidade de falhas catastróficas (p_{fc})							
	RB (10^{-8})	PNV (10^{-8})	PNV-TB $^{(+)}$ (10^{-8})	PNV-TB $^{(-)}$ (10^{-8})	PNV-TA (10^{-13})	CRB (10^{-8})	SCOP (10^{-8})	MR (10^{-8})
0	8,00012	24,01	24,0100	24,0100		24,0231	0,16	12,572
1	8,10822		24,0102	24,0102		24,0261	16,0002	12,586
2	8,21653		24,0104	24,0104		24,0295	16,0004	12,602
3	8,32503		24,0106	24,0106		24,0333	16,0006	12,618
4	8,43373		24,0108	24,0126	2,401	24,0373	16,0008	12,634
5	8,54263		24,0110	24,0128		24,0417	16,0010	12,651
6	8,65173		24,0112	24,0130		24,0465	16,0012	12,668
7	8,76103		24,0114	24,0133		24,0515	16,0014	12,685
8	8,87053		24,0116	24,0135		24,0569	16,0016	12,703
9	8,98023		24,0118	24,0137		24,0626	16,0018	12,722

Tabela 5.14: Probabilidade de falhas catastróficas (p_{fc}) (Caso 3)

Prob. de faltas independentes (10^{-3})	Probabilidade de falhas benignas por erro de valor (p_{fbv})							
	RB (10^{-6})	PNV (10^{-6})	PNV-TB$^{(+)}$ (10^{-6})	PNV-TB $^{(-)}$ (10^{-4})	PNV-TA (10^{-13})	CRB (10^{-4})	SCOP (10^{-6})	MR (10^{-9})
0	0,000004	0,001	0,000003	0,000003	2,42783	6,03854	0,0199	0,001
1	1,00416	3,005	3,004	3,004	2,99241	7,44999	2,4146	1,0014
2	4,00831	11,997	11,996	11,996	3,61579	9,00844	8,7830	1,2953
3	9,01246	26,9649	26,9639	26,9639	4,29785	10,7136	19,1012	1,3209
4	16,0166	47,8969	47,8959	47,8959	5,03846	12,5651	33,3454	1,3508
5	25,0207	74,7808	74,7798	74,7798	5,83752	14,5628	51,4921	1,3852
6	36,0249	107,605	107,604	107,604	6,69490	16,7062	73,5176	1,4240
7	49,0290	146,357	146,356	146,356	7,61048	18,9952	99,3985	1,4675
8	64,0331	191,025	191,024	191,024	8,58414	21,4293	129,111	1,5155
9	81,0372	241596	241,595	241,595	9,61576	24,0084	162,633	1,5683

Tabela 5.15: Probabilidade de falhas benignas por valor (p_{fbv}) (Caso 3)

As performabilidades dos esquemas PNV, PNV-TB e CRB são inferiores ao do RB. Em relação ao esquema PNV-TA, o RB apresenta melhor performabilidade para valores pequenos de probabilidade de faltas independentes (inferiores a 2×10^{-3}).

Prob. de faltas independentes (x 10^{-2})	Esquema SCOP								
	1	2	3	4	5	6	7	8	9
p_1	0,9800	0,9606	0,9415	0,9229	0,9046	0,8868	0,8692	0,8521	0,8353
μ	8,8673	9,0153	9,1603	9,3024	9,4416	9,5780	9,7116	9,8425	9,9707

Tabela 5.16: Probabilidade da saída do esquema SCOP ser fornecida pelo primeiro ajustador e tempo médio de execução de uma iteração

A performabilidade do esquema MR também diminuiu a medida em que a probabilidade de faltas independentes aumenta. As razões são as mesmas que as apresentadas para o RB, ou seja, a medida que a probabilidade de ocorrência de faltas aumenta, as diferentes alternativas vão sendo executadas com maior freqüência e, como conseqüência, o tempo médio de execução aumenta. Mas se compararmos os valores da tabela 5.12, verificamos que as variações sofridas pelo MR são menores que as experimentadas pelo RB. A performabilidade do MR permanece bem próxima dos valores obtidos pelo SCOP ao longo da faixa considerada de probabilidades de faltas independentes. O esquema MR apresenta melhores resultados de performabilidade que o RB, PNV, PNV-TB, PNV-TA e o CRB.

5.5.3 Considerações gerais

A performabilidade do esquema MR foi calculada a partir de um conjunto de classes compostas pelos arranjos 3 a 3 de um total de 7 algoritmos. No esquema MR, os valores de performabilidade das classes são diferentes: variando de acordo com a alternativa preferencial selecionada. Considerando que a distribuição dos dados de entrada é uniforme, a performabilidade do esquema MR foi calculada a partir dos valores médios de performabilidade das classes.

Os resultados obtidos no segundo caso de estudo permitem verificar a influência de faltas relacionadas na performabilidade dos esquemas. Os valores de probabilidade de faltas independentes nos algoritmos alternativos no caso 2 são baixos, fazendo com que na maioria dos experimentos a execução dos esquemas envolva com maior freqüência poucas alternativas. Por exemplo, nos esquemas RB e MR as saídas são fornecidas em quase 99.99% das iterações pela alternativa principal (ou preferencial) e no esquema SCOP o resultado de consenso é obtido com maior freqüência no primeiro ajustador. Esta particularidade faz com que o tempo médio de execução destes esquemas sejam praticamente constante, isto é, não variam com o aumento ou diminuição da probabilidade de faltas relacionadas (ver tabela 5.16). Do ponto de vista da Segurança de Funcionamento, estes experimentos permitem verificar que nos esquemas baseados no PNV existe uma relação entre o número de versões executadas e a probabilidade de falhas catastróficas. Por exemplo, a execução de três versões nos esquemas PNV, PNV-TB e CRB faz com que a relação entre as probabilidades de falhas

catastróficas e a probabilidade de faltas relacionadas seja três. Por outro lado, esta relação é dois no esquema SCOP quando, inicialmente, duas versões são executadas.

O caso 3 permite verificar as características dos esquemas analisados quando da ativação de faltas independentes nos algoritmos de aplicação. Observando os valores de performabilidade na tabela 5.12 e de probabilidade de falhas catastróficas na tabela 5.14 é possível constatar que a ativação de faltas independentes nos algoritmos alternativos de aplicação penaliza mais os esquemas que executam as alternativas seqüencialmente, ou seja, os esquemas RB e MR. Entretanto, a performabilidade destes modelos (MR e RB) quando do aumento das probabilidades de faltas independentes seriam ainda piores se o mecanismo de teste de aceitação não fosse utilizado. Nos esquemas que executam paralelamente as diferentes versões, como é o caso do PNV, PNV-TB, PNV-TA e CRB, a influência de faltas independentes na performabilidade é mínima, pois a falha de uma versão é mascarada pelo ajustador a partir dos resultados corretos. O esquema SCOP também sofre a influência de faltas independentes (ver tabela 5.12). Entretanto, para valores de probabilidade de faltas independentes baixos, esta influência é mínima na performabilidade do esquema. Desta forma, a performabilidade diminui com o aumento da probabilidade de faltas independentes nos esquemas que fazem uso de mecanismos de recuperação de erro (por exemplo nos esquemas MR e RB) e no esquema SCOP em que a terceira versão é executada quando o resultado de consenso não é obtido pelo primeiro ajustador. Esta aumento é principalmente causado pela elevação do tempo médio de execução destes esquemas. O esquema mais suscetível às variações de probabilidades de faltas independentes é o RB. O MR e o SCOP apresentam variações mínimas de performabilidade e de tempo médio de execução em função das variações de faltas independentes.

Considerando os resultados de análise nos três casos apresentados, podemos observar que, embora as estruturas dos esquemas sejam semelhantes, as seqüências de execução dos mecanismos e das alternativas de aplicação são determinantes para os diferentes valores de performabilidade dos esquemas. Por exemplo, embora os esquemas PNV-TA e MR apresentem os mecanismos de teste de aceitação e o ajustador, os valores de performabilidade são diferentes: o tempo médio de execução do PNV-TA é bastante elevado e a probabilidade de falhas catastróficas é bastante reduzida. No esquema MR o tempo médio de execução é

menor. Desta forma, para valores de probabilidades de faltas relacionadas menores que 100×10^{-9}, o uso do MR é preferível em relação ao PNV-TA.

5.6 Conclusão

Neste capítulo foi apresentado um estudo comparativo do esquema MR com os principais esquemas apresentados na literatura. A medida utilizada como parâmetro de referência de comparação foi a performabilidade que combina características de Segurança de Funcionamento e de Desempenho de um sistema.

Com base nos casos de estudos realizados neste capítulo é possível constatar uma melhora na performabilidade de esquemas que apresentam algum tipo de flexibilidade, como é o caso do SCOP, PNV-TB e o MR. Entretanto, esta melhora só é válida em algumas faixas de probabilidades, sendo que em determinadas situações os valores de performabilidade destes esquemas podem ser piores do que dos esquemas tradicionais, como o RB e PNV.

Com base nos experimentos realizados foi possível analisar o comportamento do mecanismo de teste de escalonamento introduzido no esquema MR com o intuito de flexibilizar e selecionar alternativas mais apropriadas aos dados de entrada. De uma maneira geral a performabilidade do esquema MR atende as expectativas, sendo na média superior a da maioria dos esquemas em todas as faixas analisadas.

CAPÍTULO 6

UM MODELO DE EXECUÇÃO ASSÍNCRONO TEMPORIZADO PARA O ESQUEMA MR

O propósito deste capítulo é apresentar alguns resultados experimentais obtidos a partir de um protótipo do MR implementado no LCMI/UFSC. Como foi colocado nos capítulos anteriores, a implementação do esquema MR depende da técnica de replicação utilizada e também do tipo de sincronismo adotado no modelo de execução.

A principal característica do MR que é a seleção dinâmica da classe de algoritmos alternativos se mantém, independente da técnica de replicação ou do sincronismo no modelo de execução. Porém, o desempenho, a semântica de falhas e a delimitação temporal no processamento são fatores que dependem do modelo de execução.

Com objetivo então de exploramos as potencialidades do MR, descrevemos neste capítulo um modelo de execução assíncrono temporizado ("*timed asynchronous*"), baseado na técnica de replicação líder/seguidores. Um resumo deste trabalho foi apresentado em [Nacamura 95]. Antes, colocamos com base na literatura, as principais características e diferenças entre sistemas síncronos e assíncronos temporizados. Em seguida, o modelo introduzido é descrito e analisado no sentido de estabelecer as semânticas de falhas e os limites de tempo (probabilistas). Uma análise de pior caso é utilizada para tal. As implementações realizadas e os testes desenvolvidos no sentido de validar estas implementações são descritos no capítulo.

6.1 Sistemas síncronos, semi-síncronos e assíncronos

Os sistemas distribuídos, dependendo das características do suporte de comunicação e das velocidades relativas entre seus processadores são classificados em **síncronos** e **assíncronos**. Os **sistemas distribuídos síncronos** apresentam um limite máximo (Φ) entre as velocidades relativas de processamento entre dois nós quaisquer do sistema e também um limite máximo (δ) na duração de uma comunicação entre seus processadores.

Os **sistemas síncronos** ou modelos síncronos de execução são normalmente usados em aplicações de tempo real crítica (*hard real-time systems*). Estes sistemas se caracterizam por cargas computacionais conhecidas a priori (carga estática). Os relógios precisam estar sincronizados, no sentido de estabelecer uma base de tempo global. O escalonamento que determina as velocidades relativas limitadas no processamento é estático, com ativação *timer trigger* como no sistema MARS [Kopetz 89a], ou ainda dinâmico, baseado em prioridades fixas com ativações por eventos [Tindell 94].

Os **sistemas assíncronos** foram inicialmente caracterizados pela não existência dos limites Φ e δ; ou seja, nestes sistemas para uma entrada e um estado, é especificado o próximo estado ou saída sem imposição de qualquer restrição de tempo sobre a ocorrência desta transição. Processadores podem então evoluir em seus processamentos e comunicações em tempos arbitrários. Fica difícil diferenciar nesses sistemas a ausência de uma mensagem por falha de *crash* ou por comunicações e processamentos extremamente lentos. A conseqüência dessa inexistência de restrições temporais é que fica impossibilitada a implementação de serviços essenciais para a tolerância a faltas em sistemas assíncronos. Como exemplos podemos citar protocolos de acordo, difusão atômica, *membership*, etc., cujas implementações ficam impossibilitadas em ambientes assíncronos [Ficher 85]. Contudo, uma grande quantidade de sistemas distribuídos não se enquadram na definição de modelos síncronos e mesmo assim, apresentam soluções implementadas para os serviços citados.

No sentido de contornar esta dificuldade, em [Dwork 88] são apresentados dois modelos que caracterizam **os sistemas distribuídos semi-síncronos**, intermediários entre os síncronos e assíncronos. No primeiro modelo, existem os limites para Φ e δ, porém estes limites não são conhecidos. Na segunda proposta, os limites para Φ e δ existem, porém só são garantidos.

após um tempo T não conhecido. Em ambos modelos é admitido portanto restrições temporais.

Em [Cristian 95] foi introduzido o modelo de **sistema assíncrono temporizado** (***timed asynchronous system***). Nesse modelo mensagens podem ser perdidas, ou sofrer atrasos no suporte de comunicação; porém, em grande parte, as mensagens chegam dentro de um tempo conhecido δ (limite probabilista). As perdas ou atrasos que superam o limite são tratados como falhas de omissão ou de desempenho no suporte de comunicação. A delimitação no processamento é colocada em [Cristian 95] na forma de um atraso máximo s de ativação no escalonamento. O escalonamento em um ambiente assíncrono é *event-trigger* (dinâmico) e as cargas computacionais não são limitadas ou conhecidas a priori. Os processamentos cujos atrasos no escalonamento excedem s (delimitação probabilista) produzem falhas de desempenho no sistema. A nível de processador, as semânticas de falhas assumidas são de desempenho e de paragem (*crash*) no modelo proposto.

O modelo assíncrono temporizado, pelas características citadas, contempla a maioria das implementações de ambientes e suportes distribuídos existentes atualmente.

6.1.1 Grupos síncronos e assíncronos temporizados

As noções de sistemas síncronos e assíncronos temporizados são discutidas em [Cristian 96] no contexto de grupos. O modelo síncrono de grupo é fundamentado em protocolos síncronos de difusão atômica. Nesses protocolos, todos os atrasos têm que ser menor que δ nas comunicações ponto-a-ponto. Os atrasos de escalonamento por sua vez são limitados, inferiores a s. Com premissas de falhas em nós e em *links* e considerando ainda uma redundância espacial no suporte de comunicação e a taxa de difusões inferiores a capacidade da rede (limitações no controle de fluxo sobre a rede), o atraso de uma difusão apresenta um limite Δ. Ou seja, um processador executando a difusão de uma mensagem m no instante t, em $t+\Delta$ ou todos os processadores corretos despacham os códigos de aplicação correspondentes à mensagem ou não.

Um grupo assíncrono temporizado é caracterizado pela incerteza na comunicação. O desejado é que a maioria das comunicações *timed* se realizem, ou seja, que se concretizem dentro das

restrições de tempo especificadas (*timeouts*). As condições adversas nesses sistemas são provocadas pela cargas não limitadas ou pelas perdas de mensagens.

Nos grupos assíncronos temporizados, a noção de **estar conectado** é mais representativa do que **estar correto**. Um processador está **conectado** em um intervalo de tempo quando recebe todas as mensagens difundidas no grupo. Um sistema assíncrono é dito ***estável*** em um intervalo de tempo quando não ocorrem alterações no *membership* do grupo (ou por falha ou por *restart* de processador), e porque os processos ou estão conectados ou desconectados no grupo no intervalo de tempo considerado.

Em grupos assíncronos (*timed asynchronous*) as propriedades temporais são mais fracas; os atrasos com que são percebidas as alterações de *membership* (falhas e reinserções de processos) e os pedidos de serviço são limitados somente quando condições de estabilidade são mantidas [Cristian 96]. Um sistema assíncrono relativamente bem dimensionado em termos de carga suportável deve provavelmente apresentar longos períodos de estabilidade. Os atrasos são limitados *(bounded)* com base na probabilidade que as condições de estabilidade se mantenham em tempo de execução. O conhecimento da distribuição dos atrasos é necessário para estimar esta probabilidade.

6.2 Execução assíncrona temporizada do MR implementado segundo o modelo de replicação líder/seguidores

Antes de descrever a implementação do MR é importante a caracterização do modelo de execução. Assumimos como premissa a execução do MR em um sistema distribuído assíncrono temporizado. O ambiente de execução usado na implementação do MR é constituído de estações SPARCstation conectadas através de uma rede local Ethernet. No modelo de execução é assumido escalonamento dinâmico com ativações *event-trigger* (as replicações são ativadas por mensagem no MR). As cargas nas estações, a priori, são conhecidas e limitadas e os equipamentos homogêneos; nessas condições é fácil de verificar o limite Φ sobre as velocidades relativas entre seus processadores. Mas mesmo que as cargas não fossem limitadas, trabalharíamos com um Φ probabilista, estimado a partir das estatísticas de atrasos dos tempos de respostas das réplicas.

A comunicação no modelo de execução do MR envolve comunicações ponto-a-ponto e de grupo. Em ambas comunicações, assumimos limites probabilistas. Estes limites são estipulados a partir de distribuições obtidas em função da carga e do número máximo de nós. A escolha destes valores de limite máximo implica que quanto maior são estes valores menor é a probabilidade de falhas de desempenho no suporte, mas com a penalização sobre os tempos de resposta do servidor MR que serão maiores nestes casos. Em sentido inverso, com valores menores nos limites, o desempenho do MR é melhor, porém a probabilidade de falhas serão maiores.

Em relação a comunicação ponto-a-ponto assumimos as condições probabilistas de um limite máximo δ como definido em [Cristian 89]. As comunicações de grupo no modelo de execução do MR se dão através de um protocolo de difusão *clockless* com características de assíncronia, segundo a classificação introduzida em [Veríssimo 89] e apresentada no capítulo 3 deste texto. Neste modelo as comunicações de grupo se dão fazendo uso de difusões com ordem causal. A entrega da mensagem em todos os participantes do grupo apresenta um limite máximo Δ. Em [Birman 91] é apresentado um estudo detalhado com medidas quantitativas do desempenho do protocolo CBCAST de difusão causal, implementado no suporte ISIS. Estas medidas foram feitas em um sistema composto por estações SUN e uma rede Ethernet com carga mediana. Os valores de Δ são calculados em função do número de processos no grupo e dos tamanhos de mensagens.

Em resumo, neste trabalho assumimos os valores dos limites probabilistas Δ e δ tomando como base os trabalhos de [Cristian 89] e [Birman 91]. As violações destes limites são detectadas através de *timeouts* e tratadas como falhas de desempenho no suporte de comunicação.

6.2.1 Modelo de execução

Na replicação líder/seguidores, apresentada no item 3.2.2, um dos processos do grupo é designado *líder* e os demais *seguidores*. O processo líder se distingui dos demais por desempenhar a função de coordenação no processamento replicado. No modelo de execução proposto, as interações do cliente com o grupo são feitas através do líder, usando comunicações ponto-a-ponto. O processo líder é responsável pela manutenção da ordenação de todos os pedidos submetidos ao grupo.

A abordagem líder/seguidores adotada é bloqueante [Budhiraja 92]: o servidor líder após a execução do serviço se bloqueia, esperando pelos resultados gerados pelos seguidores e através de uma função de voto majoritário obtém o resultado de consenso que é enviado como resposta ao cliente.

A principal vantagem desta abordagem sobre as baseadas na replicação ativa ou máquina de estado [Schneider 90] é a visão transparente que o cliente tem do grupo, isto é, o cliente interage com o grupo da mesma forma que interage com um processo *stand-alone*. Além disto, as propriedades de acordo e ordenação necessárias para a manutenção do determinismo de réplica são garantidas a um custo bem menor que em replicações ativas (item 3.2.5).

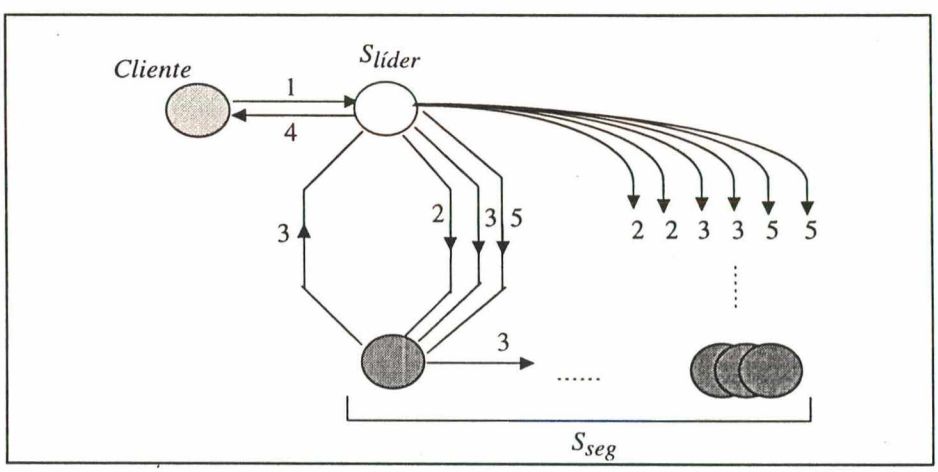


Figura 6.1: Passos de execução de um serviço

O serviço é então implementado na forma de um MR a partir de um grupo de *N* servidores, sendo que um destes processos assume o papel do líder $(S_{líder})$ e os *N-1* servidores restantes são os seguidores (S_{seg}). Na figura 6.1 são apresentados de forma esquemática os passos de execução de um pedido de serviço. O cliente interage com o grupo enviando suas chamadas de serviço ao líder $(S_{líder})$ em comunicação ponto-a-ponto (arco 1). As mensagens recebidas do cliente passam por uma verificação de mensagens antigas. Se for a retransmissão da mensagem/pedido do ciclo anterior, o servidor líder envia ao cliente a resposta correspondente para o cliente que são mantidas antes do início de um novo ciclo. Se a mensagem é nova, $S_{líder}$ difunde (difusão causal) a mensagem recebida aos servidores seguidores (arco 2). As réplicas executam então os algoritmos de aplicação, segundo o esquema MR (teste de escalonamento, seleção e execução da replicação e teste de aceitação). Os resultados são difundidos no grupo usando serviços de difusão causal (arco 3) e submetidos em votação nos

servidores. Por fim, o resultado da votação é enviado ao cliente (arco 4). O arco 5 representa o envio em difusão pelo líder da mensagem *"fim_de_ciclo_de_processamento"* (msg_{fim_cp}). Esta mensagem encerra o ciclo e sinaliza a terminação correta do processamento.

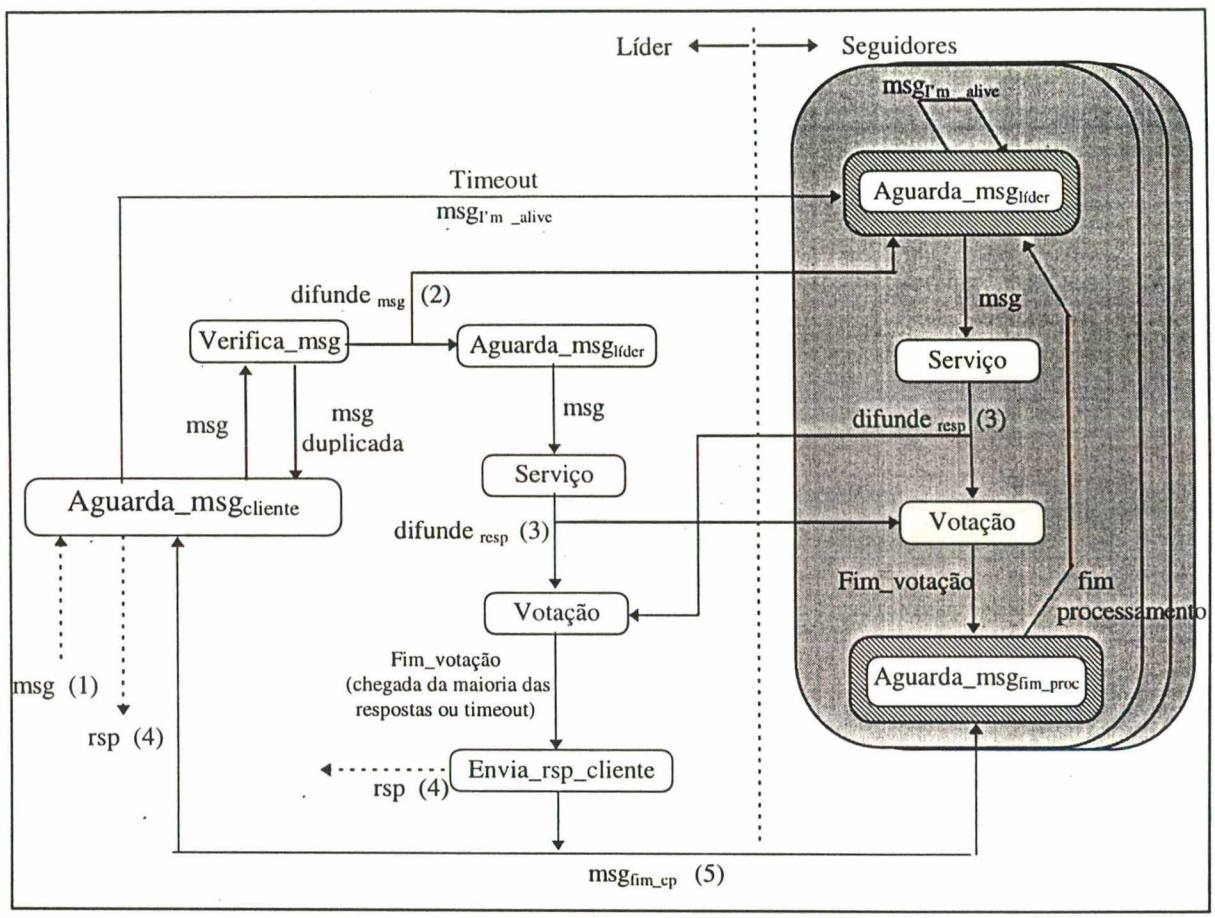


Figura 6.2: Diagrama de estado do modelo de replicação

A Figura 6.2 apresenta o diagrama de estados do modelo de replicação líder/seguidores proposto. Os estados *aguarda_msg*$_{cliente}$, *verifica_msg* e *envia_ rsp_cliente* do processo líder estão diretamente relacionados às interações com o cliente e não fazem parte da execução dos processos seguidores. As transições enumeradas são associadas a eventos externos (mensagens) e estão em correspondência com os arcos representados na figura 6.1. A mensagem msg_{fim_cp} também é utilizada pelos processos seguidores para detectar a falha do líder em um ciclo de processamento (item 6.2.2). Desta forma, as falhas do processo líder são detectadas pelos seguidores nos estados *aguarda_msg*$_{líder}$ e *aguarda_msg*$_{fim\_cp}$ (ver item 6.2.2).

