PROPOSIÇÃO DE UM MODELO DE TOLERÂNCIA A FALTAS FORMADO POR MÚLTIPLAS REPLICAÇÕES: MODELO MR

Luiz Nacamura Júnior e Joni da Silva Fraga

Laboratório de Controle e Microinformática - LCMI-DEEL-UFSC
Campus Universitário - Trindade - Florianópolis - SC
Caixa Postal 476 - CEP 88040-900
E-mail: nacamura@lcmi.ufsc.br, fraga@lcmi.ufsc.br

RESUMO: Este artigo trata de tolerância a faltas em sistemas distribuídos ou sistemas multiprocessadores. Neste texto é introduzido um esquema de tolerância a faltas, o modelo MR (**Múltiplas Replicações**), cuja característica é a ativação de redundâncias adequadas aos atributos do dado de entrada. Esta flexibilidade diferencia este esquema dos modelos mais tradicionais. O texto apresenta um estudo comparativo onde os resultados obtidos são tirados de uma simulação numérica.

ABSTRACT: This paper addresses the fault-tolerance in distributed systems or multiprocessors. This text introduces a scheme of fault-tolerance, the multiple replication model (MR), whose main characteristic is the activation of redundancies adequate to data input attributes. This flexibility distinguishes our scheme from others traditional models. It is also presented a comparative study where the results were obtained from a numeric simulation.

PALAVRAS CHAVES: Sistemas Distribuídos, Tolerância a faltas, Modelos de Tolerância a faltas

1. Introdução

A utilização de modelos de tolerância a faltas em sistemas computacionais tem por objetivo o desenvolvimento de aplicações com alto grau de confiabilidade, disponibilidade e segurança. Num sistema distribuído é possível que mesmo quando da ocorrência de falhas parciais determinados serviços ainda sejam fornecidos, muitas vezes de maneira degradada.

Vários modelos de tolerância a faltas para sistema distribuído foram e ainda estão sendo propostos na literatura. A maioria deles é baseado em três modelos bem conhecidos: o bloco de recuperação, a programação N-versões e o N-Modular Redundancy" (NMR). Os dois primeiros são usados na tolerância a faltas de software, sendo baseados no princípio da diversidade de projeto [Avizieniz 84] que pressupõe a possibilidade de desenvolver vários programas (ou algoritmos) diferentes, a partir de uma mesma especificação. A técnica NMR [Wensley 78], baseada em mecanismo de voto é a solução clássica para faltas de hardware. Os modelos usuais encontrados na

literatura combinam ou estendem as caracteristicas destas técnicas. Estes modelos também são caracterizados por estruturas pouco flexíveis e não adaptáveis às condições evolutivas de um processamento em sistema distribuído.

O modelo MR (**Múltiplas Replicações**) foi introduzido em [Nacamura 92] no sentido de permitir estruturas mais flexíveis. 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 a **diversidade de projeto** [Avizienis 84] ou mesmo de **computação imprecisa** [Yu 91] em aplicações tempo real. Os algoritmos base das replicações que compõem o esquema NMR são soluções diferentes de um mesmo problema, diferenciadas apenas pelos graus de elaboração e de desempenho. Os dados de entrada apresentam atributos que permitem escalonar replicações adequadas ao processamento dos mesmos. Esta característica de ativação de redundâncias a partir de atributos de dados de entrada, fazem do modelo MR um instrumental poderoso na construção de servidores confiáveis que atendem uma larga faixa de necessidade em termos de processamento.

O objetivo deste artigo é apresentar um estudo comparativo realizado entre o MR e os modelos mais significativos encontrados na literatura. Os resultados da comparação são obtidos de uma simulação numérica que mede a eficiência destes esquemas com base na quantificação de erros detectados e não detectados em diferentes hipóteses de faltas. Neste artigo, uma revisão bibliográfica é apresentada descrevendo os esquemas mais conhecidos de tolerância a faltas. O modelo MR é também objeto de apresentação, onde suas principais características são explicitadas. A descrição da simulação e a discussão dos resultados obtidos encerram este texto.