Embora todos os servidores (líder e seguidores) executem o processamento e a votação, somente o processo $S_{líder}$ envia ao cliente o resultado do serviço em comunicação ponto-a-ponto. A principal vantagem de todos os processos servidores executarem a votação é que em qualquer instante, no caso de falha do líder, um dos processos seguidores pode assumir o papel de líder e concluir o serviço enviando a mensagem de resposta ao cliente, sem a necessidade de uma nova votação. O uso da difusão causal nas comunicações no grupo de servidores se justifica pelo fato de que a ordenação de pedidos é imposta no modelo pelo líder. Sendo assim, a única ordenação necessária nas comunicações de grupo resume se em garantir que as mensagens de pedidos de serviço antecedam as suas respectivas mensagens de resultados, e os mecanismos de ordem causal satisfazem a essa necessidade.

6.2.2 Gerenciamento de faltas no modelo de execução

As hipóteses de faltas assumidas neste modelo de execução semi-síncrono do MR baseado na replicação líder-seguidores são:

- **Nos servidores (nos nós):** a nível do processamento dos algoritmos MR, temos a manifestação de faltas por valor e de *crash*. As faltas por valor podem apresentar as semânticas **Bohrbug** e **Heisenburg** [Gray 86]. A semântica Heisenburg corresponde a um *bug* que causa faltas transientes. Por outro lado, a semântica Bohrbug representa a ocorrência de faltas permanentes. As faltas Bohrbug podem ser toleradas a partir da diversidade de projeto, enquanto que as faltas Heisenburg caracterizadas por sua natureza transitória podem ser toleradas a partir da reexecução do mesmo software, uma vez que os softwares geralmente falham quando submetidos a entradas particulares ou condições excepcionais assumidas pelo ambiente de execução. A reexpressão da entrada (diversidade de dados) ou alterações do ambiente de execução determinam a não persistência destas faltas;

- **Na rede de comunicação:** neste nível assumimos perdas ocasionais de mensagens e o não particionamento da rede. Eventuais alterações de mensagens entre transmissões são detectadas e as mensagens descartadas. Nestas condições, é admitido a ocorrência de faltas de omissão e de desempenho no suporte de comunicação.

6.2.2.1 Detecção e tratamento de faltas no modelo de execução

O gerenciamento de faltas no modelo é baseado em contagens de falhas ocorridas durante os ciclos de processamento. A partir destas contagens são identificados os servidores faltosos e ativados os tratamentos de faltas correspondentes. Todo este procedimento está centrado no servidor líder que no final da votação monta o vetor de falhas *vf[0..N-1]* associado ao ciclo de processamento corrente. Neste vetor são registradas indicações de estado de todos os servidores. A referência usada na obtenção destas indicações é o resultado de consenso no ciclo. A figura 6.3 apresenta os algoritmos de gerenciamento e tratamento de faltas no modelo.

```
#define    0 CORR          /* resultado correto */
#define    1 ERRO_ABST     /* erro de abstenção */
#define    2 ERRO_TEMP     /* erro de temporização */
#define    3 ERRO_SOFT     /* erro de software */
#define    X FALT_CONS     /* número de faltas consecutivas toleradas */
#define    Y FALT_ALT      /* número de faltas alternadas toleradas */

int my_indice;        /* índice do processo
                                    corrente */

/* Processo líder */

task gerenc_faltas ( ) {
  typemsg  msg[N-1], valor_consenso;
  int  i=0, vf[N-1] ;
     do {
        if (msg[i] == valor_consenso)
            vf[i] = CORR;
        else  if (msg[i] == NULL)
                  vf[i] =  ERRO_TEMP;
        else   if  (msg[i] == -1)
                  vf[i] = ERRO_ABST;
        else   vf[i] = ERRO_SOFT;
        i++;
     } while (i<=N-1 && timeout !=0)
    broadcast vf to grupo_servidores;
} /* end_task */
```

```
int x,y;        /* contadores de faltas consecutivas
                                    e alternadas

/* Processos líder e seguidores */

task trat_faltas ( ) {
    int vf[N], flag = 0;
    receive vf from processo_líder;
    if (vf[my_indice] != CORR) {
       y++;
       if  (flag == 0)
           flag=1;
       else  x++;
       if ( x >= X  ||  y >= Y  ) {
          /* solicita saída do grupo */
     }
    else { flag =0; x =0; }
} /* end_task */
```

Figura 6.3: Algoritmo de detecção e tratamento de faltas

Baseado no resultado de consenso, o processo líder através da ***task*** *gerenc_faltas* (ver figura 6.3) assinala no campo *vf[i]*, referente a um servidor *i* ($0 \leq i \leq N-1$), se o processamento do mesmo foi correto ou faltoso (falha de software, falha de temporização ou o servidor se absteve). Uma vez montado este vetor, o processo líder difunde para o grupo de servidores o vetor *vf[i]* na mensagem msg_{fim_cp}. Com base no histórico dos ciclos de processamento

montado a partir destas informações *vf[i]*, um servidor (inclusive o líder) pode permanecer ou ser retirado do grupo. As estatísticas contidas, neste histórico apresentam limites: um servidor falhando em **X** ciclos consecutivos e/ou em **Y** ciclos alternados deve ser retirado do grupo. O algoritmo de tratamento de faltas do modelo implementado também é mostrado na figura 6.3 (***task trat_faltas***). Um processo faltoso pode ainda permanecer no grupo (o limite permitido de falhas, ainda não foi atingido); neste caso, o seu estado deve ser alterado de maneira que a consistência do grupo seja mantida. A tabela 6.1 resume as condições que permite reconhecer os tipos de faltas assumidos neste modelo de execução.

De maneira a facilitar o gerenciamento do grupo a solicitação de saída do grupo é feita, normalmente, pelo próprio processo faltoso no instante em que uma das condições da tabela 6.1 é atingida.

TIPOS DE FALTAS	CONDIÇÕES
Faltas de valor (Bohrburg)	#ERROS_SOFTWARE ≥ ***X***
Faltas de valor (Heisenburg)	#ERROS_SOFTWARE ≥ ***Y***
Faltas *crash*, omissão, temporização	#ERROS_TEMPORIZAÇÃO ≥ ***X***
Faltas de temporização intermitente	#ERROS_TEMPORIZAÇÃO ≥ ***Y***

Tabela 6.1: Tipos de faltas

6.2.2.2 Protocolo de diagnose de falhas de *crash* no servidor líder

Os procedimentos citados acima não são suficientes para detectar todas as situações de falhas do líder ($S_{líder}$) como, por exemplo, o *crash*. O *crash* de $S_{líder}$ durante um ciclo de processamento pode ser detectado pelos seguidores através do não recebimento da mensagem *"fim_de_ciclo_de_processamento"* (msg_{fim_cp}). Porém, em situações de ausência de pedidos de serviço no grupo, é necessário algum suporte adicional para detectar o *crash* de $S_{líder}$. Existem duas maneiras pelas quais os processos seguidores podem detectar a falha *crash* de $S_{líder}$ quando não existe nenhum pedido de serviço sendo processado pelo grupo:

- na primeira abordagem, todos os processos seguidores enviam periodicamente em π unidades de tempo a mensagem *"Are-you alive?"* a $S_{líder}$ que responde a cada uma destas mensagens recebidas;
- na segunda abordagem, $S_{líder}$ difunde periodicamente em π unidades de tempo a mensagem *"I'm alive"* a todos os processos seguidores.

Neste modelo de execução a detecção de falha do processo líder é feita através da segunda abordagem onde: o número de mensagens a nível de aplicação é menor do que na primeira abordagem em virtude do uso de um protocolo de difusão e as propriedades de acordo e de ordenação de mensagens enviadas pelos processos do grupo são garantidas pelo próprio protocolo de difusão.

Num sistema *timed asynchronous* a não recepção das mensagens msg_{fim_cp} ou $msg_{I'm\ alive}$ por um dos processos seguidores não significa que o líder apresente uma falha *crash*. Por exemplo, a mensagem pode ter sido enviada, porém não ter sido entregue devido a ocorrências de falhas de omissão ou temporização no suporte de comunicação. Desta forma, na ausência da recepção das mensagens msg_{fim_cp} ou $msg_{I'm\ alive}$ tem início a etapa de diagnose que visa detectar a falha do processo líder. O primeiro servidor seguidor que detectar estas ausências deve difundir a mensagem *ocorrência de falhas* (msg_{oc_fh}) no grupo MR. Por sua vez, todos os servidores membros do grupo, ao receberem msg_{oc_fh}, difundem no grupo *ACKs* ou *NACKs*, indicando o recebimento ou não das mensagens de sincronização (msg_{fim_cp} ou $msg_{I'm\_alive}$). Com base no número de *ACKs* ou *NACKs* difundidos é possível reconhecer o tipo de falha ocorrido no $S_{líder}$ (se é que o mesmo tenha falhado). O $S_{líder}$ se estiver ativo deve participar desse processo de diagnose.

Se mais de um servidor seguidor detectar a ausência das mensagens msg_{fim_cp} ou $msg_{I'm\ alive}$, gerando mais de uma mensagem de *ocorrência de falha*, esse fato não acarreta nenhuma implicação no protocolo de diagnose descrito. Os seguidores que já responderam a primeira sinalização com *ACKs* ou *NACKs* simplesmente descartam a segunda msg_{oc_fh}. A ordenação causal determina na precedência em todos os nós da mensagem msg_{oc_fh} de seus *ACKs* ou *NACKs* correspondentes. Todos os processos corretos terão estas mensagens nessa ordem causal.

Quando se configura no processo de diagnose a falha de *crash* do líder, o grupo é rearranjado e um dos servidores seguidores assume o papel de líder. A substituição do $S_{líder}$ é feita pelo servidor correto com o menor índice no grupo. Os algoritmos da figura 6.4 explicitam o funcionamento do protocolo de diagnose de faltas nos servidores líder e seguidores.

```
/* Processos Servidores */

task fim_proc ( ) {
    receive msg_fim_cp to líder with timeout = t1;
    if (msg_fim_cp == NULL) {
      if (msg_oc_fh != NULL) {
        broadcast msg_oc_fh to grupo_servidores;
} /* end_task */

task diagnose_faltas ( ) {
  int rsp[N-1],i=0, flag=0;
  receive msg_oc_fh to servidor;
  if (flag==0 ) {
      if (msg_fim_cp == NULL)
        broadcast ACK to grupo_servidores;
      else
        broadcast NACK to grupo_servidores;
        flag=1;
   do {
     receive rsp[i] from grupos_servidores;
     i++;
   } while ((i <= (N-1) ) ∨ (timeout !=0))
   if ((rsp[líder] == NULL) ∧ (maioria das respostas ACK)) {
        /* falha crash do processo líder */
        /* O processo de menor índice assume o papel de líder */
   }
    else
     /* falha de omissão do processo que início o protocolo
              de detecção de falha */
  }
} /* end_task */
```

Figura 6.4: Algoritmo de diagnose de faltas

6.2.3 Análise de pior caso do modelo de execução assíncrono temporizado líder/seguidores do MR

Neste item é apresentado a análise de pior caso do desempenho de um serviço MR, implementado segundo o modelo descrito no item anterior. Para o cálculo do tempo de resposta máximo, duas hipóteses são assumidas:

H_1) Pior caso de falhas de valor: escalonamento inicial do algoritmo preferencial da classe na hierarquia de especialização e ocorrência de $(k-1)$ falhas sucessivas nos testes de aceitação (classe com k algoritmos). O último algoritmo alternativo, o $k^{ésimo}$, é executado com sucesso e o resultado é validado pelo teste de aceitação;

H_2) Pior caso de falhas de *crash*: *(n-1) crash* sucessivos de servidores na função de líder, durante a execução de um serviço.

Cálculo do tempo máximo de execução do serviço MR sob hipótese H_1

No sentido de simplificar a análise, neste ponto, assumimos somente a hipótese H_1. Consideramos que os pedidos de serviço e as respectivas respostas entre o processo cliente e o grupo de servidores se dão através de comunicações ponto-a-ponto e que os servidores se executam no modelo bloqueante o serviço solicitado. Após a recepção da mensagem de pedido, nós temos que o tempo de resposta máximo (t_resp_{MAX}) que o cliente deve esperar é dado por:

$$t\_resp_{MAX} = 2\,\delta + t\_serv_{MAX} \qquad (6.1)$$

onde: t_serv_{MAX}, corresponde ao tempo máximo envolvendo os processamentos e as comunicações de grupo no servidor MR, na execução de um pedido de serviço. 2δ é o tempo relacionado com as comunicações ponto-a-ponto entre cliente e líder.

O tempo máximo de serviço é dado por:

$$t\_serv_{MAX} = t\_proc_{MAX} + timeout$$

ou seja, o tempo de resposta do processamento na réplica mais lenta t_proc_{MAX} mais o *timeout* para a espera dos resultados de processamento das réplicas. O tempo de processamento máximo (t_proc_{MAX}) é calculado com base na hipótese H_1, ou seja, em um processamento são percorridos os k algoritmos alternativos de uma classe:

$$t\_proc_{MAX} = k\,(t_{te} + t_{alg} + t_{ac} + t_{rb}) + t_{vot}$$

sendo que t_{te}, t_{alg}, t_{ac}, t_{rb} e t_{vot} são, respectivamente, os tempo de execução do teste de escalonamento, do algoritmo de aplicação, do teste de aceitação, tempo de retrocesso (*backward recovery*) e tempo para processamento dos mecanismos de votação.

O cálculo do valor de *timeout* envolvido para a espera dos resultados das réplicas antes da votação, exige algumas considerações. As réplicas MR são ativadas em seus nós por *event-trigger*, isto é, pela chegada no nó da mensagem-pedido de serviço. O valor deste *timeout* está baseado em algumas premissas:

i) o processamento nos nós não sofre interferência de outros processamentos; assumimos que a carga nos nós se resumem praticamente aos códigos das réplicas MR;

ii) a mensagem pedido, uma vez presente no líder, chegará no pior caso nas outras réplicas em Δ, que é o limite máximo para a difusão causal; o processamento no líder é imediato, ou seja, uma vez presente a mensagem-pedido o processamento se inicia;

iii) a diferença entre as velocidades de processamento das réplicas deve ser considerada neste cálculo. No pior caso, a diferença entre os tempos de processamento corresponde ao tempo em que a réplica mais rápida tem de esperar pela mais lenta:

$$diff = t\_proc_{lent} - t\_proc_{rap}$$

t_proc_{lent} e t_proc_{rap} são os tempos de processamento da réplica mais lenta e mais rápida respectivamente. Se considerarmos que existe um limite máximo Φ sobre as velocidades relativas das réplicas [Dwork 88]:

$$\Phi = \frac{t\_proc_{rap}}{t\_proc_{lent}}$$

então: $diff = t\_proc_{lent}(1 - \Phi)$ ou $diff = t\_proc_{rap}(\Phi - 1)$

iv) o *timeout* tem que considerar a pior situação de envio dos resultados de processamento entre as réplicas: $n\Delta$; isto ocorre com a não simultaneidade da comunicação no suporte e o pior caso na difusão de cada resultado.

Nas condições (i), (ii), (iii) e (iv), o valor *timeout* que garante à réplica mais rápida a obtenção dos resultados de processamento, incluindo os enviados pela mais lenta, nas piores condições, é dado por:

$$timeout = t\_proc_{MAX}\ (1 - \Phi) + (n+1)\ \Delta \qquad (6.2)$$

Com isto, o tempo de serviço máximo e o tempo de resposta máximo na condição da hipótese H_1 se resumem a:

$$t\_serv_{MAX} = t\_proc_{MAX}(2 - \Phi) + (n+1)\ \Delta \qquad (6.3)$$

e

$$t\_resp_{MAX} = 2\ \delta + t\_proc_{MAX}(2 - \Phi) + (n+1)\ \Delta \qquad (6.4)$$

Cálculo do tempo máximo de resposta do serviço MR considerando as hipóteses H_1 e H_2

Além das considerações anteriores, em análise de pior caso, assumimos a ocorrência de n - 1

falhas de *crash* sucessivas de servidores no desempenho do papel de líder, durante um ciclo de processamento, antes do envio da resposta ao cliente (hipótese H_2). A detecção da falha de um líder se dá por meio de servidores seguidores quando da ausência da mensagem msg_{fim_cp}. O protocolo de diagnose e de substituição do líder é executado em conjunto nos seguidores corretos na ausência dessa mensagem (item 6.2.2). Os *crash* sucessivos de n - 1 líderes no mesmo ciclo de processamento determinam uma correção (t_recup_{MAX}) no tempo de serviço máximo (t_serv_{MAX}, equação 6.3):

$$t'serv_{MAX} = t\_serv_{MAX} + t\_recup_{MAX} \qquad (6.5)$$

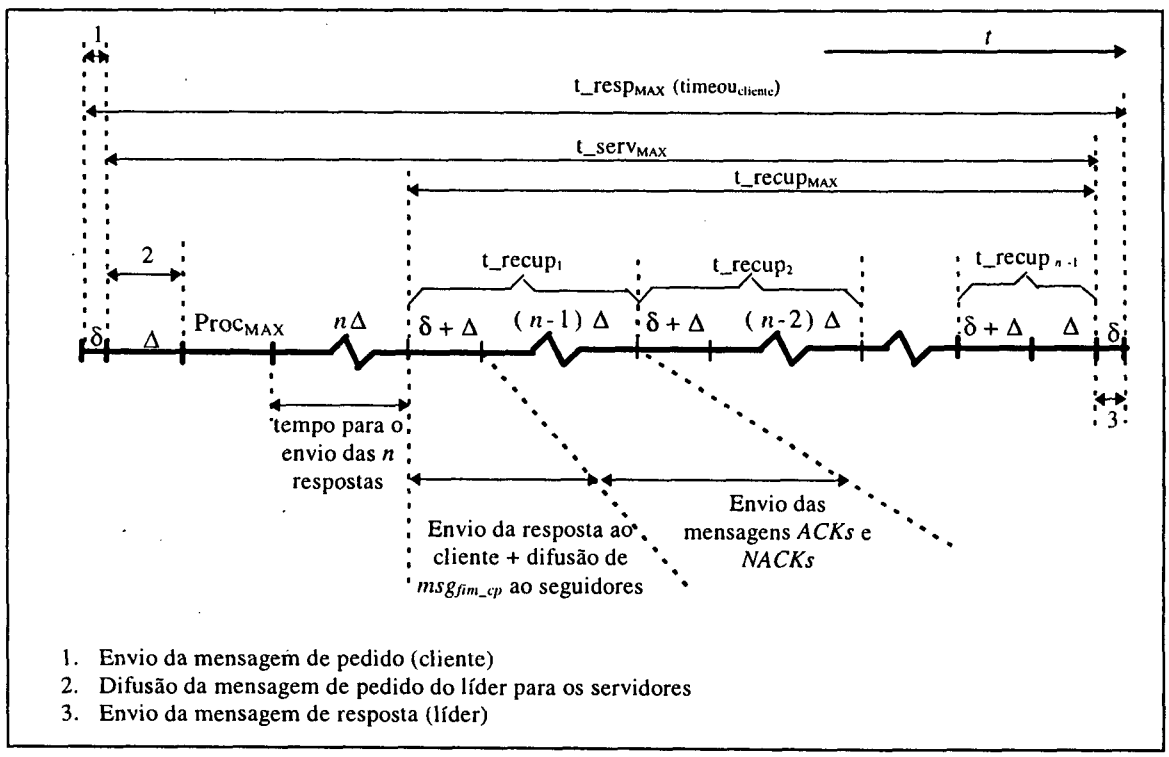


Figura 6.5: Diagrama temporal da execução de um serviço no esquema MR

O diagrama da figura 6.5 ilustra a hipótese H_2, onde um pedido submetido ao servidor replicado passa pelas várias etapas em situações de pior caso. Após o prazo para a recepção dos resultados ($n\Delta$) é armado o *timeout* de espera da mensagem de fim de ciclo ($\Delta + \delta$). O esgotamento deste o *timeout* (t_recup_1) inicia o protocolo de diagnose que no *crash* do primeiro líder tem a duração de (n-1) Δ. A armação seguinte de *timeout* de espera da ($\Delta + \delta$) deve detectar o *crash* do segundo líder (t_recup_2). Este processo se repete até que reste só uma réplica no MR (n-1 execuções do protocolo de diagnose, figura 6.5). Com isto, t_recup_{MAX} na ocorrência de n-1 *crash* de líderes é dado por:

$$t\_recup_{MAX} = (n-1)\left[\delta + \left(1 + \frac{n}{2}\right)\Delta\right] \quad (6.6)$$

Cálculo do *timeout* do processo cliente

Nos casos acima é assumido a não ocorrência de erros nas comunicações cliente/líder. De maneira a recuperar erros de omissão e de temporização que possam ocorrer nestas comunicações ponto-a-ponto, o processo cliente deve esperar após o envio do pedido durante um certo tempo ($timeout_{client}$) pela resposta do serviço. O cálculo deste *timeout*, considerando o pior caso, é dado pela expressão:

$$timeout_{cliente} = 2\,\delta + t\_serv_{MAX} + t\_recup_{MAX} \quad (6.7)$$

Após este tempo se o cliente não receber a mensagem de resposta o pedido de serviço pode ser reenviado ao servidor MR.

6.2.4 Implementações de servidores segundo a estrutura MR

O esquema MR, da mesma forma que os esquemas RB e PNV, explora o princípio da diversidade de projeto para o desenvolvimento dos algoritmos de aplicação. Desta forma, a partir de uma mesma especificação vários algoritmos de aplicação são desenvolvidos. Estes algoritmos são denominados de algoritmos genéricos de aplicação, pois podem ser executados para qualquer conjunto de dados de entrada.

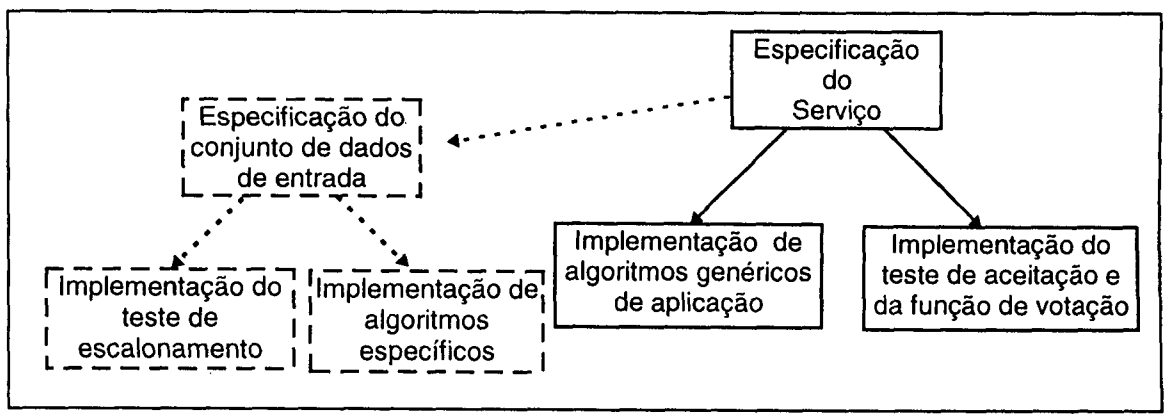


Figura 6.6: Etapas de desenvolvimento de um serviço MR

O esquema MR pressupõe que a especificação de um serviço possa se adequar a determinados tipos de entrada. A partir destas considerações, as especificações são usadas para

implementar algoritmos particulares que só são válidos para conjuntos específicos de dados de entradas o quê, de certa maneira, caracteriza o princípio da diversidade de dados. A estrutura de desenvolvimento de um serviço MR é mostrada na figura 6.6.

Na seqüência fazemos considerações sobre os suportes utilizados e as implementações dos mecanismos MR no modelo de execução descrito neste capítulo. A aplicação realizada em laboratório é apresentada a seguir, explorando várias etapas de seu desenvolvimento.

6.2.4.1 Programação dos mecanismos básicos do MR e suportes utilizados

O modelo de execução descrito do esquema MR foi implementado em um ambiente UNIX usando o *toolkit* ISIS [Birman 90]. O ISIS oferece um conjunto de facilidades para o gerenciamento e a comunicação de grupos. No ISIS o *grau de replicação* de um grupo é estabelecido no instante de criação, definindo o número mínimo de processos servidores para manter o serviço. Nesta implementação, o grau de replicação assumido para o servidor MR é de um processo.

O grupo de servidores MR tem um comportamento dinâmico, isto é, novos processos podem se associar ou deixar o grupo. As necessidades de gerenciamento para atender a esta dinâmica nos grupos é fornecida através de funções pelo ISIS. A consistência entre os integrantes do grupo é mantida através de funções do ISIS que permitem a um processo do grupo transferir o seu estado para um novo processo ou para um processo faltoso. Na implementação das comunicações de grupo é usado o protocolo CBCAST [Birman 90].

O ISIS define e mantém as semânticas de sincronismo virtual [Birman 94, Birman 87, Schiper 93]. O sincronismo virtual garante que todos os processos pertencentes a um grupo observam a todas as mudanças de configurações (associação de um novo processo ao grupo, retirada de um processo do grupo, etc.) no mesmo instante lógico. Além disto, todos os processos que fazem parte de uma mesma configuração recebem o mesmo conjunto de mensagens, independente se a ordenação é causal (CBCAST) ou total (ABCAST).

Implementação do esquema MR em processos UNIX

Os componentes que fazem parte do MR e não são relacionados com a aplicação são definidos a partir de uma biblioteca (libmr.h). Esta biblioteca é usada na programação de aplicações na forma de servidores MR, juntamente com as bibliotecas do ISIS (isis.h). Numa implementação, o programador além de definir os algoritmos alternativos de aplicação e o teste de aceitação deve, também, construir o teste de escalonamento a partir de uma função que identifica os tipos de dados de entrada.

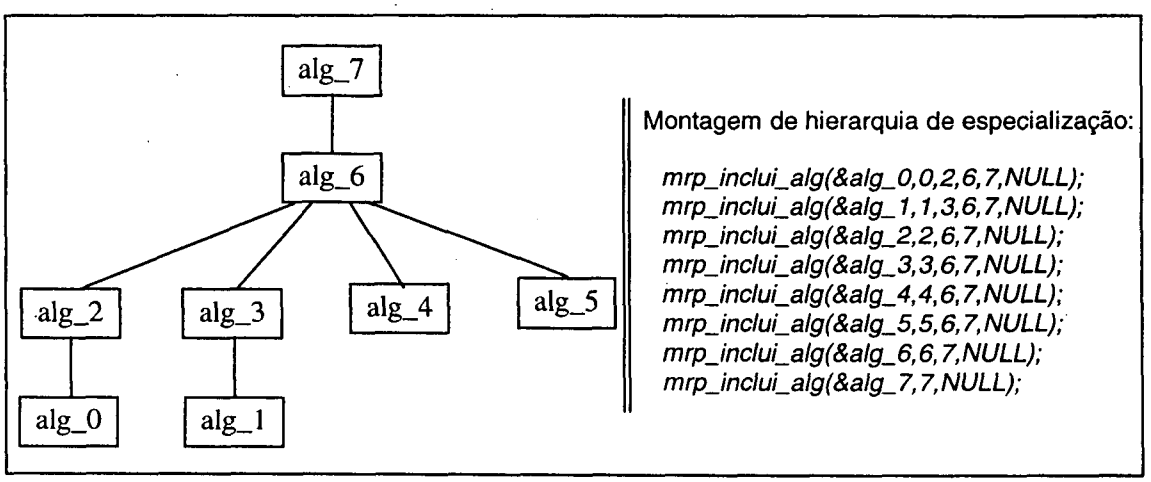


Figura 6.7: Exemplo de hierarquia de especialização

A função *mrp_inclui_alg(&alg, ident, lista_identificadores, NULL)* faz parte da biblioteca (libmr.h) e é responsável pela construção da **hierarquia de especialização**. A montagem da hierarquia de classes é feita inserindo os algoritmos de aplicação, um a um, usando a função *mrp_inclui_alg(...)*. Para facilitar a construção da hierarquia de especialização, a função *mrp_inclui_alg(...)* deve ser executada seqüencialmente do algoritmo mais especializado até o mais geral. O primeiro parâmetro desta função aponta para o algoritmo a ser inserido na hierarquia e o subseqüente indica a lista dos algoritmos que são alternativas para o mesmo. Obrigatoriamente, o último parâmetro da função *mrp_inclui_alg(...)* deve ser o símbolo NULL que indica o fim da lista de índices. A Figura 6.7 ilustra uma hierarquia de especialização composta de 8 algoritmos distribuído em 6 classes, a seqüência de chamadas a função *mrp_inclui_alg(...)* para a montagem desta hierarquia é também indicada na figura.

A biblioteca libmr.h também inclui as funções *mrp_func_teste_aceitação(...)*, *mrp_func_escalonamento(...)* e função *mrp_função_comparação(...)* que devem ser

executadas para definir, respectivamente, os procedimentos (funções) do teste de aceitação, do teste de escalonamento e da votação para cada processo MR. Os parâmetros das funções *mrp_func_teste_aceitação(...)*, *mrp_func_escalonamento (...)* e *mrp_função_comparação(...)* correspondem, respectivamente, aos ponteiros para as implementações do teste de escalonamento, do teste de aceitação e da votação. Caso o algoritmo de votação não seja redefinido pelo programador, o suporte assume o valor de uma função *default* que também faz parte da biblioteca (libmr.h). A figura 6.8 ilustra a estrutura básica de um processo (UNIX) MR.

```
#include <isis.h>
#include <libmr.h>

/* declarações de variáveis globais */
main(argc, argv) {
/* declarações de variáveis e funções */
      mrp_func_escalonamento (&esc);
      mrp_func_teste_aceitação (&tes_aceit);
      mrp_func_de_comparação (&votador);

mrp_inclui_alg (&alg, ident, lista de identificadores);
/* implementações das funções */
}
```

Figura 6.8: Estrutura de um processo (UNIX) MR

Função de votação

A função de votação faz parte da biblioteca libmr.h, sendo utilizada como *default* a comparação *bit-a-bit* de *N* valores. O programador pode redefinir a função de votação de maneira a não utilizar a default. A função de votação é executada tanto pelo servidor líder quanto pelos seguidores, após a recepção de todos os resultados enviados pelos outros servidores ou então, no final do *timeout* de sincronização (*timeout*). A figura 6.9 ilustra o procedimento correspondente a execução da votação.

Funções de gerenciamento de faltas

Também fazem parte da libmr.h, as funções de gerenciamento de faltas que implementam os protocolos de detecção e tratamento de faltas, descritos no item 6.2.2. A função de contabilização de falhas é executada por todos os processos servidores logo após a recepção da mensagem msg_{fim_cp}. Por outro lado, a função de diagnose de faltas do servidor líder só é

executada quando uma das mensagens $msg_{I'm\ alive}$ ou msg_{fim_cp} não é recebida por um dos seguidores.

```
#include <libmr.h>

servidor j: {
        :
    i=0;
    do {
      receive msg[i] to servidores;
      i++;
    } while ((i ≤ N) ∨ (timeout ≥ 0))
    valor_consenso = votação(msg , i);
} /* end servidor j */
```

Figura 6.9: Estrutura de um laço de votação

O uso do sistema ISIS como suporte de implementação pode trazer simplificações no modelo de execução do MR. Se, por exemplo, for usado as funcionalidades de *membership* deste sistema, o protocolo de diagnose de falhas de *crash* no servidor líder (item 6.2.2.2) se torna desnecessário. A definição de sincronismo virtual e as difusões de *View* usando GBCAST, garantem a detecção de *crash* do líder via o suporte ISIS sem a necessidade das trocas de mensagens especificadas no protocolo citado.

A contabilização de faltas e os passos especificados no algoritmo de detecção e tratamento de faltas do item 6.2.2.1 se mantém mesmo com o uso do suporte ISIS; isto porque, o modelo descrito neste capítulo envolve semânticas de faltas mais abrangentes (falhas por valor e *crash*) que as definidas no ISIS (*crash*).

6.2.4.2 Um servidor de multiplicação de matrizes

Uma de nossas experiências no desenvolvimento de uma aplicação distribuída a partir do esquema MR foi a implementação de um servidor que executa a multiplicação entre duas matrizes. Este serviço (multiplicação de matrizes) apresenta algumas características que facilitam a sua implementação a partir do esquema MR; entre estas características podemos citar a existência de vários algoritmos genéricos de multiplicação de matrizes apresentados na literatura, e também a existência de vários casos particulares que simplificam a multiplicação de matrizes.

O serviço de multiplicação de matrizes consiste de dois algoritmos genéricos, denominados de *gen_1* e *gen_2*. Além destes algoritmos genéricos, foram implementados algoritmos particulares que calculam o produto de matrizes somente para dados de entrada particulares:

- matrizes quadradas *(quad)*;
- uma matriz é diagonal *(diag)*;
- uma matriz é simétrica *(sim)*;
- uma matriz é identidade *(ident)* ;
- matrizes quadradas de ordem 2 *(ord2)*;
- matrizes quadradas de ordem 3 *(ord3)*.

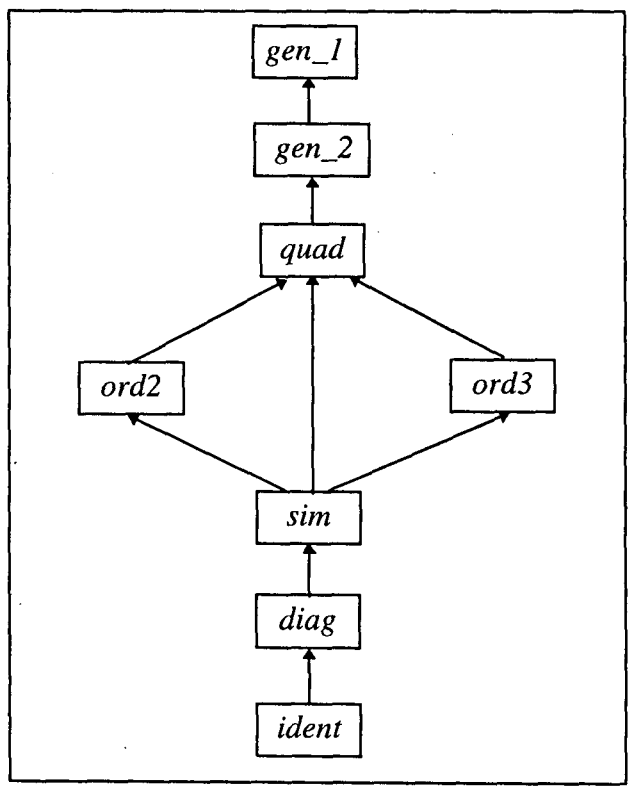


Figura 6.10: Exemplo de hierarquia de especialização

A figura 6.10 ilustra a hierarquia de especialização formada a partir destes algoritmos. Neste caso, o teste de escalonamento inicialmente verifica o tipo da matriz. Para cada tipo de matriz (dado de entrada) uma classe é selecionada. Por exemplo, se um dos dados de entrada é uma matriz identidade a classe de algoritmos selecionada começa com o algoritmo *ident*. Como toda matriz identidade também é uma matriz diagonal e simétrica, os algoritmos *diag* e *sim* podem ser executados quando da falha de execução do algoritmo *ident*. Os algoritmos

alternativos a execução de *sim* depende da ordem das matrizes: se as matrizes são quadradas o algoritmo alternativo é o *ord2*, se as matrizes são de ordem três a alternativa é o algoritmo *ord3* e se não são nem de ordem 2 nem de ordem 3 a alternativa é o algoritmo *quad*. O algoritmo *quad* é a alternativa para os algoritmos *ord2* e *ord3*. Os algoritmos *gen_1* e *gen_2* são alternativas a execução do algoritmo *quad*. O algoritmo raiz da hierarquia (*gen_1*) não possui alternativa.

Para esta aplicação específica, o teste de aceitação consiste em efetuar a operação inversa sobre o resultado. Ou seja, a matriz resultante do produto das matrizes é dividida por uma das matrizes de entrada e o resultado é comparado com a outra matriz de entrada. O algoritmo alternativo é executado no caso desta comparação não ser verdadeira. Se nenhum dos algoritmos da classe selecionada fornece um resultado que passe pelo teste de aceitação um resultado sinalizando a exceção é enviada ao cliente.

Configuração do sistema

O ambiente de execução consiste de uma rede local (Ethernet) composta por 6 estações de trabalho (*workstation*) *SparcStation*. As facilidades de gerenciamento e de comunicação de grupo são suportadas através do ISIS que executa acima do sistema operacional UNIX. A aplicação MR (servidores) é executada como processos ISIS. Para esta aplicação específica foram instanciados vários processos servidores de multiplicação de matrizes em diferentes estações do sistema.

Cálculo dos valores de timeout

Nesta implementação, na determinação dos valores empregados de *timeout* ou seja, o *timeout* de espera de resultados antes do voto ($timeout_{result}$), o timeout do cliente ($timeout_{cliente}$) e o *timeout* de espera pela mensagem de *"fim_de_ciclo_de_processamento"* ($timeout_{fim\_cp}$) foram usadas as equações (6.2) e (6.7) e $timeout_{fim\_cp} = \delta + \Delta$, respectivamente. Para uso destas equações foram considerados seis processos servidores um em cada estação, e a hierarquia de oito algoritmos descrita anteriormente. Os valores assumidos de δ e Δ foram, respectivamente, de 5 e 25,5 *mseg*. Neste caso, os algoritmos alternativos implementados no MR são simples com tempo de execução desprezível se comparados com as comunicações. Levando em conta todos estes fatores citados os valores de *timeout* obtidos a partir das

equações indicadas foram $timeout_{result}$=179 *ms*, $timeout_{cliente}$ = 670 *ms* e $timeout_{fim\_ciclo}$ = 56.4 *ms*. É importante verificar que o $timeout_{cliente}$ dá o tempo de resposta do serviço MR em pior caso. Os valores médios de tempo de resposta deste serviço são bem inferiores.

Ambiente de testes

Em tempo de execução vários cenários de testes foram criados no sentido de explorar as potencialidades do esquema MR. O primeiro cenário consiste na simulação de falhas *crash* nos servidores. Neste caso, os processos servidores são *mortos* através do comando *kill*. Se o comando *kill* mata o processo líder quando o grupo não está atendendo a nenhum pedido, a detecção da falha do líder é feita no instante em que um dos processos seguidores não recebe a mensagem *"I'm alive"*. A morte do processo líder faz com que um dos processos seguidores assuma o papel de líder depois de comprovada a falha. Por outro lado, quando o processo líder é morto durante um ciclo de processamento, o início do procedimento de diagnose de falha para a detecção da falha do líder é feita no momento em que um dos servidores não recebe a mensagem *"fim_de_ciclo_de_processamento"* que deveria ser enviada pelo líder. Neste caso, depois de comprovada a falha do líder, um dos seguidores assume o papel do líder é envia a mensagem de resposta ao cliente. A morte de processos seguidores, em qualquer uma das situações anteriores, faz com que o grupo seja reconfigurado sem a alteração do processo líder.

O segundo cenário de teste simula a ocorrência de falhas *crash* nas estações. Na impossibilidade de desligar e ligar uma estação de trabalho em qualquer instante, a simulação do *crash* de uma estação consiste em matar, também através do comando *kill*, o processo *isis* que é o responsável pelo suporte de tempo de execução do ISIS. Neste caso, embora a estação continue a operar normalmente o processo (MR) correspondente deixa de se comunicar. A detecção e o tratamento de falhas dos servidores (líder e seguidores) são executados pelos mesmos mecanismos descritos no cenário anterior.

O terceiro cenário é o mais complexo e simula a presença de faltas de software na função que implementa o teste de escalonamento. Esta falta é ativada aleatoriamente durante a execução do serviço. Quando da detecção de um erro gerado pela ativação deste tipo de falta, o contador que contabiliza o número de falhas alternadas é incrementado. Se for o caso, o

contador de falhas consecutivas também é incrementado. Quando um destes contadores atinge o limite imposto de 3 falhas consecutivas e/ou 5 falhas alternadas permitidas, o processo faltoso solicita a sua retirada do grupo.

Esta aplicação envolvendo um grupo de servidores de multiplicação de matrizes foi intensivamente testada, combinando a presença das falhas descritas nos cenários anteriores (falhas *crash* de servidores, falhas *crash* de estações e falhas de software do teste de escalonamento). Durante a execução dos serviços, novos processos e estações (através de processos ISIS) foram inseridos e retirados do sistema com o intuito de verificar o comportamento dinâmico da aplicação. Em todos os cenários de testes o comportamento da aplicação se mostrou adequado.

Ambiente de Depuração

Numa primeira versão do protótipo de servidores de matrizes, os passos de execução dos processos (líder e seguidores) foram acompanhados através de indicações apresentadas na tela (janela), onde os servidores são executados. Devido a complexidade da aplicação e a dificuldade de visualização dos passos de execução de cada processo foi desenvolvida uma interface gráfica, utilizando o sistema X-Windows, para a monitoração dos servidores do grupo. Esta interface gráfica permite visualizar *on-line* os processos que compõem o grupo, a fase do ciclo de processamento de cada servidor, o resultado fornecido por cada servidor, o estouro de timeout em um servidor, etc. Associado a esta interface existe ainda um histórico que permite ao usuário retornar a um determinado ciclo de processamento e em seguida acompanhar *off-line* passo a passo o processamento dos servidores. Durante a execução dos testes do MR, a existência deste mecanismo de histórico foi fundamental na identificação e correção de erros.

Embora esta interface facilite a depuração da aplicação, o seu uso é bastante limitado em operação normal devido a impossibilidade de controlar o escalonamento de janelas no ambiente X-Windows. Desta forma, quando o sistema é executado em modo de depuração, os timeouts dos processos servidores devem ser aumentados em relação aos utilizados no ambiente de execução normal. Este procedimento evita que os servidores seguidores executem freqüentemente os mecanismos de detecção de falha do processo líder sem necessidade. Uma outra restrição no uso desta interface, é que o número de eventos

manipulados pela interface é limitado. Em algumas situações, devido ao grande número de eventos gerados pela aplicação, pode ocorrer o *crash* do processo responsável pela interface gráfica. Neste caso, não é mais possível de acompanhar a execução dos processos através da interface, entretanto, a partir das indicações mostradas na janela (tela) é possível de verificar que a aplicação contínua a ser executada normalmente.

6.3 Outros modelos de execução para o MR

O esquema MR também pode ser implementado segundo um modelo de execução síncrono baseado na abordagem de Máquina de Estado [Schneider 90]. Os sistemas síncronos são utilizados, normalmente, no suporte de aplicações distribuídas com restrições temporais fortes (*hard real-time systems*). Nestes sistemas, os escalonamentos ou são estáticos e ativados pela passagem de tempo (*time-trigger*); ou são dinâmicos, com ativações *event-trigger* funcionando com mecanismos de prioridade fixa. O conhecimento a priori de uma carga estática e limitada garante o atendimento das restrições temporais na execução síncrona de uma aplicação.

O determinismo de réplicas nestes ambientes síncronos implica na necessidade de acordo e ordem temporal sobre as mensagens de entrada. A comunicação *bounded* é implementada com protocolos síncronos como os propostos em [Cristian 85], [Kopetz 94] e [Cristian 90] que garantem o *delivery* simultâneo das mensagens, ou seja, uma mensagem difundida no instante t é engajada no contexto dos participantes do grupo no instante $t + \Delta$. Modelos síncronos implicam na necessidade da existência de uma base de tempo global (relógios físicos sincronizados). Com o uso deste **tempo global**, os escalonamento nos processadores e no suporte de comunicação são executados dentro dos limites deterministas especificados.

Com base neste cenário síncrono, o esquema MR pode ser utilizado na implementação de aplicações distribuídas. Os clientes e servidores podem ser estruturados na forma de MRs. No caso o cliente pode ser construído, como no capítulo 4, por uma só classe de algoritmos. O escalonamento global pode ser estático, determinando uma escala de execução descrita a partir de um grafo de precedência que se executa ciclicamente.

Podemos ter também a execução do MR em um modelo assíncrono temporizado, na forma de réplicas ativas (Máquina de Estado [Schneider 90]), sem a existência de um servidor

privilegiado. A essência deste modelo é o atendimento das propriedades de acordo e ordenação das mensagens difundidas em cada *view* de *membership*. O protocolo assíncrono temporizado ABCAST, implementado no ISIS, garante estas propriedades uma vez que é fundamentado no conceito de sincronismo virtual [Birman 94].

6.4 Possibilidades do uso do esquema MR

Neste item nós descrevemos algumas aplicações e discutimos a viabilidade do uso do esquema MR na implementação de alguns serviços que compõem estas aplicações. Embora o MR possa ser utilizado na implementação de outros serviços, nós procuramos focalizar os serviços onde a flexibilidade fornecida no MR possa potencialmente oferecer alguma tipo de vantagem.

6.4.1 Sistema de navegação de aeronaves

No transporte aéreo civil, o esquema MR pode ser usado na implementação de alguns serviços no sistema de navegação. A figura 6.11 mostra a estrutura completa do Sistema de Gerenciamento do Veículo (VMS - *Vehicle Management System)* [Hynes 95], definido para o transporte aéreo civil de alta velocidade. O sistema VMS auxilia o piloto no planejamento, na navegação e no controle de vôo. A partir das informações recebidas do controlador de tráfego aéreo, o *translator* calcula a **trajetória de referência de vôo** (RFP - *Reference Flight Path*) que especifica completamente o plano de vôo (isto é, a altitude, a velocidade o ângulo de vôo, etc.) junto com as informações dos pontos de controle de *flap* e *gear* (trem de pouso), ponto de decida da aeronave e da distância ao próximo ponto de referência.

Um dos principais componentes do sistema VMS é o **módulo Sistema de Navegação** (figura 6.11) que, periodicamente, compara a trajetória atual do avião com a planejada e gera as operações que visam anular os erros de trajetória. Estas operações são geradas a partir de algoritmos que calculam os movimentos vertical e horizontal do avião. As operações geradas no módulo Sistema de Navegação são enviadas ao Controle de Vôo que altera a velocidade, a altitude e o ângulo de vôo da aeronave.

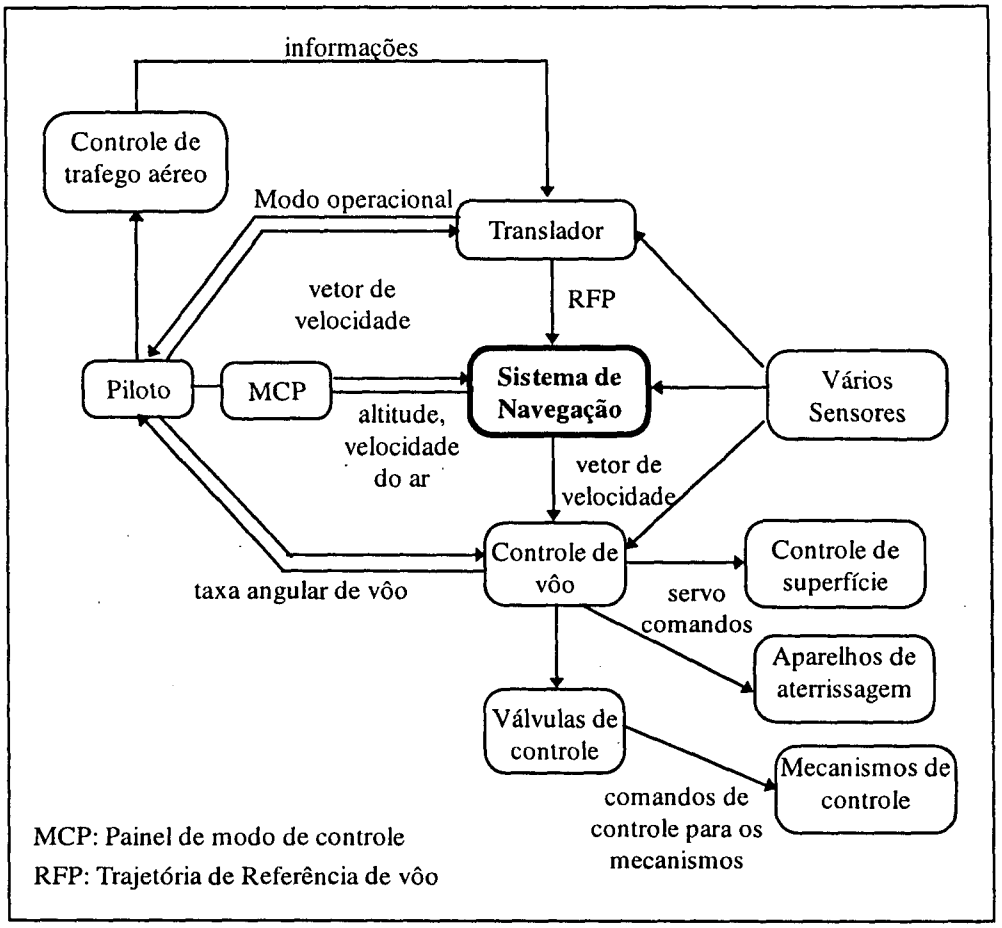


Figura 6.11: Sistema de gerenciamento de Veículos (Aeronaves)

Neste exemplo ilustrativo do uso do MR, nós nos restringimos na descrição da implementação do cálculo do movimento horizontal de aterrissagem do avião, como um servidor MR. O cálculo do movimento vertical pode ser feito usando a mesma sistemática. A descrição completa do sistema VMS pode ser encontrada em [Modugno 96, Hynes 95].