2. Trabalhos relacionados

Atualmente, existem na literatura várias propostas de arquiteturas ou modelos que combinam redundâncias de software e de hardware. A maioria destes esquemas são composições de duas técnicas bem conhecidas: o **Bloco de Recuperação** (RB) [Randell 75] e a **Programação N-Versões** (PNV) [Avizienis 77]. Na abordagem de bloco de recuperação, um serviço confiável, isto é, um serviço com capacidade de tolerar faltas, é implementado a partir de blocos compostos de um algoritmo principal, de um ou mais algoritmos secundários e de um teste de aceitação. Os algoritmos do bloco de recuperação são ordenados segundo critérios pré-estabelecidos, por exemplo, grau de precisão, desempenho, etc. Quando o serviço é ativado, o algoritmo principal é executado e o resultado produzido submetido ao teste de aceitação. Caso o resultado gerado não passe pelo teste, como técnica de "backward", o estado anterior à execução do algoritmo principal é restaurado, e a primeira alternativa na ordenação de algoritmos é então executado e seus resultados também testados. Este processo de testes, recuperações e ativações de algoritmos se dá até o ponto em que um dos algoritmos da ordenação produza um resultado que passe pelo teste de aceitação, ou então que não exista mais algoritmo disponível na ordenação. Neste último caso, uma exceção é gerada, devendo ser tratada nos níveis superiores. Por diferentes razões, o teste de

aceitação constitui-se no aspecto mais crítico do esquema de bloco de recuperação. 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].

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 votador que a partir de um procedimento de comparação, obtém um resultado de consenso. Caso não seja possível obter um valor de consenso, o votador envia uma mensagem sinalizando a exceção, devendo o tratamento desta exceção ser executado nos níveis superiores. A comparação dos resultados pelo votador consiste no aspecto mais complexo da técnica de N-versões, pois não existe um procedimento de votação geral que pode ser empregado em todas as situações. Por exemplo, quando os valores a serem comparados são caracteres ou inteiros, a comparação se torna bastante simples, e consiste na verificação de igualdade, na qual prevalece o resultado majoritário (votação exata). Entretanto, quando os resultados são diferentes dos citados, o procedimento de comparação é mais complexo, consistindo em obter um valor de consenso a partir de resultados diferentes porém corretos (votação inexata) [Giandomenico 90], [Shin 89] e [Blough 90].

O esquema de Bloco de Recuperação em Consenso (CRB) [Scott 87] procura eliminar os problemas das duas técnicas anteriores a partir da composição de seus componentes. Sendo assim, a estrutura do esquema CRB consiste de N versões (algoritmos) de um mesmo programa, de um teste de aceitação e de um procedimento de votação. Da mesma forma que no RB, as versões são classificadas segundo critérios pré-estabelecidos. Quando da chamada do serviço, todas as versões são executadas paralelamente e suas saídas são submetidas ao votador. Na implementação proposta por Scott assumi-se que não existem falhas de modo comum entre as versões (falhas de modo comum são aquelas que têm por origem uma mesma falta, presente em duas ou mais versões que força resultados idênticos e errôneos). Considerando esta hipótese, o procedimento de votação pode se basear na obtenção apenas de duas saídas iguais (votação 2-out-of-N). Se não existe duas saídas iguais, a saída da melhor versão é examinada através de um teste de aceitação que pode aceitar ou então rejeitar a saída. No caso de rejeição as outras versões servirão de alternativas e portanto também terão suas saídas submetidas ao teste de aceitação.