Durante o procedimento de aterrissagem o piloto pode especificar o tipo de aproximação desejada: menor consumo de combustível ou menor tempo de aterrissagem. O Sistema de Navegação deve calcular então as operações horizontais. Utilizando o esquema MR no Sistema de Navegação, nós podemos especificar então um servidor para o cálculo das operações horizontais com no mínimo três algoritmos: um genérico que calcula a trajetória sem nenhuma otimização, um que calcula a trajetória otimizando o tempo de aterrissagem e um terceiro que otimiza o consumo de combustível. Neste caso, a escolha do algoritmo preferencial em um ciclo de processamento é feita a partir do comando que define o tipo de aproximação desejada pelo piloto. Durante o procedimento de descida, dependendo da

seleção enviada pelo piloto, ora o algoritmo de otimização do tempo ora o de otimização de combustível atuam como principal.

Com essas considerações, é assumida a hierarquia de especialização formada pelos três algoritmos de cálculo de trajetória, composta por duas classes de três algoritmos; o algoritmo genérico nunca é o preferencial e sempre corresponde a última alternativa em ambas classes.

O calculo da trajetória vertical pelo servidor MR é feito em relação a posição atual. A posição atual é obtida com base em valores fornecidos por sensores (entradas externas). O determinismo de réplica, quando são envolvidas diferentes fontes emissoras (sensores), é mantido através de mecanismo que garanta o acordo e a mesma ordem dos eventos (capítulo 3).

É claro que por tratarmos com uma aplicação tempo real crítica, o modelo de execução do MR deve ser síncrono. E portanto, os valores dos diferentes sensores têm que ser difundidos entre as réplicas MR fazendo uso de uma comunicação de grupo *bounded.*

O teste de aceitação do Sistema de Navegação pode ser implementado a partir da comparação da altitude atual com a prevista pela execução, no módulo Sistema de Navegação, da alternativa corrente para o cálculo de trajetória horizontal. Se a calculada (prevista) for maior, o resultado do processamento da alternativa é rejeitado e, após a restauração do estado, a alternativa subsequente é processada.

A implementação deste serviço de direção de aeronaves ilustra bem a flexibilidade do esquema MR em relação a outros esquemas:

- a implementação deste serviço no esquema PNV envolveria o desenvolvimento de no mínimo seis algoritmos diferentes: três de otimização de tempo de vôo e três de otimização de combustível. Além disto, estes algoritmos devem ser estruturados como dois serviços distintos, com as mensagens, dependendo da seleção do piloto, sendo enviadas a cada conjunto de servidores PNV. Os resultados são gerados por algoritmos diferentes, de maneira que o resultado de consenso deve ser obtido através de uma votação inexata;

- A estrutura estática do esquema RB não permite a implementação deste serviço adequadamente. Para que o serviço satisfaça a esta especificação é necessário

definir dois blocos de recuperação, sendo que num o algoritmo principal é o de otimização de combustível e no outro é o algoritmo de otimização de tempo.

6.4.2 Algoritmos de ordenação

A ordenação (*sorting)* de dados em um *array* é um dos problemas fundamentais e ainda bastante discutido na área da ciência da computação. Normalmente, a ordenação simplifica a procura, facilitando assim a pesquisa de dados em um *array* (seqüência de entrada dos dados a serem ordenados).

Os algoritmos de ordenação podem ser divididos em duas classes: algoritmos clássicos (*bubble sort, quick sort, heap short, shell short,* etc.) e adaptativos [Estivill-Castro 92]. Os algoritmos de ordenação adaptativos simplificam o processo de ordenação explorando a ordem inicial dos dados no *array,* melhorando assim o desempenho na ordenação. Nesses algoritmos adaptativos o tempo de ordenação de um *array* é função do tamanho do *array* (número de itens) e da **desordem** da seqüência dos dados de entrada. A desordem de uma seqüência pode ser avaliada através de várias medidas; porém nesse trabalho nós nos limitamos a descrever apenas algumas destas medidas:

- inversões **(inv):** corresponde ao número de itens da seqüência que estão fora de ordem quando cada item é comparado com o item posterior;
- remoções **(rem):** indica o número mínimo de itens que devem ser retirados da seqüência original para se obter um *sub-array* ordenado;
- trocas **(exc):** número mínimo de trocas de posições de itens da seqüência original para se ordenar o array.

Por exemplo, para a seqüência W_1 = <6,2,4,7,3,1,10,5,8>, nós temos os seguintes valores: $inv(W_1) = 4$, $rem(W_1)=5$ e $exc(W_1) = 6$. Como pode ser observado, os valores dessas medidas para uma mesma sequência podem ser diferentes. Diferentes algoritmos são considerados ótimos para diferentes medidas [Estivill-Castro 92]:

- o desempenho do **algoritmo de inserção *stright*** (*stright)* é ótimo em relação a medida *inv*;

- o desempenho do **algoritmo Cksort** *(C Quicksort)* é ótimo em relação a *exc* e a *rem*;
- o desempenho do **algoritmo Csort** é ótimo em relação a *inv* e a *rem*.

Levando em consideração as medidas e os algoritmos adaptativos citados acima, podemos construir um servidor de ordenação que se executaria replicado. Na estruturação na forma de um servidor MR seriam consideradas essas medidas e algoritmos para as construções do teste de escalonamento e da hierarquia de especialização do MR.

A tabela 6.2 apresenta para as medidas *exc*, *inv* e *rem* as expressões que determinam os limites mínimos de desempenho nas ordenações de seqüências de entradas (X). Os algoritmos a serem usados na ordenações não poderão apresentar melhor desempenho que os valores calculados dos limites mínimos através dessas expressões e das seqüências de entrada consideradas.

Medidas de desordem	Limites inferiores
exc	$\Omega(\|X\| + exc(X) \log[exc(X)+1]))$
inv	$\Omega(\|X\| (1+\log[inv(X)/\|X\|)+1]))$
rem	$\Omega(\|X\| + rem(X) \log[rem(X)+1]))$

Tabela 6.2: Limites inferiores de desempenho para algoritmos de ordenação

Com os algoritmos de ordenação *stright, Cksort e Csort* podemos definir as classes de algoritmos que vão compor a hierarquia de especialização. O critério estabelecido para definição das classes e da ordenação dos algoritmos nessas classes é baseado nas medidas de desordem e nas expressões que determinam os limites mínimos de desempenho da tabela 6.2. As classes possíveis para os algoritmos considerados são:

classe 1:	*stright*	*Cksort*	*Csort*
classe 2:	*stright*	*Csort*	*Cksort*
classe 3:	*Cksort*	*stright*	*Csort*
classe 4:	*Cksort*	*Csort*	*stright*
classe 5:	*Csort*	*stright*	*Cksort*
classe 6:	*Csort*	*Cksort*	*stright*

As classes acima apresentam os algoritmos dispostos numa ordenação decrescente de desempenho; o algoritmo preferencial ocupa sempre a posição mais a esquerda.

O teste de escalonamento envolve a cada seqüência de entrada, calcular os valores das medidas (*exc*, *inv* e *rem*) e os respectivos limites de desempenho. A ordenação dos valores obtidos para esses limites de desempenho indica a classe selecionada na hierarquia. Considerando a seqüência (W_1) de exemplo citada acima, o teste de escalonamento além dos valores das medidas (inv = 4, rem =5 e exc = 6) forneceria os limites:

$$\Omega_{rem}(12,8907) \qquad \Omega_{inv}(13,0) \qquad \Omega_{exc}(14,0705)$$

a disposição dos algoritmos em cada classe, segundo o critério de ordenação decrescente de decrescente de desempenho, determinará que no exemplo a classe ativada pelo teste de escalonamento será a classe (*Csort*, *Cksort* , *stright*).

Este exemplo de aplicação ilustra bem a flexibilidade do esquema MR na seleção dinâmica de algoritmos. Enquanto nos esquemas baseados no RB os algoritmos alternativos são executados sempre na mesma sequência, no esquema MR a seqüência dos algoritmos executados em cada ciclo de processamento pode variar.

6.4.3 Programas paralelos

Em sistemas paralelos, onde parâmetros de desempenho mudam dinamicamente de maneira complexa e imprevisível, é difícil do compilador fazer previsões de valores ótimos para parâmetros do programa. A execução de um programa pode ser estática, dinâmica ou adaptativa em relação a determinadas condições (desempenho, correção, etc.) [Parker 96]. A execução é estática, se todos os parâmetros são determinados em tempo de compilação e os valores destes parâmetros não são alterados pelo programa em tempo de execução ou por algumas condições do sistema. A execução **dinâmica** significa que alguns parâmetros são determinados em tempo de execução, porém uma vez definidos não são mais alterados. Na execução **adaptativa** os parâmetros mudam a cada instante em conseqüência de alterações do sistema. A figura 6.12 ilustra a distinção entre os tipos de execução de um programa.

A estrutura de uma réplica no esquema MR é semelhante a um programa com execução dinâmica (figura 6.12 (b)), onde o **agente** faz o papel do teste de escalonamento. O esquema MR pode então ser utilizado para implementar serviços tolerante a faltas em um ambiente paralelo utilizando como algoritmos alternativos as versões parametrizados de um mesmo

programa. O teste de escalonamento selecionaria cada versão com base nos recursos disponíveis no momento da execução do serviço. A proposta apresentada em [Parker 96] não trata do problema de replicação.

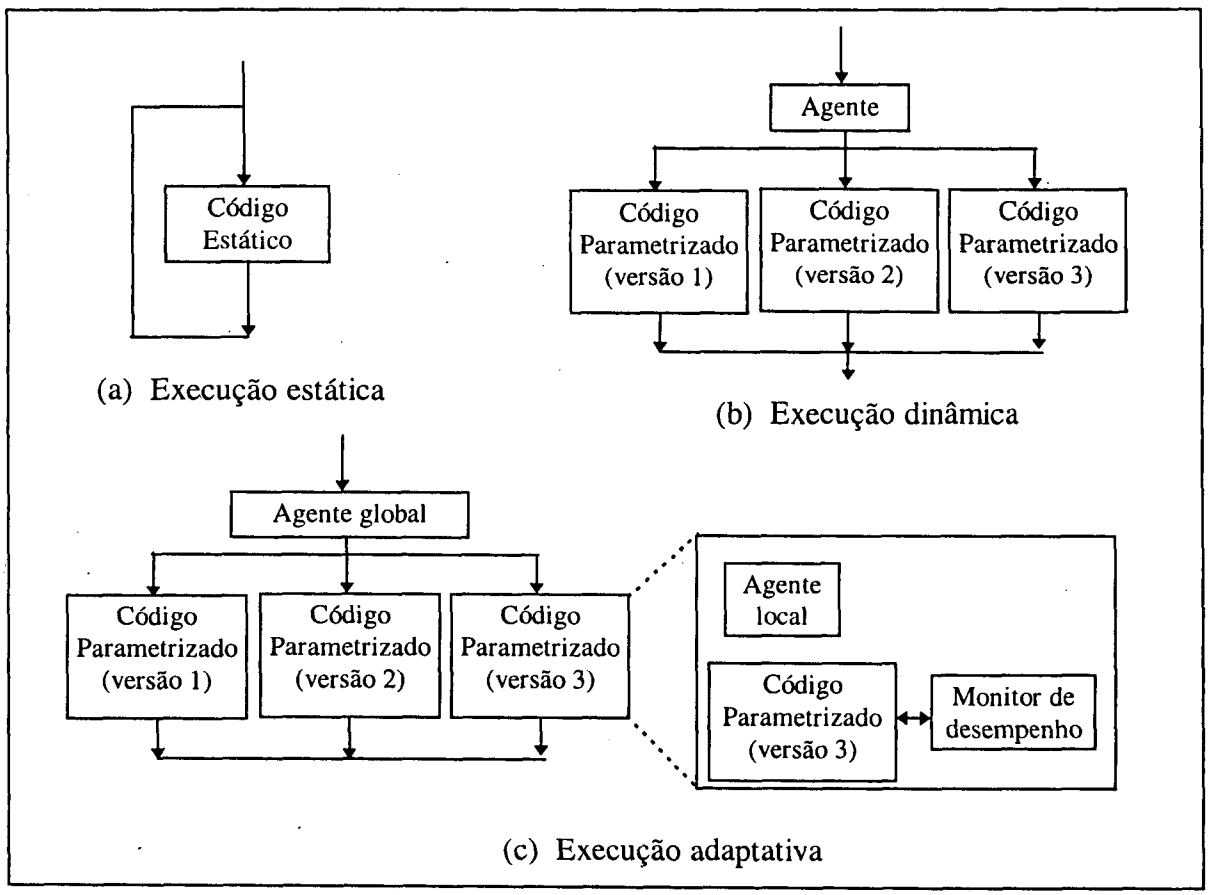


Figura 6.12: Execução estática, dinâmica e adaptativa

A princípio, o esquema MR também pode ser utilizado para implementação de serviços utilizando programas adaptáveis. Porém, esta implementação não é tão imediata e envolve o uso das técnicas e ferramentas apresentadas em [Parker 96].

É difícil de imaginar o uso dos esquemas PNV e RB na implementação deste mesmo serviço.

6.5 Conclusões

Neste capítulo foi apresentado um modelo de execução assíncrono temporizado para o MR, baseado na abordagem líder/seguidores. Em sistemas assíncronos temporizados os limites máximos entre as velocidades relativas de processamento (Φ) e das comunicações (Δ e δ) são

probabilistas. Estes limites são escolhidos conforme a margem de probabilidade de falhas de desempenho com que queremos o sistema funcionando.

Para o modelo introduzido foram descritos os algoritmos envolvidos com a detecção e o tratamento de faltas. A distinção entre as falhas *crash*, falhas por desempenho e falhas por valor está baseado em estatísticas. Um estudo de pior caso, assumindo os limites probabilistas, foi realizado. Este estudo nos permite dimensionar os valores de *timeouts* no modelo.

Este modelo de execução foi implementado na forma de um protótipo em nosso laboratório. Embora a aplicação usada fosse bastante simples, o seu desenvolvimento foi importante no equacionamento dos problemas relacionados com a implementação de serviços confiáveis a partir do MR; por exemplo, a definição dos aspectos de montagem da hierarquia de especialização, a definição do teste de aceitação, etc., foram concebidos depois de várias tentativas.

A partir da especificação desta aplicação é possível verificar uma das principais características do esquema MR: a flexibilidade. Em nenhum dos esquemas citados no capítulo 2 existe a seleção dinâmica de algoritmos, baseado em atributos dos dados de entrada; e muito menos que algoritmos diferentes façam parte de uma estrutura.

O protótipo desenvolvido foi intensamente testado no intuito de verificar a potencialidade do esquemas MR. Vários cenários de falhas foram simulados para demonstrar a capacidade do modelo de execução tolerar falhas *crash* de servidores, falhas *crash* de estações e falhas de software no teste de escalonamento e nos algoritmos de aplicação. Um ambiente de depuração utilizando os recursos gráficos do X-Windows foi desenvolvido exclusivamente para acompanhar *on-line* as etapas do processamento de um servidor MR. Nesta interface, também existe um mecanismo de histórico que permite o acompanhamento *off-line* do processamento dos servidores.

CAPÍTULO 7

Conclusões e Perspectivas Futuras

Uma visão geral do trabalho

Esta tese inicia com a descrição dos principais esquemas de tolerância a faltas propostos na literatura. Analisando o comportamento destes esquemas, nós observamos que a maioria apresenta uma estrutura fixa com escalonamento estático de versões (algoritmos) e número constante de componentes que fazem parte do processamento. Neste sentido, novos esquemas com características dinâmicas, adaptáveis à evolução de carga e de desempenho tão desejáveis a aplicações distribuídas estão surgindo. Soluções como o SCOP [Bondavalli 93] e as proposições em [Tai 94] de seleção dinâmica entre RBs e DRBs segundo as características de performabilidade do sistema, seguem essa tendência.

Em seguida foram apresentadas as principais técnicas e modelos de replicação utilizados em sistemas distribuídos. A discussão apresentada procurou enfatizar as propriedades dessas técnicas de replicação. A noção de grupo de processos é apresentada como uma forma natural de expressar os modelos de replicação em sistemas distribuídos. Normalmente, nesses modelos de replicação, somente o controle externo é enfatizado, ou seja, somente as interações entre as réplicas e a preocupação de manter a consistência dos estados. Os aspectos ligados às semânticas da aplicação são totalmente ignorados pelos modelos de replicação. Isto restringe o uso destes modelos a tolerância a faltas hardware. Para a tolerância a faltas de

valor ou de software seria necessário a extensão desses modelos através de mecanismos especiais, aproximando essas possíveis extensões dos esquemas usuais de tolerância a faltas descritos no capítulo 2.

Após esta discussão inicial, a proposta de um novo esquema de tolerância a faltas foi apresentada. Este esquema, denominado de Múltiplas Replicações (MR), apresenta uma estrutura mais flexível do que a dos esquemas usuais, permitindo assim que o esquema se adapte as condições evolutivas de uma aplicação distribuída. As múltiplas replicações no esquema MR são utilizadas no sentido de atender aos princípios da diversidade de projeto e da diversidade de dados.

A flexibilidade é alcançada no esquema MR introduzindo as noções de classe de algoritmos e hierarquia de especialização. O conjunto de algoritmos alternativos (base das replicações) que podem atender aos mesmos dados de entrada define uma classe de algoritmos. Numa mesma classe, os algoritmos estão dispostos de acordo com o grau de precisão com que processam os dados referentes a classe: de algoritmos mais especializados a algoritmos mais gerais. O conjunto total de classes gerado a partir desses algoritmos alternativos define uma hierarquia de especialização. Para um determinado dado de entrada, o teste de escalonamento é responsável pela seleção dinâmica da lista de algoritmos que fazem parte da classe de algoritmos associados ao dado de entrada. A replicação ativada deve executar o primeiro algoritmo da lista (classe), denominado preferencial. Os outros algoritmos da classe servem de alternativas ao preferencial; no caso da falha do algoritmo preferencial, a lista de algoritmos que compõem a classe é percorrida no sentido contrário a especialização, até que um dos algoritmos tenha seus resultados passando pelo teste de aceitação. Após a execução do teste de aceitação, as saídas geradas pelos algoritmos base das replicações são enviadas ao votador ou ajustador que é responsável por encontrar um único resultado de consenso.

Com base nos comandos da linguagem CSP, o esquema MR foi definido pela combinação de *alternate* de processos prefixos:

$$MR = (c^N ? m_1 \rightarrow P_1^N | c^N ? m_2 \rightarrow P_2^N | ... | c^N ? m_k \rightarrow P_k^N)$$

onde, P_1^N, P_2^N, ... P_k^N correspondem a k diferentes replicações que são ativadas no momento da recepção da mensagem de entrada m_i ($1 \leq i \leq k$), pelo teste de escalonamento.

Usando o modelo de traços CSP, verificamos a adequação do MR às provas de correção da Teoria de Processos Replicados de [Mancini 88]. Nesse sentido, o processo $MR=(c^N?m_1 \rightarrow P_1{}^N | c^N?m_2 \rightarrow P_2{}^N | \ldots | c^N?m_k \rightarrow P_k{}^N)$ é uma replicação do processo $(c?m_1 \rightarrow P_1 | c?m_2 \rightarrow P_2 | \ldots | c?m_k \rightarrow P_k)$, se a maioria das N réplicas do processo base é não faltosa (Teorema T_3).

O MR foi visto como componente elementar para a composição de Sistemas Distribuídos Replicados (SDR). Nesse caso, as comunicações entre processos MRs são feitas através de processos específicos, denominados de Sistemas de Comunicação (SC), introduzido no sentido de expressar as comunicações de grupo. O uso do MR como unidade básica de processamento distribuído é mostrado no texto através dos modelos de interação Cliente/Servidor e Grafos de Precedência (ou modelo *data-driven* [Mori 86]).

Uma análise com o objetivo de quantificar o desempenho e a Segurança de Funcionamento do esquema MR foi apresentada no capítulo 5. Essa análise utiliza como medida a performabilidade (*performability* [Meyer 80]). Esse estudo explorou faixas de faltas independentes e relacionadas e permitiu que se fizesse uma comparação quantitativa entre o MR e outros esquemas de tolerância a faltas presentes na literatura. Considerando os resultados obtidos nos diferentes estudos de casos, nós concluímos que embora as estruturas dos esquemas sejam semelhantes, a ordem com que os mecanismos de tolerância a faltas e os algoritmos alternativos de aplicação são executados determinam valores de performabilidade diferentes nos esquemas analisados. Constatamos que existe uma melhor performabilidade de esquemas que apresentem algum tipo de flexibilidade (SCOP, PNV-TB e o MR). A partir da análise de performabilidade, nós observamos que o comportamento do MR do ponto de vista do desempenho e da Segurança de Funcionamento é bastante satisfatório e atendeu as nossas expectativas, sendo que em média a performabilidade do esquema MR é melhor do que a apresentada pela maioria dos esquemas em todas as faixas de probabilidade analisadas.

Finalmente, nós discutimos e apresentamos a implementação do esquema MR em um sistema assíncrono temporizado [Cristian 96], segundo o modelo de replicação Líder/Seguidores [Powell 91]. O modelo de execução apresentado foi implementado em um ambiente UNIX, utilizando as facilidades de comunicação de grupo fornecidas pelo *toolkit* ISIS [Birman 90]. Uma aplicação envolvendo um servidor de matrizes confiável foi implementada com a

intenção de explorar uma das principais características do MR que é o escalonamento dinâmico de replicações. Para esta aplicação foram definidos procedimentos de testes e de depuração envolvendo vários cenários de faltas (*crash* de servidores, falhas de software no teste de escalonamento, etc.), onde foi possível verificar as potencialidades do esquema MR.

7.1 Contribuições e limitações da proposta

Principais contribuições

Dentro dos objetivos traçados para este trabalho e das atividades desenvolvidas durante este período, nós podemos enumerar os seguintes pontos de destaques desta tese:

⇒ proposta do esquema MR que apresenta estruturas flexíveis, permitindo a seleção dinâmica das replicações, segundo as propriedades dos dados de entrada;

⇒ uso da noção de grupo de processos como suporte à implementação de esquemas de tolerância a faltas; no caso o MR;

⇒ um estudo no sentido de mostrar que o modelo MR é plenamente adequado às provas de correção introduzidas no trabalho de [Mancini 88].

⇒ validação do esquema MR:

- análise de performabilidade do MR;
- implementação de um servidor estruturado na forma de um MR.

Limitações do trabalho

Na literatura existem alguns trabalhos abordando a flexibilização em modelos de replicação com base na variação do número de réplicas que fazem parte do processamento replicado de um serviço [Little 94][Cristian 94][Wang 95]. Normalmente, a variação do número de réplicas é controlada por funções de gerenciamento: **serviço de gerenciamento de réplicas** em [Little 94] e **gerenciador de disponibilidade** em [Cristian 94]. Essas funções de gerenciamento estabelecem uma configuração inicial com base na qualidade de serviço desejada. Durante o tempo de execução, a configuração inicial é alterada (variação do número de réplicas) em situações onde a qualidade desejada (disponibilidade, desempenho, etc.) não está sendo mais atendida. No presente trabalho, não tratamos com esse tipo de flexibilidade baseado na variação do número de réplicas para atender expectativas de disponibilidade ou de desempenho. A técnica empregada no MR pode, através do teste de escalonamento,

selecionar diferentes algoritmos alternativos conforme o não atendimento de requisitos de desempenho na evolução do processamento, que consiste na troca de algoritmos mais elaborados por algoritmos mais rápidos. Essa técnica então não altera o número de réplicas, mas sim a precisão dos resultados quando a qualidade desejada não é alcançada. De certa forma, podemos afirmar que ambas as técnicas são complementares.

7.2 Perspectivas futuras

Considerando o estado atual deste trabalho, nós discutimos a seguir algumas atividades que podem ser executadas em complemento a esta tese de doutorado.

Proposta de uma metodologia

A evolução natural deste trabalho consiste em definir uma metodologia de desenvolvimento de serviços confiáveis usando o esquema MR. Existem metodologias que já são consideradas clássicas de desenvolvimento de algoritmos alternativos a partir de uma mesma especificação [Lyu 91] e [Kelly 91]. Porém estas metodologias consideram sempre o mesmo conjunto de dados de entrada. Desta forma, os algoritmos gerais presentes na estrutura de um serviço MR podem ser implementados a partir destas metodologias. Os algoritmos customizados (especializados) para os tipos de dados de entrada devem ser desenvolvidos através de uma outra abordagem que considere não só a especificação do serviço, mas também a especificação dos dados de entrada.

A figura 7.1 ilustra as etapas envolvidas em uma proposta inicial de metodologia de desenvolvimento de serviços confiáveis através do MR. A partir da especificação de serviço, são gerados os algoritmos gerais e os testes de aceitação. Os algoritmos alternativos customizados (especializados) são gerados a partir da mesma especificação de serviço, porém considerando o conjunto de dados de entrada.

Na configuração do serviço são especificados os parâmetros de qualidade de serviço (QoS) desejado (por exemplo, performabilidade, disponibilidade, confiabilidade, etc.). Com base nestes parâmetros, a hierarquia de especialização é montada e o teste de escalonamento associado criado.

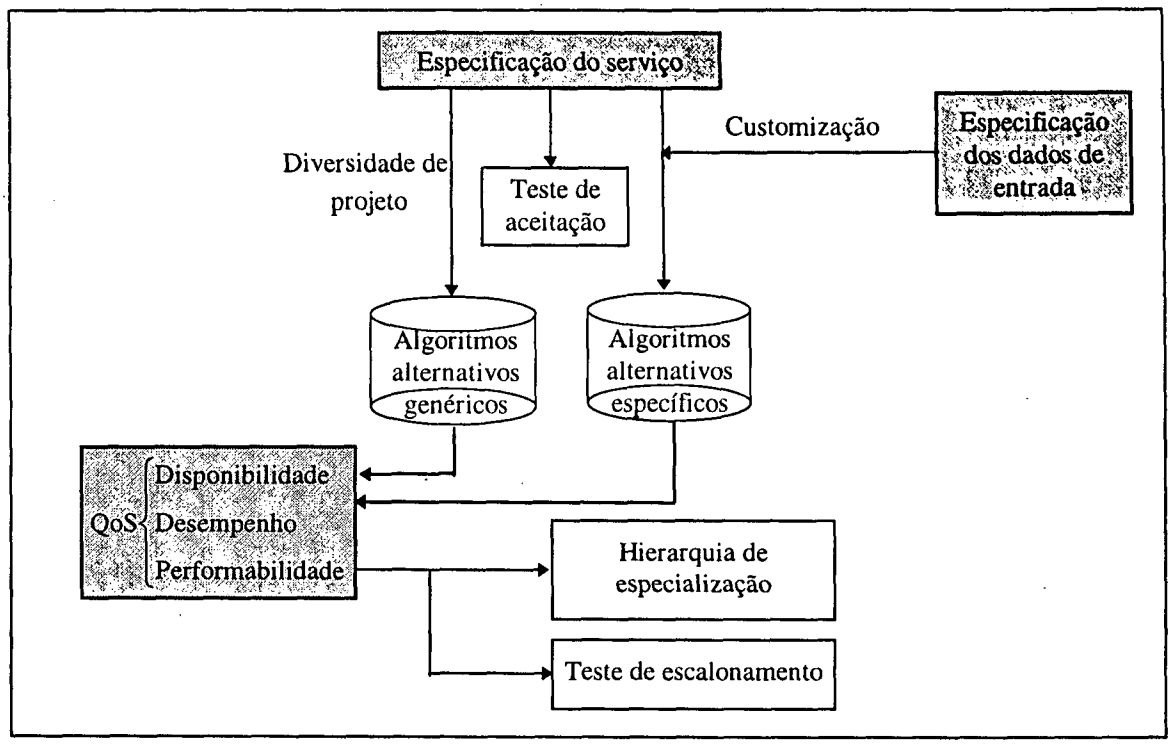


Figura 7.1 Etapas de desenvolvimento de aplicações estruturados segundo o MR

Incorporação do MR em estruturas de linguagem

A incorporação do MR em estruturas de linguagem no sentido de facilitar a programação dos mesmos é uma perspectivas interessante. Existem vários trabalhos na literatura que abordam o uso da Programação Orientada a Objetos e do conceito de Reflexão Computacional na implementação de aplicações tolerantes a faltas [Fabre 95],[Lung 96] e [Lisbôa 95]. Os princípios do Paradigma de Orientação a Objetos e da Reflexão Computacional são propícios para a programação dos mecanismos do MR. A abordagem Meta-Objetos [Maes 87] favorece a implementação do teste de escalonamento, teste de aceitação e vários aspectos do controle da execução das replicações a nível de meta.

SDR compostos a partir de MRs

No início, uma das principais motivações desse trabalho era trabalhar com aplicações distribuídas replicadas. Os ambientes alvos para essas aplicações eram redes de *transputers*. Continuamos acreditando na potencialidade do desenvolvimento de SDRs a partir do MR como unidade de estruturação.

Driven-performabillty framework

Uma perspectiva interessante de trabalho consiste em utilizar o esquema MR como um *framework* para a configuração de serviços confiáveis. O servidor MR sofreria então variações de configuração segundo medidas obtidas de performabilidade. O uso da performabilidade como medida na seleção de configurações (*driven-performability*) pode ter implicações no número de réplicas no MR, no número de algoritmos alternativos nas classes, etc.

Em contrapartida, as múltiplas vantagens atribuídas aos sistemas distribuídos que são cada vez mais utilizados atualmente, estão os cuidados necessários no sentido de garantir a sobrevivência de serviços e funcionalidades nesses sistemas diante de falhas parciais. Esperamos nessa tese ter contribuído com um instrumento eficiente na construção de serviços tolerantes a faltas.

ANEXO A

COMMUNICATION SEQUENCIAL PROCESSES (CSP)

O CSP (*Communicating Sequential processes*) foi introduzida em [Hoare 78] como a base para uma linguagem de programação concorrente. O CSP usa os comandos guardados de Dijkstra [Dijskstra 75] como estruturas de controle seqüencial e como o único meio de introduzir e controlar o não determinismo. O comando paralelo especifica a execução concorrente de comandos seqüenciais (processos). Todos os processos iniciam simultaneamente, e o comando paralelo termina satisfatoriamente somente quando todos os processos são executados corretamente. Formas simples de comandos de entrada e de saída são usados no CSP para expressar a comunicação entre processos concorrentes. Os processos CSP só podem se comunicar entre si através de comandos de entrada e de saída que correspondem a envio e a recepção de mensagens.

A descrição completa da linguagem CSP pode ser encontrada em [Hoare 85]. Neste texto nós nos limitamos a apresentar apenas um sumário dos principais comandos CSP utilizados no capítulo 4.

A.1. Processos

Um processo corresponde a um modelo do comportamento de um objeto descrito através de um conjuntos de eventos considerados relevantes. Este conjunto de eventos define o **alfabeto** do processo. Os principais comandos e conceitos relacionados a um processo CSP são:

- Processo atômico (*Atomic process*): A linguagem CSP apresenta os processos atômicos STOP que expressa o *deadlock* (bloqueio do processo, sem a execução de outras operações) e o SKIP que é um processo que não executa nenhuma operação, porém expressa a terminação correta;

- Operador prefixo ($x \rightarrow P$): descreve um objeto que primeiro executa o evento x e então comporta-se exatamente como o processo P;

- Escolha ($x \rightarrow P \mid y \rightarrow Q$): descreve um objeto que inicialmente engaja o evento x ou y. Depois da ocorrência do primeiro evento, o comportamento do objeto é descrito por P se o primeiro evento foi x ou por Q se o primeiro evento foi y;

- A expressão $(x{:}B \rightarrow P(x))$ define um processo que primeiro executa a escolhe de um evento y que faz parte do conjunto B e então comporta-se da mesma forma que $P(y)$;

- Recursão $(\mu X{:}A \,.\, F(X))$: a expressão $\mu X{:}A \,.\, F(X)$ descreve o comportamento de um processo recursivo, onde $F(X)$ é uma expressão guardada[1] que contém o processo X e A o alfabeto de X. Normalmente, é descrito de maneira simplificada, sem a representação do alfabeto A, pela expressão $\mu X{:}\, F(X)$;

- Paralelo ($P \parallel Q$): Se os alfabetos dos processos P e Q são iguais ($\alpha P = \alpha Q$), a notação $P \parallel Q$ representa o comportamento do processo que comporta-se da mesma forma que a composição dos processos P e Q interagindo em sincronização *lock-step*. Quando $\alpha P \neq \alpha Q$, os eventos que fazem parte do alfabeto de P, mas não de Q podem ocorrer independentemente de Q. Da mesma forma, Q somente pode engajar sozinho os eventos que estão em seu alfabeto, mas não estão do de P;

- Saída ($c \,!\, e$): O evento $c \,!\, e$ corresponde ao envio do valor e através do canal c. O significado de $c \,!\, e$ e o mesmo que $c \,.\, e$. A expressão $c \,!\, e \rightarrow P$ significa a envio do valor e pelo canal c e então o objeto se comporta exatamente como o processo P;

---

[1] A descrição de um processo que começa com um prefixo é denominada guardada.

- Entrada ($c ? e$): O evento $c ? e$ corresponde a recepção do valor e através do canal c. A expressão $c ? e \rightarrow P$ representa um objeto que primeiro recebe o evento e no canal c e depois se comporta da mesma forma que o processo P;

- Ocultação ($P \backslash A$): Seja P um processo e A um conjunto de canais; então $P \backslash A$ é um processo que tem o mesmo comportamento de P, porém as ações que ocorrem nos canais C são ocultadas; isto é, podem ocorrer sem sincronização;

- Expressão condicional ($P \lhd b \rhd Q$): se b uma expressão booleana, a expressão ($P \lhd b \rhd Q$) que comporta-se semelhante a P se b é verdadeira; caso contrário (b é falsa) o comportamento é semelhante;

A.2. Traços (*Traces*)

Um traço do comportamento de um processo é uma seqüência finita de símbolo que registra os eventos engajados pelo processo até um determinado instante. Um traço é representado por uma seqüência de símbolos que são separados por vírgulas colocados entre os caracteres < >:

<x, y> consiste de dois eventos, x seguido por y;
<x> é uma seqüência contendo somente o evento x;
<> é uma sequência vazia (não contém nenhum evento).

Os principais comandos que são executados sobre os traços de um processo e utilizados no capítulo 4 são:

- Concatenação ($s \hat{} t$): concatena o traço s com t;

- Restrição ($s \lceil A$): a expressão $s \lceil A$ resulta num traço formado por s com a retirada de todos os símbolos que não pertencem ao conjunto A;

- Head e tail (s_0 e s'): se s é uma seqüência não vazia, o primeiro termo é representado por s_0 e a seqüência resultante da retirada deste primeiro símbolo é denotada s';

- Star (A^*): é o conjunto de todos os traços finitos (incluindo <>) que é formado a partir dos símbolos do conjunto A;

- Ordenação ($s \leq t$): a relação $s \leq t$ é verdadeira se s é um prefixo de t, isto é, $t = s \hat{} w$ para qualquer traço w. $t < s$ com a condição de que $s \leq t$ e $s \neq t$.

- Length ($\# t$): corresponde ao tamanho (número de termos) de um traço.

APÊNDICE B

Modelos de Performabilidade de Esquemas de Tolerância a Faltas

Neste apêndice são apresentados os Modelos de Performabilidade de alguns dos esquemas de tolerância a faltas baseados principalmente na diversidade de projeto e que foram inicialmente descritos no Capítulo 2. A performabilidade de cada esquema é calculada a partir das expressões obtidas nos submodelos de Segurança de Funcionamento e de Desempenho. O estudo comparativo apresentado no Capítulo 4 foi feito a partir dos modelos descritos a seguir. Para cada esquema, são mantidos os principais tipos de faltas de software assumidos nos trabalhos de [Tai 93a], [Chiaradonna 94] e [Laprie 90], porém nós introduzimos em cada modelo a possibilidade de ocorrências de faltas de hardware. No caso do Bloco de Recuperação que da mesma forma que o MR faz uso de técnicas de recuperação em retrocesso do erro, é introduzido também no submodelo de Desempenho o tempo de restauração do estado inicial do bloco (tempo de *rollback*) quando da detecção de um erro.

A construção dos modelos de performabilidade de cada esquema tem por objetivo encontrar as expressões que permitem calcular:

- a probabilidade de falhas catastróficas - p_{fc};
- a probabilidade de falhas benignas de valor - p_{fbv};
- a função densidade de probabilidade da duração de uma iteração do esquema - $f_{sist}(y)$.

A probabilidade de faltas benignas por erro de temporização (p_{fbt}) e a performabilidade ($E[M_t]$) de cada esquema são obtidas através da substituição destas probabilidades nas equações 5.1 e 5.7 apresentadas no capítulo 5.

B.1 Esquema de Bloco de Recuperação (RB)

O esquema de Bloco de Recuperação (RB) considerado nesta modelagem é composto por duas alternativas (principal e secundária) e de um teste de aceitação (figura B.1 (a)). Inicialmente, o algoritmo principal (P) é executado e o resultado obtido submetido ao teste de aceitação (TA) que pode aceitar um resultado (saída 1), rejeitar um resultado correto ou incorreto (caminho interno) ou aceitar um resultado incorreto (saída 4). O algoritmo secundário (S) somente é executado quando o teste de aceitação rejeita o resultado gerado pelo algoritmo principal. Neste caso, ocorre o *rollback*, isto é, o restabelecimento do estado inicial do bloco para a execução do secundário. Se o resultado gerado pelo secundário não é aceito, independente do resultado ser correto ou não, o sistema ou envia um valor *default* ou não envia nenhum resultado (saída 3). A saída 5 é equivalente a saída 4, porém neste caso o resultado incorreto foi obtido pelo algoritmo secundário.

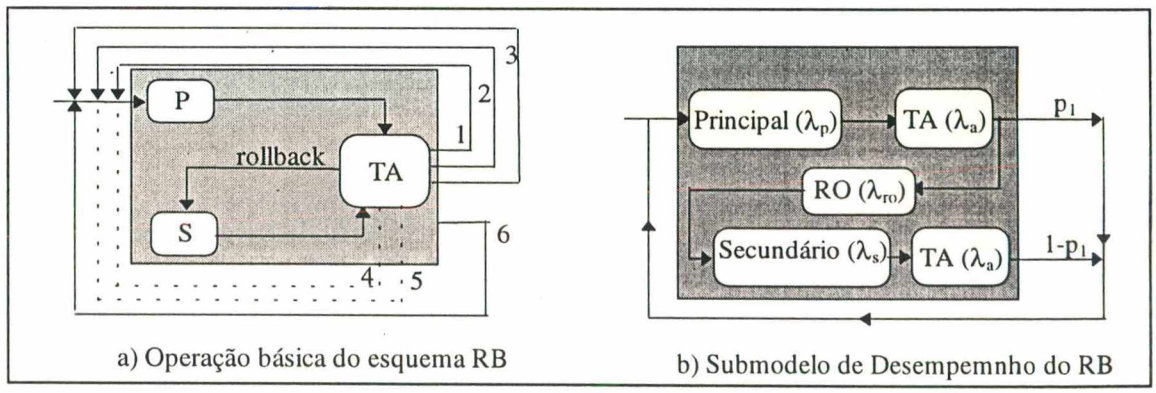
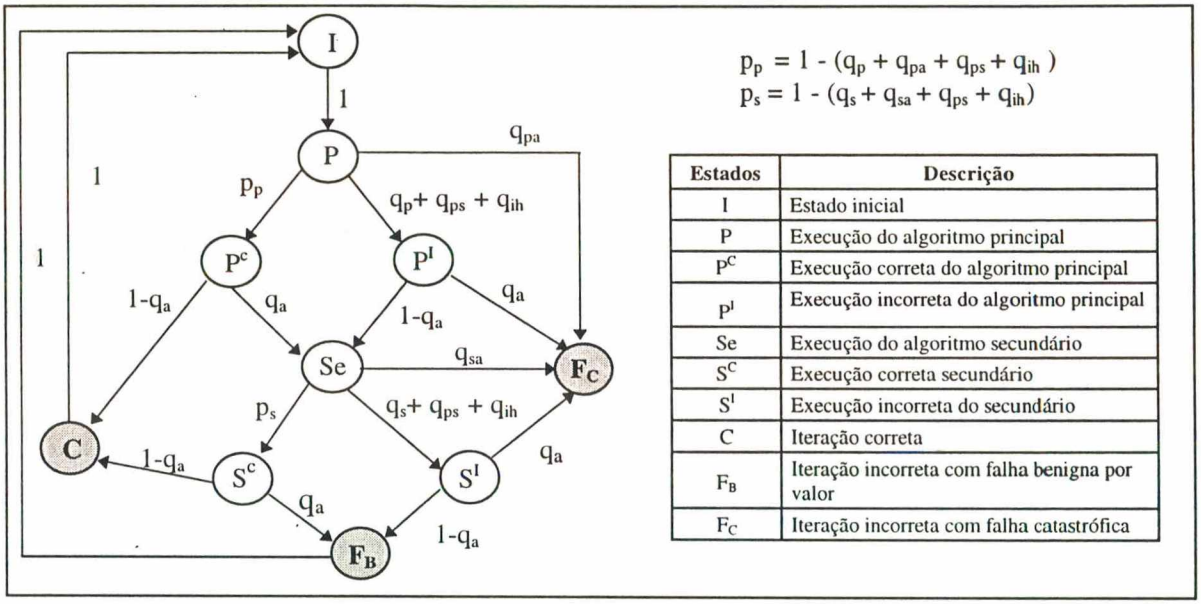


Figura B.1: Estrutura e submodelo de Desempenho do Esquema RB

Do ponto de vista da segurança de funcionamento, se o teste de aceitação aceita um resultado incorreto ocorre uma falha catastrófica. Esta condição é representada na figura B.1(a) através de linhas tracejadas (saídas 4 e 5). A saída 6 está relacionada ao término do deadline estabelecido a uma iteração; neste caso a iteração corrente é abortada e a próxima começa imediatamente. As saídas 3 e 6 estão relacionadas a ocorrências de falhas benignas por erro de valor e por temporização, respectivamente.

Tipos de faltas		Probabilidades
Falta(s) de software independente(s)	nos algoritmos alternativos *(principal* e secundário*)*	q_p , q_s
	no *TA* (resultado correto rejeitado ou resultado incorreto aceito)	q_a
Faltas de software relacionadas	entre os algoritmos alternativos *(principal e secundário)*	q_{b1b2}
	entre *pref e TA* e entre o secundário e o *TA*	q_{pa} , q_{sa}
Falta(s) de hardware independente	em um elemento de processamento	q_{ih}

Tabela B.1: Tipos de faltas e probabilidades associadas



Estados	Descrição
I	Estado inicial
P	Execução do algoritmo principal
P^C	Execução correta do algoritmo principal
P^I	Execução incorreta do algoritmo principal
Se	Execução do algoritmo secundário
S^C	Execução correta secundário
S^I	Execução incorreta do secundário
C	Iteração correta
F_B	Iteração incorreta com falha benigna por valor
F_C	Iteração incorreta com falha catastrófica

Figura B.2: submodelo de Segurança de Funcionamento do esquema RB

Os tipos de faltas considerados na modelagem do esquema RB são apresentados na tabela B.1. O submodelo de Segurança de Funcionamento do esquema RB (figura B.2) é construído a partir da descrição de operação anterior e dos tipos de faltas considerados na modelagem (tabela B.1). As transições entre os estados no submodelo de Segurança de Funcionamento estão relacionadas com as probabilidades de ocorrências das faltas descritas na tabela B.1. A partir do submodelo de Segurança de Funcionamento são obtidas as expressões que permitem calcular as probabilidades de falhas catastróficas (p_{fc}) e de falhas benignas por erro de valor (p_{fbv}) para o esquema RB:

$$p_{fc} = q_a(q_p + q_{ps} + q_{ih}) + q_{Se}\,(q_{sa} + q_a(q_s + q_{ps} + q_{ih})) + q_{pa}$$

$$p_{fbv} = q_{Se}\,(p_s\, q_a + (1-q_a)(q_s + q_{ps} + q_{ih}))$$

onde, q_{Se} é a probabilidade do algoritmo secundário ser executado, isto é:

$$q_{Se} = p_p\, q_a + (1-q_a)\,(q_p+q_{ps}+q_{ih})$$

O submodelo de Desempenho do esquema RB é mostrado na figura B.1(b). Os parâmetros λ_p, λ_s, λ_a e λ_{ro} são, respectivamente, as taxas de execução dos algoritmos principal e secundário, do teste de aceitação e de *rollback*. Neste trabalho, nós assumimos que o tempo de execução destes componentes são exponencialmente distribuídos e independentes. A função densidade de probabilidade (*fdp*) do tempo de execução (iteração) do esquema RB ($f_{sist}^{RB}(y)$) é obtida a partir do submodelo de Desempenho pela transformada inversa de Laplace de $F_{sist}^{RB}(s)$. A transformada $F_{sist}^{RB}(s)$ é calculada a partir das transformadas de Laplace das *fdps* do tempo de execução dos componentes do esquema, considerando que a iteração pode terminar após a execução do principal e do teste de aceitação ou então após as execuções das alternativas (principal e secundária), do *rollback* e teste de aceitação (executado duas vezes), ou seja:

$$F_{sist}^{RB}(s) = p_1\, F_p(s)\, F_a(s) + (1-p_1)\, F_p(s)\, F_{ro}(s)\, F_s(s)\, (F_a(s))^2$$

onde, $F_p(s)$, $F_s(s)$, $F_{ro}(s)$, $F_a(s)$ são as transformadas de Laplace das *fdps* do tempo de execução da alternativa principal, da alternativa secundária, de *rollback* e do teste de aceitação, respectivamente, e p_1 é a probabilidade da iteração terminar após a execução do principal e do teste de aceitação. A probabilidade p_1 obtida diretamente no submodelo de Segurança de Funcionamento é dada por:

$$p_1 = p_p\,(1-q_a) + (q_p + q_{ps} + q_{ih})\, q_a + q_{pa}$$

Uma vez calculada a função $f_{sist}^{RB}(y)$ (transformada inversa de $F_{sist}^{RB}(s)$) é possível se obter a probabilidade de falhas benignas por erro de temporização (p_{fbt}) através da equação 5.1, e a performabilidade do esquema RB usando a equação 5.7.

B.2 Esquema de Programação N-Versões (PNV)

A estrutura de um serviço implementado a partir do esquema PNV é mostrada na figura B.3(a). No esquema PNV, as três versões V_1, V_2 e V_3 são executadas paralelamente e os resultados enviados a um ajustador. Quando a maioria das versões são executadas corretamente, o resultado de consenso obtido pelo ajustador é correto (saída 1). O ajustador

ou envia um resultado *default* ou não envia nenhum quando não existe uma maioria (saída 2). Se existe uma maioria resultante de erros relacionados entre duas ou mais versões, o resultado de consenso obtido pelo ajustador é incorreto (saída 3). No caso, a saída 3 está relacionada com a ocorrência de falhas catastróficas. A saída 4 corresponde ao caso em que a duração de uma iteração excede ao deadline estabelecido; quando isto ocorre a execução do serviço corrente é abortada e a próxima iteração começa imediatamente. As saídas 2 e 4 estão relacionadas com a ocorrências de falhas benignas por erro de valor e de temporização, respectivamente.

Os tipos de faltas considerados na modelagem do esquema PNV e as probabilidades associadas são descritas na tabela B.2.