A abordagem N Componentes Auto-Verificáveis (NSCP) ("N Self-Checking Programming) apresentada em [Laprie 87] é composta de N componentes com mecanismos internos de detecção e confinamento de erro, ou seja cada componente é estruturado a partir de dois algoritmos (versões) distintos e de um comparador. Embora, os N componentes se executem paralelamente (2N versões), somente um dos componentes é o ativo, os outros são usados como "spares". Os resultados fornecidos

pelas versões do componente ativo são comparados e caso sejam "iguais" (a noção de igualdade depende do procedimento de votação), o componente é considerado correto e o resultado obtido pela comparação corresponde a saída do esquema NSCP. Por outro lado, se os resultados não coincidem, o controle é chaveado para o próximo componente "spare" que envia um resultado, caso seja possível de obtê-lo, ou passa o controle adiante. Esta sequência prossegue até que um dos componentes envie um resultado correto ou então que se esgote o número de "spares". Traçando um paralelo como o modelo RB, esta abordagem apresenta a vantagem do teste de aceitação ser reduzido a um comparador (exato ou inexato) entre dois resultados, o que facilita a sua implementação. Em [Laprie 90] é descrito e analisado uma série de combinações de 2N versões de um programa ou algoritmos em componentes auto-verificáveis. Este esquema apresenta a desvantagem de necessitar de um número elevado de hardware redundantes (2N).

O modelo DRB ("Distributed Recovery Block") proposto em [Kim 88], consiste na execução em paralelo de dois elementos de processamento, sendo que um deles é o nó primário e o outro o secundário. Estes nós são blocos de recuperações com apenas duas alternativas, ou seja cada um dos nós é composto de duas alternativas e de um teste de aceitação. Além disto, do ponto de vista estrutural, os nós são idênticos, isto é, apresentam o mesmo conjunto de algoritmos e o mesmo teste de aceitação. A distinção ocorre durante a execução, quando o nó primário apresenta uma das alternativas como principal e o secundário a outra. Ambos os nós recebem os dados de entrada, e após a execução, os resultados são submetidos ao teste de aceitação. Se a saída produzida pelo nó primário, através da alternativa principal, é aceita pelo teste, o resultado é enviado como resposta do serviço. Em caso contrário, o controle é transferido para o nó 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 os ciclos de processamento quando da ocorrência de faltas no nó primário, ou seja, a função do nó primário é desempenhada pelo antigo secundário, com o nó faltoso (nó primário do ciclo anterior) passando a ser o secundário após a reconfiguração.

Com relação ao mecanismo de tolerância a faltas empregado, os modelos acima podem ser divididos em três classes: a primeira classe é composta dos modelos RB e DRB que usam o teste de aceitação para a verificação da correção dos resultados, a segunda do modelo NVP que utiliza o votador para mascarar erros que porventura venham a ocorrer em um ou mais algoritmos e a terceira composta pelos modelos NSCP e CRB que se utilizam tanto de teste de aceitação como de votador.

A tolerância a faltas de hardware, nos esquemas citados (à exceção do RB), é implementada através da execução dos algoritmos disponíveis em nós de processamentos distintos de um sistema distribuído ou de um sistema multiprocessadores. Um modelo de estruturação de aplicações bastante conhecido e utilizado na prática com fim específico de tolerar faltas de hardware é o NMR (**"N Modular Redundancy"**). O NMR pode ser visto como um caso particular de programação N-versões, em que as N cópias de um

mesmo algoritmo são executadas paralelamente e os resultados são enviados para uma votação exata.

3. O modelo de Múltiplas Replicações (MR)

O modelo MR ("Múltiplas Replicações"), introduzido em [Nacamura 92] 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 a **diversidade de projeto** ou mesmo de **computação imprecisa** em aplicações tempo real. 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. E 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 (rede local) 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. Fundamentado nisto, o esquema MR pode ser sintetizado pela expressão:

$$MR := (C^N ? m_A \rightarrow P_A^N \mid C^N ? m_B \rightarrow P_B^N \mid C^N ? m_C \rightarrow P_C^N \mid ... \mid C^N ? m_K \rightarrow P_K^N);$$

onde 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 da replicação 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 técnicas usuais de detecção de erros e mascaramento de erro como **voto majoritário** ou **ajustadores** e/ou **teste de aceitação** e temos o esquema MR completo.

A implementação das replicações que compõem o esquema MR pode se dar seguindo modelos de implementação de processamento de grupo normalmente presentes na literatura: primário/secundário [Birman 89], líder/seguidores [DELTA_4 90], ou réplicas ativas e não privilegiadas. A definição por um destes modelos de implementação dependerá então da combinação de réplicas ativas e passivas, e dos mecanismos de detecção de erro usados na implementação do MR. Em [Nacamura 92] são examinados os aspectos ligados aos diferentes modelos de implementação do esquema MR e das diferentes combinações de mecanismos de detecção de erro.

Neste artigo nos deteremos no uso de replicações segundo o modelo de réplicas ativas e não privilegiadas (implementação no estilo NMR). Consideramos também as saídas (resultados) de cada uma das réplicas submetidas a um **teste de aceitação**. Com isto, a tolerância a faltas de projeto (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, a subseqüente restauração do estado inicial do MR neste ciclo de processamento e o escalonamento de uma nova replicação que possa atender os atributos do dado de entrada, embora de maneira menos precisa. Este processo pode se repetir com diferentes replicações que podem atender os atributos do dado, até que se tenha um resultado que passe no teste de aceitação.

Técnicas de tolerância a faltas de software estão baseadas no princípio da diversidade de projeto que estabelece a possibilidade de, a partir de uma metodologia meticulosa, implementar vários algoritmos independentes entre si, de uma mesma especificação conseguida dos requisitos de um problema. Os esquemas RB, DRB que também fazem uso da diversidade de projeto, apresentam os diferentes algoritmos alternativos ordenados estaticamente. Esta ordenação é uma relação pré-estabelecida e independente dos dados de entrada, ou seja, os dados são sujeitos a mesma sequência de algoritmos, até que um deles forneça um resultado que passe pelo teste de aceitação.

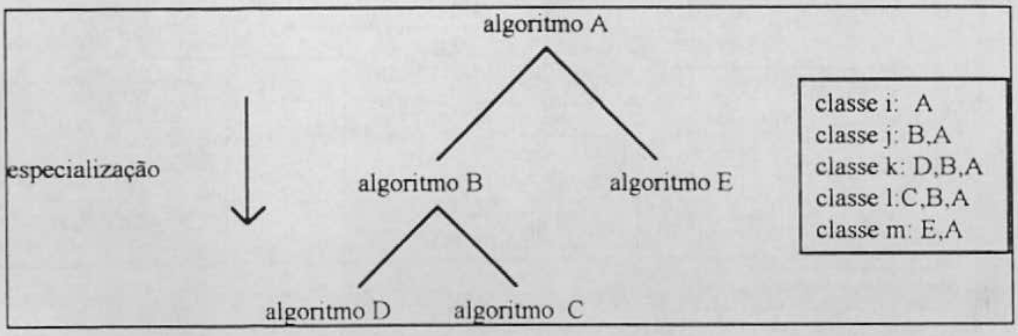

Figura 1: Hierarquia de especialização e classes de algoritmos

No esquema MR, os algoritmos base das replicações estão classificados de acordo com o tipo ou características do dado de entrada. Além disto, diferentes algoritmos podem atender o mesmo tipo de dados, com mais ou menos precisão, formando então o que chamamos de **classes de dados**. Numa mesma classe de dados, os algoritmos atendem ao tipo de classe, ordenados de acordo com o grau de precisão: tem-se algoritmos mais especializados e algoritmos mais gerais para o trato do dado do tipo. A ordenação geral dos algoritmos segundo atributos, formando as classes é denominada de **hierarquia de especialização**. A figura 1 ilustra a formação de classes usando uma hierarquia de algoritmos.