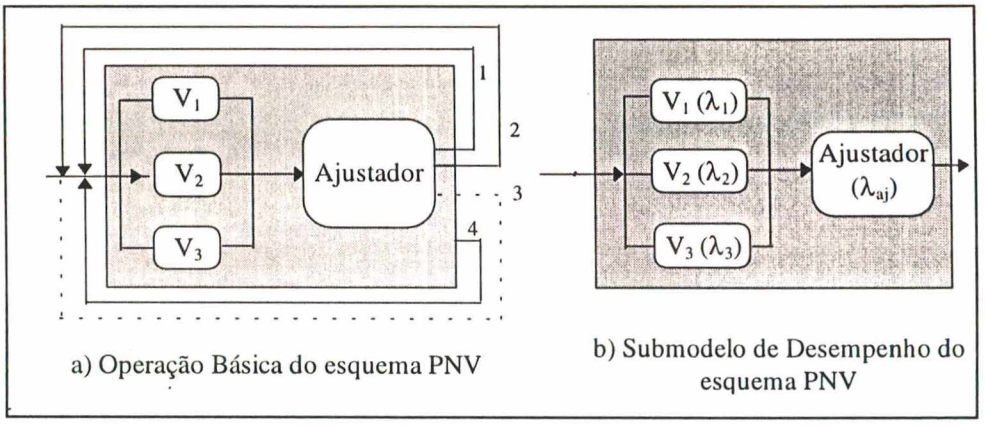


Figura B.3: Estrutura de submodelo de Desempenho esquema PNV

Tipos de faltas		Probabilidades
Falta(s) de software independente(s)	em uma versão	q_{iv}
	no ajustador (não reconhece maioria existente)	q_{d1}
	no ajustador (reconhece maioria inexistente)	q_{d2}
Faltas de software relacionadas	nas três versões	q_{3v}
	em duas versões	q_{2v}
Faltas de hardware independente	em um elemento de processamento	q_{ih}

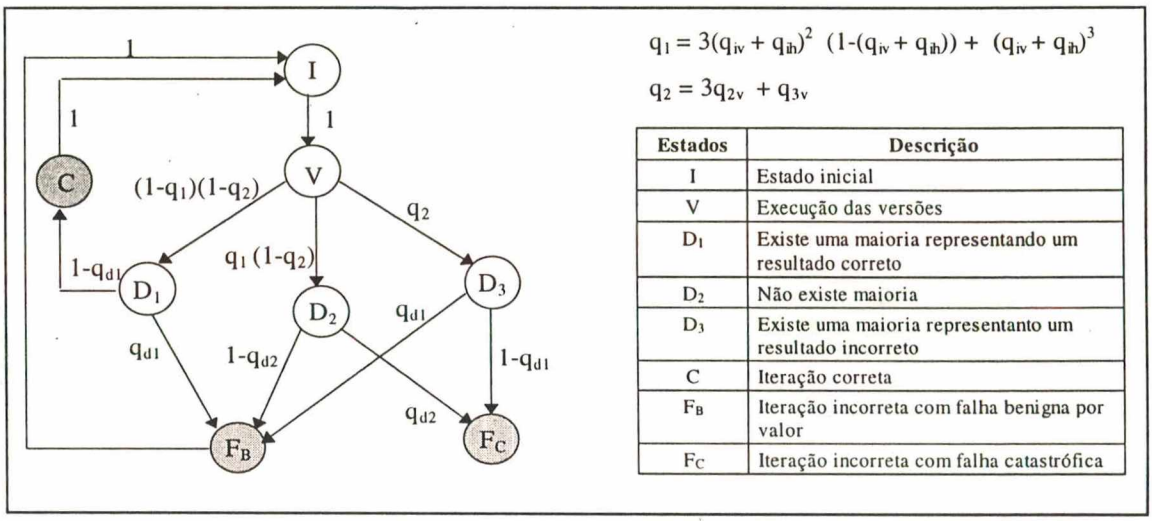
Tabela B.2: Tipos de faltas e probabilidades associadas

A figura B.4 apresenta o submodelo de Segurança de Funcionamento construído com base nos tipos de faltas (tabela B.2) e na descrição do esquema PNV. As probabilidades de faltas catastróficas (p_{fc}) e de falhas benignas com erros por valor (p_{fbv}) são calculadas a partir do submodelo de Segurança de Funcionamento:

$$p_{fc} = q_1 (1-q_2) q_{d2} + q_2 (1-q_{d1})$$

$$p_{fbv} = (1-q_1) (1-q_2) q_{d1} + q_1 (1-q_2)(1-q_{d2}) + q_2 q_{d1}$$

onde, a probabilidade q_1 corresponde a possibilidade de ocorrências de duas ou mais faltas independentes nas versões V_1, V_2 e V_3 ou nos nós de processamento em que estas versões são executadas e q_2 a probabilidade associadas com a ocorrência de duas ou mais faltas relacionadas entre as versões. Os valores das probabilidades q_1 e q_2 são calculados a partir das expressões apresentadas na Figura B.4.



Estados	Descrição
I	Estado inicial
V	Execução das versões
D_1	Existe uma maioria representando um resultado correto
D_2	Não existe maioria
D_3	Existe uma maioria representanto um resultado incorreto
C	Iteração correta
F_B	Iteração incorreta com falha benigna por valor
F_C	Iteração incorreta com falha catastrófica

Figura B.4: Submodelo de Segurança de Funcionamento para o Esquema PNV

O submodelo de Desempenho do esquema PNV é ilustrado na figura B.3 (b). O tempo de execução das versões V_1, V_2 e V_3 são exponencialmente distribuídos e independentes com taxas de execução λ_1, λ_2 e λ_3, respectivamente. Considerando que as variáveis aleatórias y_1, y_2 e y_3 estão associadas, respectivamente, aos tempos de execução das versões V_1, V_2 e V_3, o tempo máximo da execução em paralelo das três versões, representado pela variável y, é dada pela expressão $y = max\{y_1,y_2,y_3\}$, cuja função distribuição de probabilidade *(PDF)* eqüivale a:

$$G(y) = \begin{cases} (1-e^{-\lambda_1 y})(1-e^{-\lambda_2 y})(1-e^{-\lambda_3 y}) & \text{se } y \geq 0 \\ 0 & \text{se } y < 0 \end{cases}$$

A função densidade de probabilidade do tempo máximo de execução das três versões em paralelo corresponde a derivada da função $G(y)$, ou seja, $f(y) = G(y)/dy$. A transformada de Laplace do tempo de execução de um serviço PNV corresponde, então, a:

$$F_{sist}^{PNV}(s) = F(s)\, F_{aj}(s)$$

onde, $F_{aj}(s)$ é a transformada de Laplace da *fdp* do tempo de execução do ajustador e F(s) a transformada de Laplace da *fdp* de *f(y)*. A função densidade de probabilidade do tempo de execução do serviço $f_{sist}^{PNV}(y)$ corresponde a transformada inversa de Laplace de $F_{sist}^{PNV}(s)$. Da mesma forma que no esquema anterior, a probabilidade de faltas benignas por erro de temporização (p_{fbt}) e a performabilidade do esquema podem ser obtidas através das equações (5.1) e (5.7), respectivamente.

B.3 Esquema de Programação N-Versões com *Tie-Breaker* (PNV-TB)

O esquema PNV-TB é uma variante do esquema PNV que incorpora uma estratégia de sincronização no ajustador. A estrutura do esquema PNV-TB é mostrada na figura B.5 (a). Na implementação de um serviço com três versões, o ajustador tenta primeiramente obter o resultado de consenso a partir dos dois primeiros resultados gerados pelas versões. Se a tentativa de comparação é bem sucedida e os resultados são corretos um resultado de consenso correto é obtido (saída 1). Entretanto, se a comparação é bem sucedida, porém os resultados são incorretos em decorrência de erros relacionados presentes em duas ou mais versões, o resultado de consenso obtido é incorreto (saída 4). Quando a comparação dos dois primeiros resultados não é bem sucedida, o ajustador 2 é executado considerando o resultado fornecido pela terceira versão. Neste caso, a funcionalidade é equivalente ao do esquema PNV. Quando a maioria das versões são executadas corretamente, o resultado de consenso é correto (saída 2). Se não existe uma maioria, o resultado não é fornecido (saída 3). Caso exista uma maioria, porém esta maioria é resultante de erros relacionados entre duas ou mais versões, o resultado de consenso é incorreto (saída 5). As saídas 4 e 5 estão relacionadas com a ocorrência de falhas catastróficas. A saída 6 corresponde a situação em que a iteração corrente é abortada em virtude do término do deadline. As saídas 3 e 6 correspondem a ocorrências de falhas benignas por erro de valor e de temporização, respectivamente.

Os tipos de faltas considerados na modelagem do esquema PNV-TB são os mesmos do esquema PNV (tabela B.2). O submodelo de Segurança de Funcionamento é construído então a partir dos tipos de faltas apresentados na tabela B.2 e da funcionalidade do esquema PNV-TB (figura B.6). As probabilidades de falhas catastróficas (p_{fc}) e de falhas benignas por erro de valor (p_{fbv}) obtidas do submodelo de Segurança de Funcionamento (figura B.6) são:

$$p_{fc} = (q_{2v} + q_{3v})(1-q_{d1}^2) + 2q_{2v}[q_{d2} + (1-q_{d2})(1-q_{d1})] + q_{ih}^2(1-q_2)\,q_{d2}(1+(1-q_{d2})) + 2q_i(1-q_i)(1-q_2)\,q_{d2}(1+(1-q_{d2})\,q_i)$$

$$p_{fbv} = ((q_{2v} + q_{3v})\,q_{d1}^2 + 2q_{2v}(1-q_{d2})\,q_{d1} + q_i^2(1-q_2)(1-q_{d2})^2 + 2q_i(1-q_i)(1-q_2)(1-q_{d2})(q_i(1-q_{d2}) + q_{d1}(1-q_i)) + (1-q_1)(1-q_2)\,q_{d1}^2$$

onde, q_i soma das probabilidades de falhas independentes de hardware (q_{ih}) e software em uma versão (q_{iv}), ou seja, $q_i = q_{iv} + q_{ih}$.

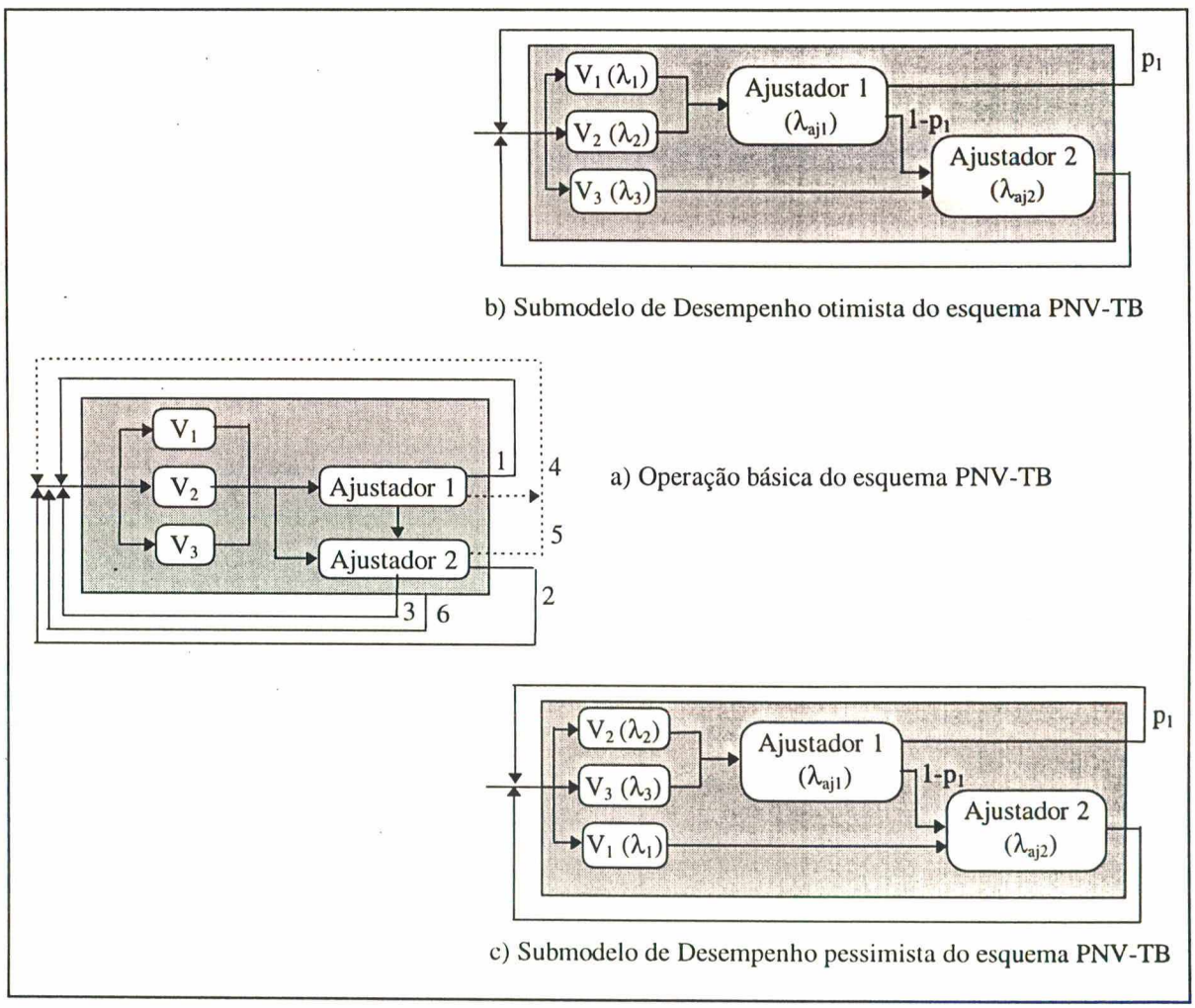
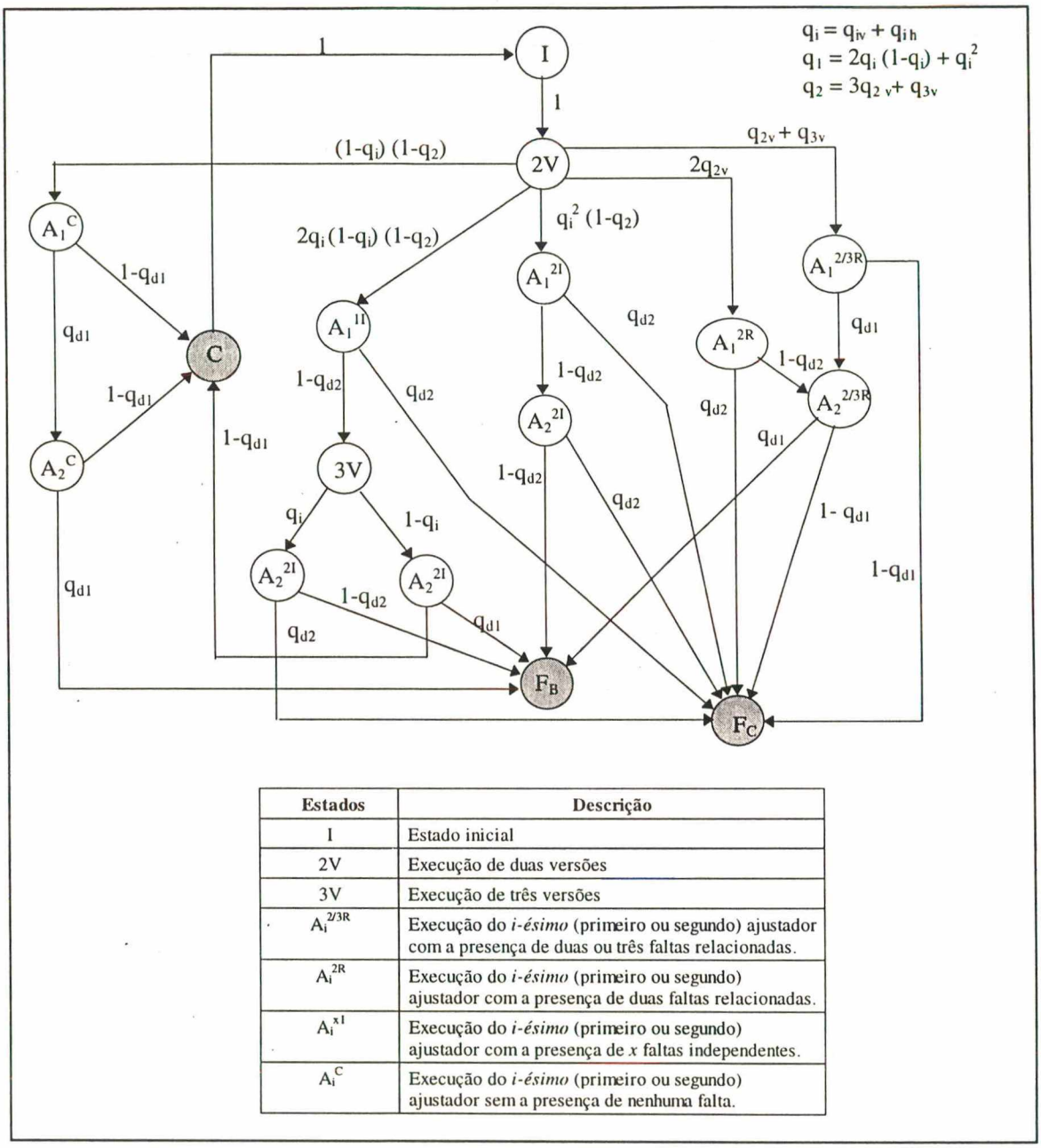


Figura B.5: Estrutura e submodelos de Desempenho do esquema PNV-TB



Estados	Descrição
I	Estado inicial
2V	Execução de duas versões
3V	Execução de três versões
$A_i^{2/3R}$	Execução do *i-ésimo* (primeiro ou segundo) ajustador com a presença de duas ou três faltas relacionadas.
A_i^{2R}	Execução do *i-ésimo* (primeiro ou segundo) ajustador com a presença de duas faltas relacionadas.
A_i^{xI}	Execução do *i-ésimo* (primeiro ou segundo) ajustador com a presença de *x* faltas independentes.
A_i^C	Execução do *i-ésimo* (primeiro ou segundo) ajustador sem a presença de nenhuma falta.

Figura B.6: Submodelo de Segurança de Funcionamento do PNV-TB

Como nos casos anteriores os tempos de execução das versões V_1, V_2 e V_3 apresentam uma distribuição exponencial e independente com taxas de execução de λ_1, λ_2 e λ_3, respectivamente, sendo que $\lambda_1 \leq \lambda_2 \leq \lambda_3$. No caso do esquema PNV-TB dois submodelos de desempenho são considerados: um otimista em que as duas versões com taxas de execução menores (λ_1 e λ_2) fornecem os resultados na primeira execução da função de ajuste (figura B.5 (b)) e outro pessimista (figura B.5 (c)) em que a primeira função de ajuste é executada com os resultados fornecidos pelas duas versões com taxas de execução maiores (λ_2 e λ_3):

- a função densidade de probabilidade do tempo de execução otimista ($f_{sist}^{PNV\text{-}TB(-)}(y)$) é calculada a partir da transformada inversa de Laplace da seguinte expressão :

$$F_{sist}^{PNV\text{-}TB(-)}(s) = p_1 F_{MAX\,(V1,V2)}(s) F_{aj1}(s) + (1-p_1) F_{MAX\,(V1/V2/aj1,\,V3)}(s) F_{aj2}(s)$$

- a função densidade de probabilidade do modelo de performabilidade pessimista ($f_{sist}^{PNV\text{-}TB(+)}(y)$) é calculada a partir da seguinte expressão:

$$F_{sist}^{PNV\text{-}TB(+)}(s) = p_1 F_{MAX\,(V2,V3)}(s) F_{aj1}(s) + (1-p_1) F_{MAX\,(V2/V3/aj1,\,V1)}(s) F_{aj2}(s)$$

onde, $F_{MAX\,(V1,V2)}(s)$ é a transformada de Laplace do tempo máximo de execução da versão V_1 combinada com a versão V_2 e $F_{MAX\,(V1/V2/aj1,\,V3)}(s)$ é a transformada de Laplace do tempo máximo de execução das execuções das versões V_1 e V_2 em série com o ajustador 1 combinada com a execução da versão V_3. As transformadas $F_{MAX\,(V2,V3)}(s)$ e $F_{MAX\,(V2/V3/aj1,\,V1)}(s)$ são equivalentes as anteriores, porém a versão V_3 e executada numa primeira etapa e V_1 na segunda. A *fdp* do tempo de execução das versões V_1 e V_2 é calculada através da derivada da seguinte função distribuição de probabilidade *(PDF)*:

$$G_{V_1V_2}(y) = \begin{cases} (1-e^{-\lambda_1 y})(1-e^{-\lambda_2 y}) & \text{se } y \geq 0 \\ 0 & \text{se } y < 0 \end{cases}$$

ou seja, $f_{V1V2}(y) = dG_{V1V2}(y)/dy$ e $F_{MAX\,(V1,V2)}(s)$ é a transformada de Laplace de $f_{V1V2}(y)$.

A transformada de Laplace $F_{MAX\,(V1/V2/aj1,\,V3)}(s)$ é calculada da mesma maneira, porém a função distribuição de probabilidade *(PDF)* é dada pela seguinte expressão:

$$G(y) = \begin{cases} \left(\int_0^y f_{v1v2aj}(y)\,dy \right) (1-e^{-\lambda_3 y}) & \text{se } y \geq 0 \\ 0 & \text{se } y < 0 \end{cases}$$

desta forma, $f_{V1V2aj}(y)$ é a transformada inversa de Laplace do produto $F_{MAX(V1,V2)}(s)\, F_{aj1}(s)$.

O parâmetro p_1 corresponde a probabilidade de uma iteração do esquema PNV-TB terminar após a execução do primeiro ajustador, sendo calculado a partir do submodelo de Segurança de Funcionamento:

$$p_1 = (1-q_1)(1-q_2)(1-q_{d1}) + 2q_i(1-q_i)(1-q_2)q_{d2} + q_i^2(1-q_2)q_{d2} + 2q_{2v}(1-q_{d2}) + (q_{2v}+q_{3v})(1-q_{d1})$$

B.4 Esquema de Programação N-Versões com Teste de Aceitação (PNV-TA)

O esquema PNV-TA é uma variante do PNV que submete os resultados de consenso obtidos pelo ajustador a um teste de aceitação. A execução do teste de aceitação tem o objetivo de reduzir a ocorrência de falhas catastróficas decorrentes de erros relacionados em duas ou mais versões. A estrutura do esquema PNV-TA é mostrada na figura B.7. Da mesma forma que no esquema PNV as três versões são executadas paralelamente e os resultados enviados ao ajustador. O ajustador ou envia um resultado *default* ou não envia nenhum resultado quando não existe uma maioria (saída 1). Quando existe uma maioria (correta ou incorreta) o resultado de consenso é enviado ao teste de aceitação que pode aceitar um resultado (saída 2), rejeitar um resultado correto ou incorreto (saída 3) ou aceitar um resultado de consenso incorreto (saída 4). Como nos casos anteriores, a saída 4 está relacionada com a ocorrência de falhas catastróficas. A saída 5 corresponde ao término do deadline estabelecido para uma iteração; neste caso a iteração corrente é abortada e a próxima começa no mesmo instante.

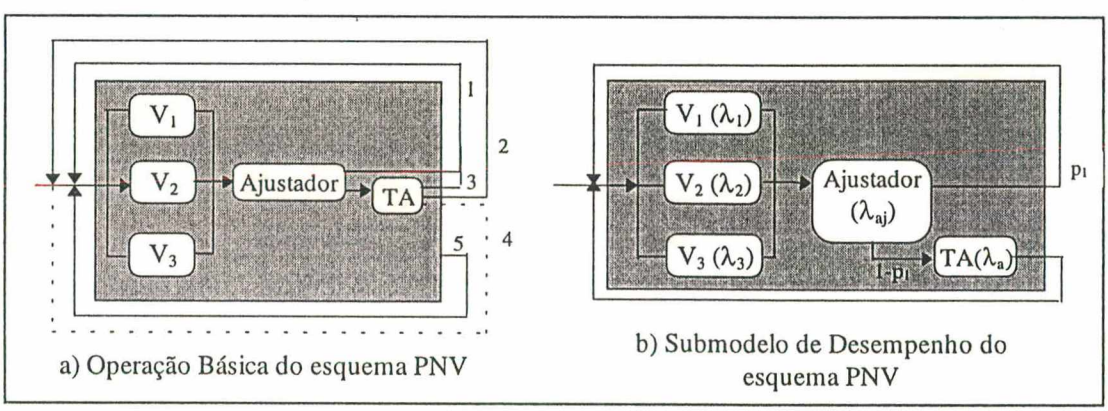


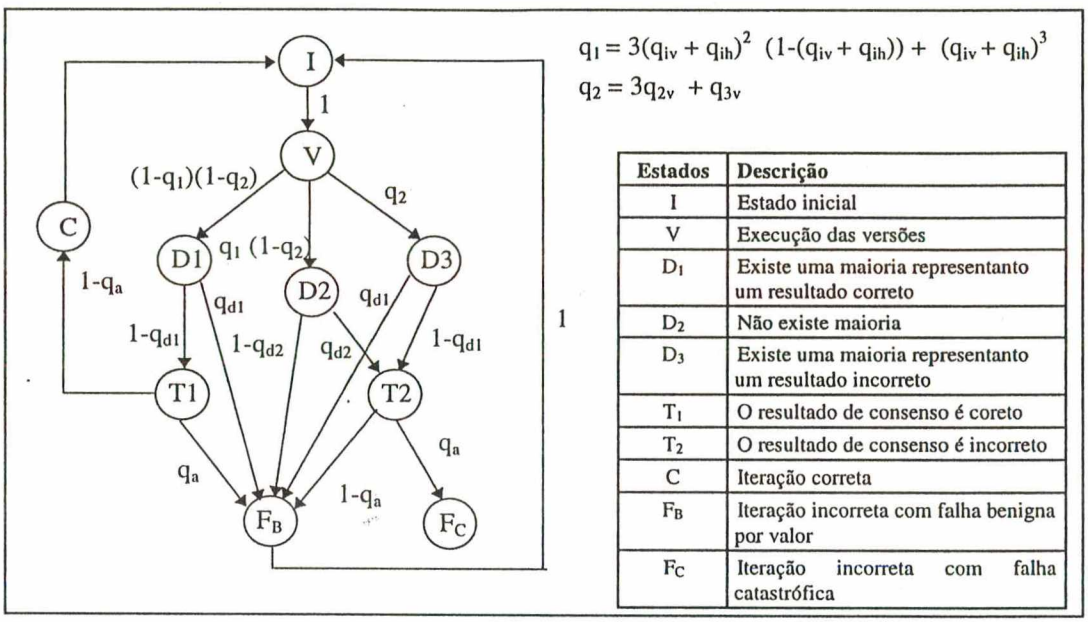
Figura B.7: Estrutura e submodelo de Desempenho do Esquema PNV-TA

Os tipos de faltas considerados na modelagem do esquema PNV-TA são os mesmos do PNV (tabela B.2) com a adição dos tipos de faltas que podem ocorrer no teste de aceitação (tabela B.1). O submodelo de Segurança de Funcionamento do esquema PNV-TA é mostrado na

figura B.8. As probabilidades de falhas catastróficas (p_{fc}) e de falhas benignas por erro de valor (p_{fbv}) para o esquema PNV-TA são calculadas pelas seguintes expressões:

$$p_{fc} = [q_2 (1-q_{d1}) + q_1 (1-q_2) q_{d2}] q_a$$

$$p_{fbv} = (1-q_1)(1-q_2) [(1-q_{d1}) q_a + q_{d1}] + q_1(1-q_2) [(1-q_{d2}) + q_{d2} (1-q_a)] + q_2 [q_{d1} + (1-q_{d1})(1-q_a)]$$



Estados	Descrição
I	Estado inicial
V	Execução das versões
D_1	Existe uma maioria representanto um resultado correto
D_2	Não existe maioria
D_3	Existe uma maioria representanto um resultado incorreto
T_1	O resultado de consenso é coreto
T_2	O resultado de consenso é incorreto
C	Iteração correta
F_B	Iteração incorreta com falha benigna por valor
F_C	Iteração incorreta com falha catastrófica

Figura B.8: Submodelo de Segurança de Funcionamento do esquema PNV-TA

O submodelo de Desempenho do esquema PNV-TA corresponde ao submodelo de Desempenho do esquema PNV com a execução em alguns casos do teste de aceitação (figura B.7(b)). Desta forma, a transformada de Laplace da função densidade de probabilidade do esquema PNV-TA é dada por:

$$F_{sist}^{PNV\text{-}TA}(s) = p_1 F_{sist}^{PNV}(s) + (1-p_1) F_{sist}^{PNV}(s) F_a(s)$$

onde, $F_{sist}^{PNV}(s)$ e $F_a(s)$ correspondem a transformada de Laplace da *fdp* do esquema PNV e do teste de aceitação, respectivamente. O valor de p_1 é a probabilidade do ajustador não obter uma maioria; neste caso, o teste de aceitação não é executado. O valor p_1 obtido do submodelo de Segurança de Funcionamento é dado pela expressão:

$$p_1 = (1-q_1)(1-q_2) q_{d1} + q_1(1-q_2) (1-q_{d2}) + q_2 q_{d1}$$

A *fdp* do tempo de execução do esquema PNV-TA, representada por $f_{sist}^{PNV\text{-}TA}$, corresponde a transformada inversa de Laplace de $F_{sist}^{PNV\text{-}TA}(s)$.

Com os valores de p_{fbv} e p_{fc} e da *fdp* $f_{sist}^{PNV\text{-}TA}$, a probabilidade de faltas benignas por erro de temporização (p_{fbt}) e a performabilidade do esquema PNV-TA podem ser calculadas através das equações (5.1) e (5.7), respectivamente.

B.5 Esquema de Bloco de Recuperação em Consenso (CRB)

A funcionalidade do esquema CRB é semelhante a do esquema PNV-TA com exceção de que o teste de aceitação somente é executado quando o resultado de consenso não pode ser obtido devido a não existência de uma maioria. A figura B.9 ilustra a estrutura do esquema CRB com três versões. Da mesma forma que no esquema PNV, as três versões (V_1, V_2 e V_3) são executadas paralelamente e os resultados obtidos são enviados ao ajustador. Quando a maioria das versões são executadas corretamente, o resultado de consenso obtido pelo ajustador é correto (saída 1). Por outro lado, pode ocorrer o consenso sobre um valor incorreto (saída 2). Quando o ajustador não é capaz de obter uma maioria, os resultados são enviados ao teste de aceitação que verifica cada resultado individualmente. Na execução do teste de aceitação, nós assumimos que a V_1 corresponde a alternativa principal, a versão V_2 a segunda alternativa e V_3 a terceira alternativa. O teste de aceitação (TA) pode aceitar um resultado correto (saída 3), rejeitar um resultado correto ou incorreto (saída 4) ou aceitar um resultado incorreto (saída 5). As saída 2 e 5 estão relacionadas a ocorrências de falhas catastróficas. A saída 6 está relacionada com o término do deadline estabelecido para uma iteração; neste caso a iteração corrente é abortada e a próxima começa imediatamente.

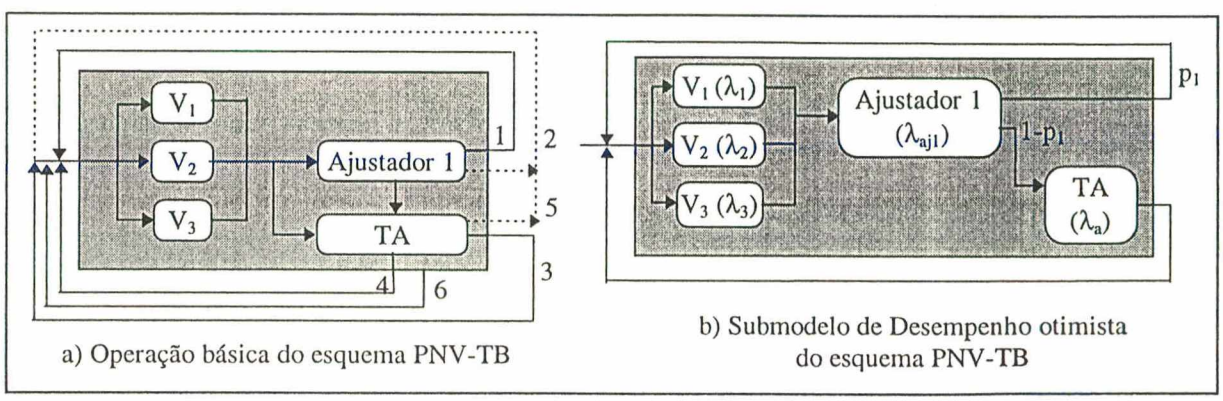


Figura B.9: Estrutura e submodelos de Desempenho do Esquema CRB

Tipos de faltas		Probabilidades
Falta(s) de software independente(s)	em uma versão	q_{iv}
	no ajustador (não reconhece maioria existente)	q_{d1}
	no ajustador (reconhece maioria inexistente)	q_{d2}
	no *TA* (resultado correto rejeitado ou resultado incorreto aceito)	q_a
Faltas de software relacionadas	nas três versões	q_{3v}
	em duas versões	q_{2v}
	entre *pref e o TA*, entre o secundário e o *TA* e entre o terceiro e o *TA*	q_{pa} , q_{sa}, q_{ta}
Faltas de hardware independente	em um elemento de processamento	q_{ih}

Tabela B.3: Tipos de faltas e probabilidades associadas

Os tipos de faltas considerados na modelagem do esquema CRB correspondem a uma combinação dos tipos considerados na modelagem dos esquemas RB e PNV (tabela B.3).

O submodelo de Segurança de Funcionamento do esquema CRB é construído com base em sua descrição funcional e nos tipos de faltas considerados (Figura B.9). A partir deste submodelo são calculadas as probabilidades de faltas catastróficas (p_{fc}) e benignas por erros por valor (p_{fbv}):

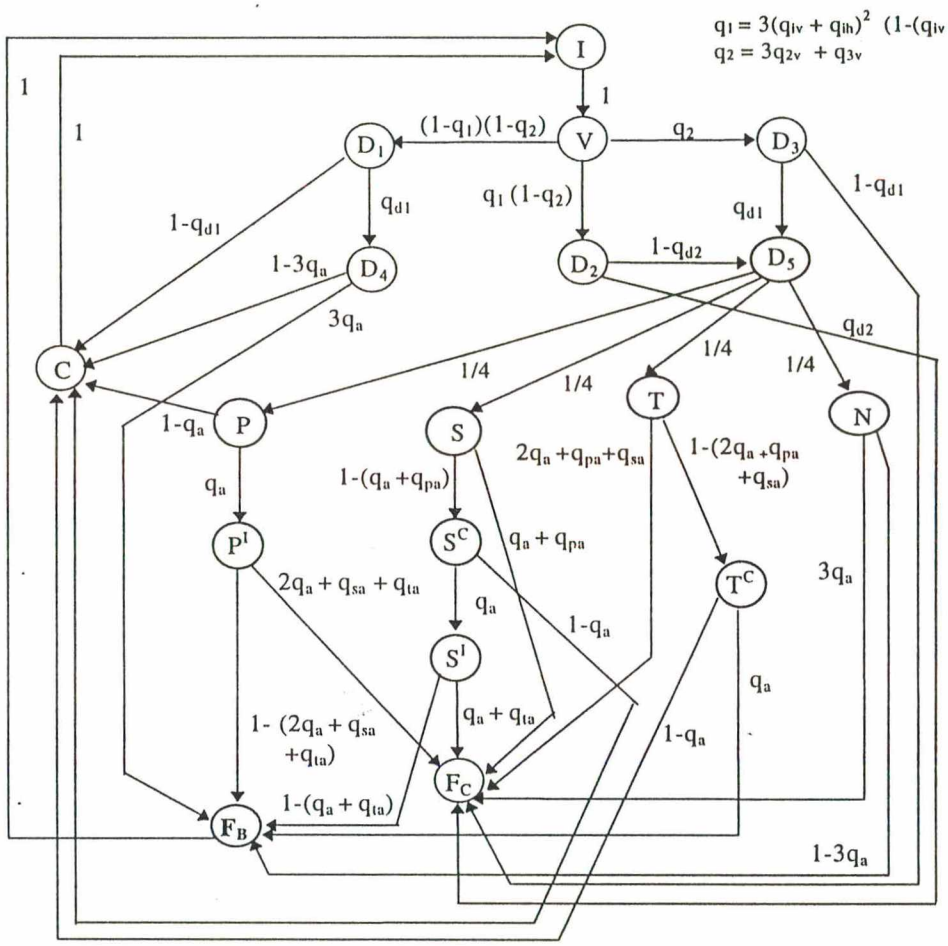
$$p_{fc} = [q_2\, q_{d1} + q_1\,(1-q_2)\,(1-q_{d2})]\,(1/4)\,[q_a\,(2q_a + q_{sa} + q_{ta}) + q_a\, q_{pa} + q_a(1-q_a-q_{pa})\,(q_a+q_{ta}) + (2q_a+q_{pa}+q_{sa}) + 3q_{d2}q_a] + q_1\,(1-q_2)\,q_{d2} + q_2\,(1-q_{d1})$$

$$p_{fbv} = [q_2\, q_{d1} + q_1\,(1-q_2)\,(1-q_{d2})]\;[(1/4)\,[q_a\,(1-(2q_a+q_{sa}+q_{ta})) + q_a\,(1-(q_a+q_{pa}))\,(1-(q_a+q_{ta})) + (1-(2q_a+q_{pa}+q_{sa}))q_a + (1-3q_a)] + 3(1-q_{d1})(1-q_{d2})\,q_{d1}q_a$$

O submodelo de Desempenho do esquema CRB corresponde ao mesmo submodelo do esquema PNV, com o acréscimo do teste de aceitação que é executado após a execução do ajustador quando um resultado de consenso não é obtido ou por não existir uma maioria ou por falha do ajustador. Desta forma, a transformada de Laplace da função densidade de probabilidade do esquema CRB é igual a:

$$F_{sist}^{CRB}(s) = (1-p_1)\,F_{sist}^{PNV}(s) + p_1 F_{sist}^{PNV}(s)\,F_a(s)$$

onde, $F_{sist}^{PNV}(s)$ e $F_a(s)$ correspondem, respectivamente, a transformada de Laplace da *fdp* do esquema PNV e do teste de aceitação e p_1 é a probabilidade do ajustador não obter o resultado de consenso.



Estados	Descrição	Estados	Descrição
I	Estado inicial	V	Execução das versões
D_1	Existe uma maioria representando um resultado correto	D_2	Não existe maioria
D_3	Existe uma maioria representando um resultado incorreto	D_4	Existe uma maioria correta, mas o resultado de consenso não foi obtido devido a uma falha do ajustador
D_5	Existe dois ou mais resultados incorretos (o ajustador não conseguiu obter o resultado de consenso	P	O resultado fornecido pelo principal (versão 1) é correto (os demais são incorretos)
S	O resultado fornecido pelo segundo algoritmo (versão 2) é correto (os demais são incorretos)	T	O resultado fornecido pelo terceiro algoritmo (versão 3) é correto (os demais são incorretos)
N	Não existe nenhum resultado correto entre os três fornecidos	S^C	O teste de aceitação depois de rejeitar o resultado incorreto fornecido pelo principal testa o resultado correto fornecido pelo segundo algoritmo
P^I	O resultado correto fornecido pelo principal é recusado (não existe outro resultado correto)	T^C	O teste de aceitação depois de rejeitar os resultados incorretos fornecidos pelo principal e pelo segundo algoritmo testa o resultado correto fornecido pelo terceiro algoritmo
S^I	O resultado correto fornecido pelo segundo algoritmo é recusado (não existe outro resultado correto)	F_B	Iteração incorreta com falha benigna
C	Iteração correta	F_C	Iteração incorreta com falha catastrófica

Figura B.10: Submodelo de Segurança de Funcionamento para o Esquema CRB

A probabilidade p_1 é conseguida diretamente no submodelo de Segurança de Funcionamento do esquema CRB:

$$p_1 = (1-q_1)\,(1-q_2)\,[(1-q_{d1}) + q_{d1}(1-3q_a)] + q_{d2}\,(1-q_{d1})$$

A *fdp* do tempo de execução do esquema CRB, denominada $f_{sist}^{CRB}(y)$, corresponde a transformada inversa de Laplace da função $F_{sist}^{CRB}(s)$.

Da mesma forma que nos esquemas anteriores, a probabilidade de faltas benignas por erro de temporização (p_{fbt}) e a performabilidade do esquema CRB podem ser obtidas através das equações (5.1) e (5.7), respectivamente, usando as probabilidades p_{fc} e p_{fbv} e a fdp $f_{sist}^{CRB}(y)$.

B.6 Esquema SCOP (*Self-Configuring Optimistic Programming*)

No esquema SCOP a execução dos algoritmos (versões) é organizada em fases, sendo que o subconjunto de versões executadas em cada fase é selecionado dinamicamente. Da mesma forma que em [Chiaradonna 94], a modelagem é feita para o esquema SCOP composto por três versões (figura B.11) e que inicialmente duas versões são executadas paralelamente e os resultados obtidos são enviados ao ajustador que dependendo destes resultados pode obter um resultado de consenso correto (saída 1), incorreto (saída 2) ou então ser incapaz de obtê-lo devido a divergência existentes entre estes resultados (caminho interno). Neste último caso, a terceira versão é executada e a função de ajuste e novamente executada considerando os resultados fornecidos pelas três versões. Nesta segunda execução da função de ajuste o resultado de consenso pode ser correto (saída 3), incorreto (saída 4) ou impossível de ser obtido (saída 5). Para a última possibilidade, ou um resultado *default* é enviado como resposta ou nenhum resultado é enviado. As saídas 2 e 4 estão associadas a ocorrências de falhas catastróficas no sistema. A saída 6 representa o término do deadline estabelecido; neste caso, a execução do serviço corrente é abortada e a próxima iteração começa imediatamente.

Os tipos de faltas consideradas no esquema SCOP são as mesmas do esquema PNV (tabela B.2). O submodelo de Segurança de Funcionamento do esquema SCOP é mostrado na figura B.12. A partir deste submodelo são calculadas as probabilidades de faltas catastróficas (p_{fc}) e benignas por erros por valor (p_{fbv}):

$$p_{fc} = q_2 [(1-q_{d1}) + (q_{d1} (1-q_{d1}) ((q_{2v} + q_{3v}) + (1-(q_{2v} + q_{3v})))] + q_i^2 (1-q_2) q_{d2} [1 + (1-q_{d2}) q_i + (1-q_{d2}) (1-q_i)] + 2q_i (1-q_i)(1-q_2) [q_{d2} + (1-q_{d2})q_i q_{d2})]$$

$$p_{fb} = (1-q_1)(1-q_2) q_{di}^2 + 2q_i (1-q_1) (1-q_2) (1-q_{d2}) [q_i (1-q_{d2}) + q_{d1} (1-q_i)] + q_{d1}^2 (1-q_1)(1-q_2) (1-q_{d2})^2 + q_2 q_{d1}^2$$

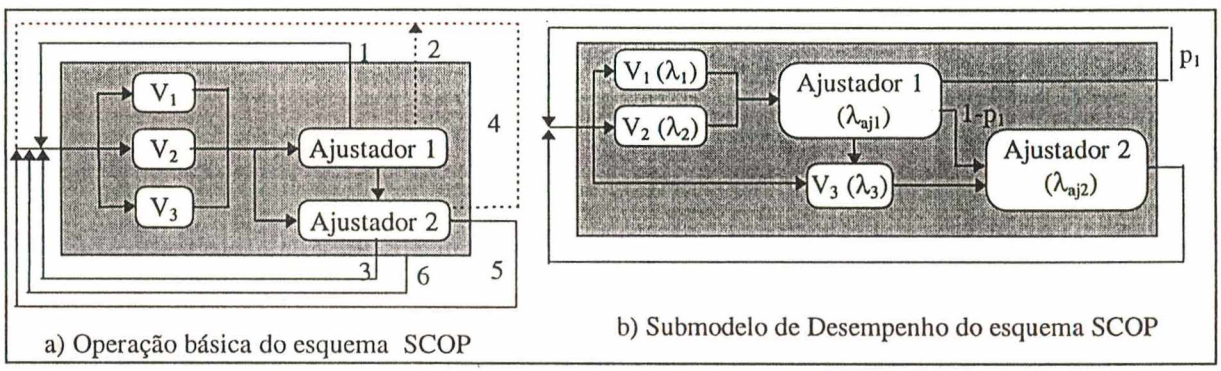


Figura B.11: Estrutura e submodelo de Desempenho do esquema SCOP

O submodelo de Desempenho do esquema SCOP é calculado pela seguinte expressão:

$$F_{sist}^{SCOP}(s) = p_1 G_{2v}(s) F_{aj1}(s) + (1-p_1) G_{2v}(s) F_{aj1}(s) G_{v3}(s) F_{aj2}(s)$$

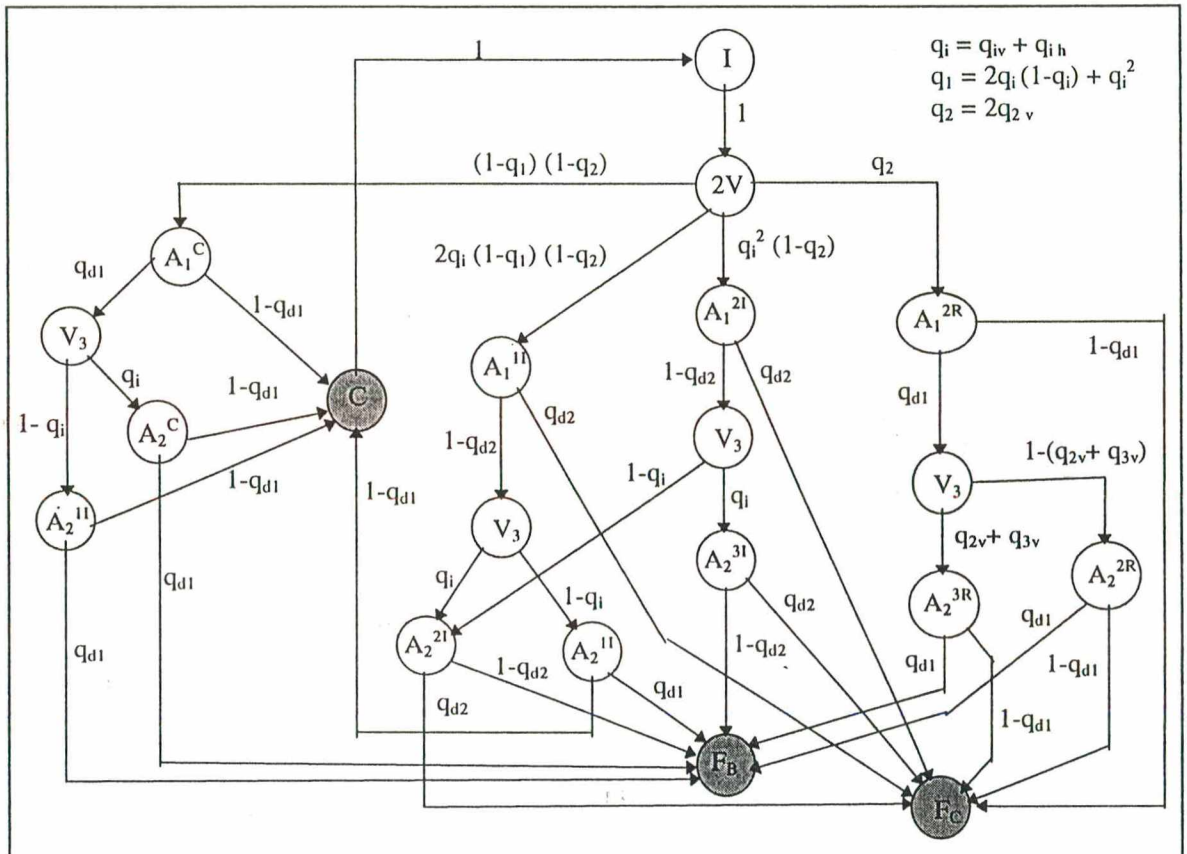
onde, $G_{2v}(s)$, $G_{v3}(s)$, $F_{aj1}(s)$ e $F_{aj2}(s)$ correspondem, respectivamente, a transformada de Laplace da *fdp* do tempo de execução de duas versões em paralelo, da terceira versão (V_3), do ajustador 1 e do ajustador 2, e p_1 é a probabilidade da iteração do esquema terminar após a execução do primeiro ajustado. A probabilidade p_1 é obtida do submodelo de Segurança de Funcionamento a partir da seguinte expressão:

$$p_1 = (1-q_1)(1-q_2) (1-q_{d1}) + 2q_i (1-q_1)(1-q_2) q_{d2} + q_i^2 (1-q_2) q_{d2} + q_2 (1-q_{d1})$$

A transformada $G_{2v}(s)$ é calculada a partir da transformada de Laplace da *fdp* do tempo de execução de duas versões em paralelo, denominado $f_{2v}(y)$, que é igual a derivada da função distribuição de distribuição de probabilidade (*PDF*) do tempo de execução das duas versões em paralelo:

$$G_{2v}(y) = \begin{cases} (1-e^{-\lambda_1 y})(1-e^{-\lambda_2 y}) & se\ y \geq 0 \\ 0 & se\ y < 0 \end{cases}$$

Da mesma forma que nos esquemas anteriores, a probabilidade de faltas benignas por erro de temporização (p_{fbt}) e a performabilidade do esquema SCOP podem ser obtidas através das equações (5.1) e (5.7), respectivamente, usando as probabilidades p_{fc} e p_{fbv} e a função $f_{sist}^{SCOP}(y)$.



Estados	Descrição
I	Estado inicial
2V	Execução de duas versões
V_3	Execução de três versões
A_i^{kR}	Execução do *i-ésimo* (primeiro ou segundo) ajustador com a presença de duas ou três faltas relacionadas.
A_i^{2R}	Execução do *i-ésimo* (primeiro ou segundo) ajustador com a presença de *k* faltas relacionadas.
A_i^{xI}	Execução do *i-ésimo* (primeiro ou segundo) ajustador com a presença de *x* faltas independentes.
A_i^{C}	Execução do *i-ésimo* (primeiro ou segundo) ajustador sem a presença de nenhuma falta.

Figura B.12: Submodelo de Segurança de Funcionamento para o Esquema SCOP

BIBLIOGRAFIA

[Amir 95] Y. Amir, **"Replication Using Group Communication Over a Partitioned Network"**, Thesis the Doctor of Philosophy, Hebrew University of Jerusalem, 1995.

[Alonso 90] R. Alonso and L.L. Cova, **"Managing Replicated Copies in Very Large Distributed Systems"**, Proceedings of the IEEE Workshop on Replicated Data, Nov., 1990.

[Ammann 87] P.E. Ammann and J.C. Knight, **"Data Diversity: An Approach to Sotware Fault Tolerance"**, 17th International Symposium on Fault Tolerant Computing Systems (FTCS), Pittsburgh, 1987, pp. 122-126.

[Anderson 81] T. Anderson and P.A. Lee, **"Fault Tolerance - Principies and Practice"**, Prentice-Hall, 1981.

[ANSA 90] **"A Model for Interface Groups"**, ANSA, ISA Project, APM/RC.093.00, May, 1990.

[Avizienis 77] T. Avizienis and L. Chen, **"On the Implementation of N-Version Programming for Software Fault-Tolerance During Program Execution"**, Proc. COMPSAC 77, Chicago, Nov 1977, pg. 149-155.

[Avizienis 84] A. Avizienis and J.P.J. Kelly, **"Fault Tolerance by Design Diversity: Concepts and Experiments"**, Proc. of the IEEE Computer, Vol. 17, N° 8, August 1984, pg. 67-80.