Para um dado de entrada, o teste de escalonamento é responsável pela determinação da lista de algoritmos que fazem parte da **classe do dado**. O primeiro algoritmo da lista é denominado **preferencial** (o mais preciso), cuja replicação

correspondente será executada na recepção do dado, no ciclo de processamento. Os outros algoritmos da lista serão alternativas ao preferencial: a lista é percorrida no sentido contrário a especialização, até que um dos algoritmos tenha seus resultados passando pelo teste de aceitação. Da figura 1, pode-se tirar que se o dado de entrada pertence a 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) só são executados no caso de falha do algoritmo preferencial. É importante observar o comportamento dinâmico da seleção de algoritmos, por exemplo, para um dado de entrada de classe m, a lista de algoritmos é formada pelos algoritmos E e A.

Por exemplo, considere o caso prático de busca de um determinado elemento em uma estrutura de dados. Esta estrutura pode ser um fila, uma árvore (binária ou não binária) ou então um grafo, sendo que para cada uma destas estruturas existem uma infinidade de algoritmos conhecidos de busca. A partir das características de uma fila, de uma árvore e de um grafo é possível montar uma hierarquia de classes como a apresentada na figura 2.

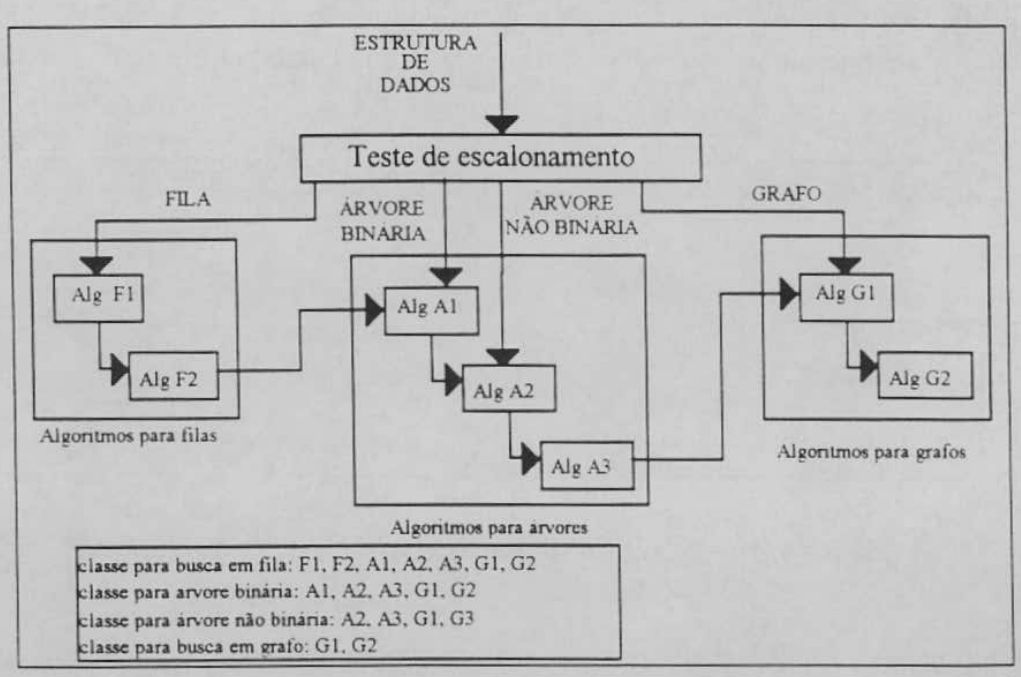

Figura 2: Hierarquia de especialização de algoritmos de busca em estruturas de dados

No modelo de implementação de replicação considerado neste texto (NMR), teremos replicações ativas do mesmo algoritmo escalonadas no sistema distribuído pelo dado de entrada em cada ciclo de processamento. 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. Em hipóteses de faltas arbitrárias, por

exemplo, são necessários (N/2+1) resultados iguais submetidos ao votador (votação majoritária). As premissas de faltas assumidas no modelo de implementação podem simplificar o votador ou simplesmente esvaziá-lo no seu significado. Por exemplo, em faltas de "crash" ou omissão, basta receber um dos resultados das N réplicas, e teremos o resultado correto do ciclo de processamento. No caso de faltas de temporização mecanismos de "watch-dog" (disponivel no teste de aceitação ou mesmo no escalonador), que na verificação do não atendimento das caracteristicas temporais do algoritmo preferencial, pode interromper o ciclo, ativando um outro algoritmo menos preciso, mas em condição ainda de cumprir as restrições temporais da aplicação (computação imprecisa).