[Avizienis 86] A. Avizienis and J.C. Laprie, **"Dependable Computing: From Concepts to Design Diversity"**, Proc. of the IEEE, Vol. 74, N° 5, May, 1986, pg. 629-638.

[Barborak 93] M. Barborak, M. Malek and A. Dahbura, **"The Consensus Problem in Fault-Tolerant Computing"**, ACM Computing Surveys, Vol. 25, N° 2, June 1993, pg. 171-220.

[Birman 87a] K. Birman, and T. Joseph, **"Reliable Communication in the Presence of Failures"**, ACM Trans. on Computer Systems, Vol. 5, N° 1, Feb., 1987, pp.47-76.

[Birman 87b] K. Birman and T. Joseph, **"Exploration Virtual Synchronity in Distributed Systems"**,In Proc. of the 11th ACM Symposium on Operating System Principles, Austin, Texas, Nov. 1987, pp. 123-128.

[Birman 90] K. Birman, T. Cooper, K. Mazzulo, M. Makpangou, K. Kane, F. Schmuck, M. Wood, **"The ISIS System Manual 2.1"**, Dept. of Computer Science, Cornell University, Sept., 1990.

[Birman 91] K. Birman, A. Schiper, and P. Stephenson, "**Lightweight Causal and Atomic Group Multicast**", ACM. Trans. On Computer Systems, 9(3), Aug., 1991.

[Birman 93] K.P. Birman, "**The Process Group Approach to Reliable Distributed Computing**", Communication of the ACM, Vol.36, N° 12, 1993, pp 37-53.

[Birman 94] K. Birman and R. van Renesse, **"Reliable Distributed Computing with the ISIS Toolkit"**, Los Alamitos, CA., IEEE Computer Society Press, 1994.

[Blough 90] D.M. Blough and G.F. Sullivan, "**A Comparation of Voting Strategies for Fault Tolerant Distributed Systems**", In Proc. Ninth IEEE Symposium on Reliable Distributed Systems, Alabama, pg. 136-145.

[Bondavalli 90] Bondavalli, A. and Simoncini, L., **"Failure Classification with Respect to Detection"**, Predictably Dependable Computing Systems (PDCS) Esprit Basic Research Action, 1990.

[Bondavalli 93] Bondavalli, A., Di Giandomenico, F. and Xu, J., **"A Cost-Effective and Flexible Scheme for Software Fault Tolerance"**, Journal of Computer Systems Science and Engineering, Vol. 8, N° 4, 1993, pp. 234-244.

[Bondavalli 95] Bondavalli, A., Stankovic, J. and Strigini, L., **"Adaptable Fault Tolerance for Real-Time Systems"**, Responsive Computer Systems: Steps Towards Fault Tolerant Real-Time Systems, Kluwer Academic Publisher, 1995, pp. 187-208.

[Brasileiro 95] F.V. Brasileiro, **"A Software Approach to the Construction of Fail-Controlled Nodes for Distributed Systems"**, VI Simpósio de Computadores Tolerantes a Falhas, Canela, Brasil, Ago., 1995.

[Budhiraja 92] N. Budhiraja, K. Marzullo, F. Schneider, and S. Toueg, "**Optimal Primary-Backup Protocols**", In Proceedings of Sixth International Workshop on Distributed Algorithms, Haifa, Israel, 1992, pp.362-378.

[Buther 93] Buther, R.W. and Finelli, G.B., **"The Unfeasibility of Quantifying the Reliability of Life-Critical Real-Time Software"**, IEEE Trans. on Software Engineering, Vol. 19, N° 1, Jan., 1993, pp. 3-12.

[Chang 84] J.M. Chang and N.F. Maxemchuck, **"Reliable Broadcast Protocols"**, ACM Trans. on Computer Systems, Vol. 2, N° 3, Aug, 1984.

[Chérèque 92] M. Chérèque, D. Powell, P. Reynier, J.L. Richier, and J. Voiron, **"Active Replication in Delta-4"**. In the 22th Annual Int. Symp. on Fault-Tolerant Computing (FTCS-22), IEEE Computer Society Press, Boston, Massachusetts, Jul., 1992, pp. 28-37.

[Chiaradonna 94] S. Chiaradonna, A. Bondavalli and L. Strigini, **"On Performability Modelling and Evaluation of Software Fault Tolerance Structures"**, Esprit Basic Research Project 6362 - Predictably Dependable Computing Systems (PDCS 2), Second Year Report, Sept., 1994, pp. 739-759.

[Costes 81] A. Costes, J. Doucet, C. Landrault and J.C. Laprie, **"SURF: A Program for Dependability Evaluation of Complex Fault-Tolerant Computing Systems"**, In Digest of the 11th Annual Symposium on Fault-Tolerant Computing, IEEE, New York, June, 1981, pp. 313-319.

[Cristian 85] F. Cristian, H. Aghili, R. Strong and D. Dolev, **"Atomic Broadcast: From simple Message Diffusion to Byzantine Agreement"**, In the 15th Annual Int. Symp. on Fault-Tolerant Computing (FTCS-15), IEEE Computer Society Press, Ann Arbor, Michigan, Jun., 1985,pp. 200-206.

[Cristian 89] F. Cristian, **"Synchronous Atomic Broadcast for Redundant Broadcast Channels"**, The Journal of Real-Time Systems, Vol. 2, N° 3, Sept. 1990, pp. 195-212.

[Cristian 94] F. Cristian and S. Mishra, **"Automatic Availability Management in Asynchronous Distributed Systems"**, Proceeding of the Second International Workshop on Configurable Distributed Systems, Pittsburgh, March, 1994.

[Cristian 95] F. Cristian and F. Schmuck, **"Agreeing on Processor Group Membership in Asynchronous Distributed Systems"**, Technical Report CSE95-428, University of California at San Diego, EUA, 1995.

[Cristian 96] F. Cristian, **"Synchronous and Asynchronous Group Communication"**, Communications of the ACM, vol 39, N° 4, April, 1996, pp. 88-97.

[Dasgupta 91] P. Dasgupta, R. LeBlanc Jr, M. Ahamad and U. Ramachandran, **"The Clouds Distributed Operating System"**, IEEE Computer, Vol. 24, N° 11, 1991, pp. 34-44.

[Davies 81] D.W. Davies, **"Protetion"**, in Distributed Systems. Architecture and Implementation (chapter 10), Lecture Notes in Computer Science 105, Springer-Verlay, 1981, pp. 211-245.

[Denning 83] D.E. Denning, **"Cryptography and Data Security"**, Addison-Wesley, 1983.

[Downing 90] A.R. Downing, I.B. Greenberg, and J.M. Peha, **"OSCAR: A System for Weak-Consistency Replication"**, Proc. of the IEEE Workshop Replicated Data, Nov., 1990.

[Duenãs 92] L.T.R. Duenãs, **"Uma Proposta de Algoritmo Assíncrono para Difusão Confiável Atômica"**, Tese de Mestrado, LCMI/UFSC, Dez., 1992.

[Dwork 88] C. Dwork, N. Lynch and L. Stockmeyer, **"Consensus in the Presence of Partial Synchrony"**, Journal of the ACM, Vol. 35, N° 2, Apr, 1988, pp. 288-323.

[Edwards 94] H. Edwards and O. Rees, **"A Model for Failures in Dependable Systems"**, APM. 1143.01, APM Ltd., Cambridge, UK, March, 1994.

[Estivill-Castro 92] V. Estivill-Castro, D. Wood, **"A Survey of Adaptive Sorting Algorithms"**, ACM Computing Surveys, Vol. 24, N° 4, Dec., 1994, pp. 441-476.

[Ezhilchelvan 89] P.D. Ezhilchelvan, S.K. Shrivastava, and A. Tully, **"Constructing Replicated Systems Using Processors With Point to Point Communication Links"**, Proceedings of the 16th Annual Symposium on Computer Architecture, Jerusalem, June, 1989, pp. 177-184.

[Fabre 95] J. Fabre, N. Nicomette, T. Pérennou, R.J. Stroud and Z. Wu, **"Implementing Fault Tolerant Application using Reflective Object-Oriented Programming"**, Proceedings of the 25th IEEE International Symposium on Fault-Tolerant Computing, Pasadena (CA), June, 1995.

[Ficher 85] M. Ficher, N. Lynch, and M. Paterson, **"Impossibility of Distributed Consensus with one Fault Processor"**, Journal of the ACM, Vol. 32, N° 2, Apr. 1985, pp. 374-382.

[Fraga 91] Fraga, J.S., Rodrigues, V., e Silva, E.S., **"Programação do DRB através da Linguagem LIS"**, XIII SEMISH, Santos, Ago., 1990.

[Garcia-Molina 90] H. Garcia-Molina and D. Barbará, **"The Case for Controlled Inconsistency in Replicated Data"**, Proc. of the IEEE Workshop on Replicated Data, Nov., 1990.

[Giandomenico 90] F.L Giandomenico and Strigini, **"Adjudicators for Diversity Redundant Components"**, in Proceedings of the 20th International Symposium on Fault-Tolerant Computing , Alabama, 1990, pg 114-123.

[Gray 86] J. Gray, **"Why do Computers Stop and What can be done about it?"**, In Proc. 5th Symp. on Reliability in Distributed Software and Database Systems", IEEE, Los Angeles, CA., Jan., 1986, pp. 3-12.

[Grüsteidl 91] G. Grünsteidl, H. Kantz and H. Kopetz, **"Communication Reliability in Distributed Real-Time Systems"**, 10th IFAC Workshop on Distributed Computer Control Systems, Semmering, Austria, Sept. 1991.

[Hadzilacos 93] V. Hadzilacos and S. Toueg, **"Fault-Tolerant Broadcasts and Related Problems"**, Distributed Systems (Chapter 5) Second Edition, Ed. Sape Mullender, Addison-Wesley, ACM Press, 1993.

[Hoare 85] C.A.R. Hoare, **"Communications Sequential Processes"**, Prentice-Hall International, 1985.

[Hoüzle 94] U. Hoüzle, **"Adaptive Optimization for Self: Reconciling High Performance with Exploratory Programming"**, PhD Thesis, Computer Science Department, Stanford University, Aug., 1994.

[Hudak 93] Hudak, J.J., Suh, B.H., Segall, Z. and Siewiorek D.P., **"Evaluation & Comparation of Fault-Tolerant Software Techniques"**, IEEE Trans. on Reliability, Vol. 42, N° 2, June, 1993, pp. 190-204.

[Hynes 95] C. Hynes, **"An Example Guidance Mode Specification"**, Technical Report, NASA Ames, 1995.

[Jalote 94] P. Jalote, **"Fault Tolerance in Distributed Systems"**, PRT Prentice Hall, 1994.

[Johnson 88] A.M. Johnson Jr and M. Malek, **"Survey of Software Tools for Evaluating Reliability, Availability and Serviceability"**, ACM Comp. Surveys, Vol. 20, N° 4, Dec., 1988, pp. 227-269.

[Kaashoek 92] M.F. Kaashoek, A.S. Tanenbaum, **"Efficient Reliable Gruop Communication for Distributed Systems**, Rapport IR-295, Faculteit Wiskunde en Informatica, Vrije Universiteit, Jul. 1992.

[Kelly 91] J.P.J. Kelly, T.I. Mcvittie and W.I. Yamamoto, **"Implementing Design Diversity to Achieve Fault Tolerance"**, IEEE Software, July, 1991, pp. 61-71.

[Kim 88] K.H. Kim, "**Approaches to Implementation of a Repairable Distributed Recovery Block Scheme**", 18th International Symposium of Fault-Tolerant Computing, Pittsburgh, June, 1988.

[Kim 92a] K. Kim, H. Hopetz, K. Mori, E. Shokri, and G. Grünsteidl, "**An Efficient Decentralized Approach to Processor-Group Membership Maintenance in Real-Time LAN Systems: The PRHB/ED Scheme**", In Proceedings of the 11th Symposium on Reliable Distributed Systems, Houston, USA, Oct., 1992.

[Kim 92b] K. Kim and C.E. Collum, "**Fault-Tolerant Execution of Real-Time Tasks Through Duplex Assignment within Parallel Computers**", in Proc. of the International Conference on Parallel and Distributed Systems, Taiwan, Dec., 1992.

[Kopetz 89] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, C. Senft, and R. Zainlinger, "**The MARS Approach**", IEEE Micro, Vol. 9, N° 1, Feb., 1989, pg. 25-40.

[Kopetz 91] H. Kopetz, G. Grünsteidl, and J. Reisinger, "**Fault-Tolerant Membership Service in a Synchronous Distributed Real-Time System**", In Dependable Computing for Critical Applications, Vol. 4 of Dependable Computing and Fault-Tolerant Systems, A. Avizienis and J.C. Laprie (ed), Springer Verlag, 1991, pg. 441-429.

[Kopetz 94] H. Kopetz and G. Grünsteidl, "**TTP - A Protocol for Fault-Tolerant Real-Time Systems**", IEEE Computer, Vol. 27, N° 1, Jan. 1994,pp. 14-23

[Koutny91] M. Koutny, L. Mancini and G. Pappalardo, "**Replication in Acyclic Networks of Communicating Processes**", Techical Report, Computing Laboratory, The University of Newcastle upon Tyne, N° 378, April, 1992.

[Lamport 82] L. Lamport, R. Shostak, and M. Pease, "**The Byzantine General Problem**", **ACM Trasns. on Programming Languages and System**", Vol. 4, N° 3, Jul. 1982, pp. 382-401.

[Ladin 90] R. Landin, B. Liskov, L. Shrira, and S. Ghemawat, "**Lady Replication Exploiting the Semantics of Distributed Services**", In Procedings of the 9th Annual Symposium on Principles of Distributed Computing, Aug., 1990, pp.43-58.

[Laprie 87] J.C. Laprie, J. Arlat, C. Beounes, K. Kanoun, C. Hourtolle, "**Hardware-and-Software Tolerance: Definition and Analysis of Architectural Solutions**", Proc. 17th International Symposium on Fault-Tolerant Computing, Pittsburg, Pennsylvania, USA, Jul. 1987.

[Laprie 90] J.C. Laprie, C. Beounes, K. Karnoun, "**Definition and Analysis of Hardware-and-Software Fault-Tolerant Architectures**", PDCS Esprit Basic Research Action, 23, May, 1990.

[Laprie 92] J.C. Laprie (Ed), "**Dependability: Basic Concepts and Terminology**", Vol. 5 of Dependable Computing and Fault-Tolerant Systems, Springer Verlag. Wien, New York, 1992, pp 23-28.

[Lung 96] L.C. Lung, "**Implementação de Técnicas de Replicação de Componentes de Software em uma Plataforma Aberta CORBA**", Dissertação de Mestrado, LCMI/UFSC, Julho, 1996.

[Lea 94] D. Lea, **"Object in Groups"**, SUNY at Oswego/NY CASE Center, Submitted, ECOOP 94.

[Lemos 88] Lemos, R., **"Funções de Gerenciamento para um Sistema Distribuído Aplicado na Automação de Usinas e Subestações"**, Dissertação de Mestrado, UFSC, BR, Ago., 1988.

[Liang 90] L. Liang , S. T. Chanson e G. W. Neufeld, **"Process Groups and Group Communications: Classifications and Requirements"**, IEEE Computer, pp 56-65, February 1990.

[Liskov 88] B. Liskov, **"Distributed Programming in Argus"**, Communications of the ACM, Mach 1988.

[Little 94] M.C. Little and D.L. McCue, **"The Replica Management Systems: A Scheme for Flexible and Dynamic Replication"**, Proceeding of the Second International Workshop on Configurable Distributed Systems, Pittsburgh, March, 1994.

[Liu 91] J.W.S. Liu, K.J. Lin, W.K. Shih, A.C. Yu, J.Y. Chung, and W.Zhao, **"Algorithms for Sheculing Imprecise Computing"**, IEEE Computer, Vol. 24, May, 1991, pp.58-68.

[Lisbôa 95] M.L.B. Lisbôa, **"MOTF: Metaobjetos para Tolerância a Falhas"**, Tese de Doutorado, Instituto de Informática, UFRGS, Dez., 1995.

[Lyu 91] M.R. Lyu and A. Avizienis, **"Assuring Design Diversity in N-Version Software: A Design Paradigm for N-Version Programming"**, in Proc. 2° International Working Conference on Dependable Computing for Critical Application, Arizona, Feb., 1991, pp. 89-98.

[Maes 87] P. Maes, **"Concepts and Experiments in Computational Reflection"**, Proc. OOPSLA'87, ACM SIGPLAN Notices, 22(12), Ed. N. Meyrowitz, 1987, pp. 147-155.

[Mancini 88] L. Mancini, and G. Pappalardo, **"Towards a Theory of Replicated Processing"**, Lecture Notes in Computer Science 331, Spring-Verlay, 1988, pp. 175-192.

[Meer 93] H.D. Meer and H. Mauser, **"A modeling Approach for Dynamically Reconfigurable Systems"**, in Responsive Computer Systems (H. Kopetz and Y. Yakuda, eds.), Vol. 7 of Dependable Computing and Fault-Tolerant Systems, Wien, Austria: Springer-Verlag, 1993, pp. 357-376.

[Melliar-Smith 90] P.M. Melliar-Smith, L.E. Moser, and V. Agrawala, **"Broadcast Protocols for Distributed Systems"**, IEEE Trans. on Parallel and Distributed Systems, Vol. 1, N° 1, Jan., 1990, pg. 17-25.

[Menascé 94] D. Menascé, V. A. F. Almeida, and L.W. Dowdy, **"Capacity Planning and Performance Modeling - From Mainframes To Client-Server Systems"**, prentice Hall, 1994.

[Meyer 80] J.F. Meyer, **"On Evaluating the Performability of Degradable Computing System"**, IEEE Trans. Computers, Vol. C-29, 1980, pp. 720-731.

[Mori 86] K. Mori, and al., **"Autonomous Decentralized Software Structure and Its Application"**, Proc. Fall Joint Comp. Conference, Dallas, Texas, Nov. 1986.

[Modugno 96] F. Modugno, N.G. Leverson, J. D. Reese, K. Partridge and S. Sandys, **"Integrated Safety Analysis of Requirements Specifications"**, University of Washington, http://www.cs.washington.edu/research/projects/safety/www, May, 1996.

[Nacamura 94] L. Nacamura Júnior and J.S. Fraga, **"Proposição de um Modelo de Tolerância a Faltas formado por Múltiplas Replicações: Modelo MR"**, 12° Simpósio Brasileiro de Redes de Computadores (SBRC), Curitiba, Maio, 1994, pp. 564-583.

[Nacamura 95] L. Nacamura Júnior and J.S. Fraga, **"Um Modelo de Execução para o Esquema de Tolerância a Faltas MR"**, VI SCTF - 6° Simpósio de Computadores Tolerantes a Falhas, Canela, RS, Brasil, 1995.

[Nelson 90] V.P. Nelson, **"Fault-Tolerant Computing: Fundamental Concepts"**, Computer, Jully 1990, pp. 19-37.

[Park 96] D. Park, **"Adaptive Execution Improving Performance Through the Run-Time Adaptation of Performance Parameters"**, PhD. Thesis (Computer Science), University of Southern California, May, 1996.

[Parker 96] R.H. Parker, **"Adaptive Computing System: Chameleon"**, Slide presentation, http://www.ito.darpa.mil/ResearchAreas/Adaptative _Computing_Systems,1996.

[Paris 88] J.F. Paris and D.D.E.Long, **"Efficient Dynamic Voting Algorithms"**, In Procedings of the 4th International Conference on Data Engineering, Feb., 1988, pp. 268-275.

[Poledna 94] S. Poledna, **"Replica Determinism in Fault-Tolerant Real-Time Systems"**, PhD Thesis, Techische Univesität Wien, April, 1994.

[Powell 90] D. Powell, **"Fault Assumptions and Assumption Coverage - A Contribution to the Fundamental Concepts of Dependability"**, LAAS Report N° 90074, February 1990.

[Powell 91] D. Powell, **"Delta-4 Architecture Guide"**, Esprit II P2252, Delta-4 Phase 3, August 1991.

[Randell 75] B. Randell, **"System Structure for Software Fault Tolerance"**, IEEE Transaction on Software Engineering, June 1975, pg. 220-232.

[Randell 94] Randell, B. and Xu, Jie, **"Recovery Block"**, Esprit Basic Research Project 6362 - Predictably Dependable Computing Systems (PDCS 2), Second Year Report, Sept., 1994, pp. 251-273.

[Reibaman 91] Reibman, A.L., Veeraraghavan, M., **"Reliability Modeling : An Overview for System Designers"**, IEEE Computer, April, 1991, pp. 49-57.

[Rennes 84] Rennels, D.A., **"Fault-Tolerant Computing - Concepts and Examples"**, IEEE Trans. on Computers, Vol. 33, N° 12, Dec., 1984, pp. 1116-1129.

[Sahner 86] R.A. Sahner and K.S. Trivedi, **"Reliability Modeling using SHARPE"**, IEEE Trans. Reliability, R-36,(2), Jun., 1987, pp. 186-193.

[Schneider 90] F. B. Schneider, **"Implementing Fault-Tolerant Service Using the State Machine Approach: A Tutorial"**, ACM Computing Survey, 22(4):299-319, Dec 1990.

[Scoth 87] P.K. Scoth, J.W. Gault and D.F. McAllister, **"Fault-Tolerance Software Reliability Modeling"**, IEEE Trans. on Software Engineering, Vol. SE-13, N° 5, May, 1987.

[Shrivastava 92] S.K. Shrivastava, P.D. Ezhilchelvan, N.A. Speirs and D.T. Seaton, **"Fail-Controlled Computer Architecture for Distributed Systems"**, Technical Report N° 333, University of Newcastle upon Type, August, 1992.

[Shin 89] K.K. Shin and J.W. Dolter, **"Alternative Majority-Voting Methods for Real-Time Computing Systems"**, IEEE Trans. on Reliability, Vol. 38, N°1, April, 1989, pg. 58-67.

[Stok 90] P.D.V. van der Stok, **"Failure Models and Classification"**, Dep. of Mathematics and Computing Science, Eindhoven University of Technology, Oct., June, 1990.

[Schepers 90] H. Scherpers, **"Terminology and Paradigms for Fault Tolerance"**, Dep. of Mathematics, Eindhoven University of Technology, Oct., 1990.

[Tai 91] A.T. Tai, **"Performability Concepts and Modeling Techniques for Real-Time Software"**, PhD dissertation, 1991, UCLA Computer Science Department, Los Angeles.

[Tai 93a] A.T. Tai, A. Avizienis, J.F. Meyer, **"Evaluation of Fault-Tolerant Software: A Performability Modeling Approach"**, Vol. 6 of Dependable Computing and Fault-Tolerant Systems, Spring-Verlag, 1993, pp. 113-135.

[Tai 93b] A.T. Tai, A. Avizienis, J.F. Meyer, **"Performability Enhancement of Fault-Tolerant Software"**, IEEE Trans. On Reliability , Vol. 42, N° 2, June, 1993, pp. 227-237.

[Tai 94] A.T. Tai, **"Performability-Driven Adaptive Fault Tolerance"**, In 24th Annual International Symposium on Fault-Tolerant Computing, Austin, Texas, June, 1994, pp.176-185.

[Tindell 94] K.W. Tindell, A. Burns, A. J. Wellings, **"An Extendible Approach for Analyzing Fixed Priority Hard Real-Time Tasks"**, Real-Time Systems, 1994, pp. 133-151.

[Trivedi 82] Trivedi, K.S., **"Probability and Statistics with Reliability, Queuing, and Computer Science Applications"**, Prentice-Hall, Englewood Cliffs, N.J., 1982.

[Tully 90a] A. Tully, and S. Shrivastava, **"Preventing State Divergence in Replicated Programs"**, In 9th Symposium on Reliable Distributed System, Computer Society Press of the IEEE, Huntsville, Alabama, EUA, 1990, pp. 104-113.

[Tully 90b] Tully, A., **"Preventing State Divergence in Replicated Distributed Systems"**, Ph.D Thesis, The University of Newcastle upon Tyne Computing Laboratory, UK, Sep., 1990.

[Ungar 87] **D. Ungar and R.B. Smith,** "Self: The Power of Simplicity", **OOPSLA'87 Conference Proceeding, Orlando, FL, Oct, 1987, pp 227-241.**

[Veríssimo 89] P. Veríssimo, **"Comunicação em Grupo Fiável em Sistemas Distribuídos sobre Rede Local"**, Tese de doutorado, IST-INESC, Dez. 1989.

[Wakerley 78] J. Wakerley, **"Error Detecting Codes Self-Checking Circuits and Applications"**. North-Holland, 1978.

[Wang 95] F. Wang, K. Ramamritham and J. Stankovic, **"Determining Redundancy Level for Fault-Tolerant Real-Time Systems"**, Special Issue of IEEE Transactions on Computer on Fault Tolerant Computing, Vol. 44, N° Feb., 1995.

[Webber 91] S. Webber, **"The Stratus Architecture"**, Chap. 13 in The Theory and Practice of Reliable System Design by D.P. Siewiorek and R.S. Swarz, Digital Press, Bedford, Mass., 1991.

[Wensley 78] J.H. Wensley et al., **"SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control"**, Proc. IEEE, Vol. 66, N° 10, Oct., 1978, pp. 1240-1255.