A figura 3 ilustra o esquema MR, com o teste de escalonamento, as diferentes replicações e ainda os mecanismos de detecção de erro, na abordagem de implementação adotada aqui. Esta figura especifica a replicação P_B^N como sendo escalonada a partir do dado de entrada.

Neste modelo de implementação existe a necessidade de comunicação em grupo ("broadcast confiável"), na implementação distribuída do votador ou ainda para garantir o determinismo de estado das réplicas a partir da entrada de dados [Schneider 90]. Uma discussão mais ampla de modelos de implementação e das necessidades de comunicação do MR pode ser encontrado em [Nacamura 92]. Neste texto nos limitamos a apresentar um estudo comparativo do modelo MR com esquemas normalmente presentes na literatura, usando medidas quantitativas, no sentido de mostrar a eficiência do esquema introduzido.

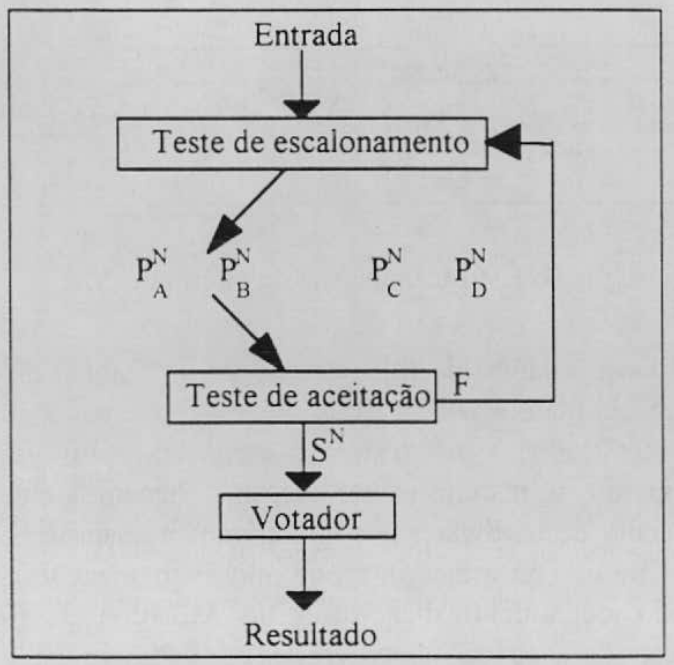

Figura 3: Implementação do MR segundo a abordagem de réplicas ativas e não privilegiadas

4. Tipos de erros do modelo MR

Num ambiente distribuído, onde as réplicas são executadas em várias estações independentes, além das faltas de projeto, existem a possibilidade de faltas transitórias e permanentes de hardware. Os sintomas de cada uma destas faltas no grupo de réplicas são distintos:

- as faltas de projeto (ou de software) se manifestam, quando ativadas, em todas as réplicas do grupo;
- as faltas de hardware (permanentes ou transitórias) podem se manifestar, a priori, em subconjuntos de réplicas do grupo.

Consequentemente, no caso específico de faltas de hardware, o número de réplicas (estações) atingidas é que determina se o grupo continua a fornecer o serviço de maneira satisfatória, ou seja, com base nos mecanismos de detecção e mascaramento de erro as faltas de hardware (assumindo comportamento arbitrário) são mascaradas pelo votador, desde que não envolvam $(N/2+1)$ réplicas. Neste texto, em relação à faltas de hardware nos limitaremos as faltas transitórias por serem as de mais difícil trato.

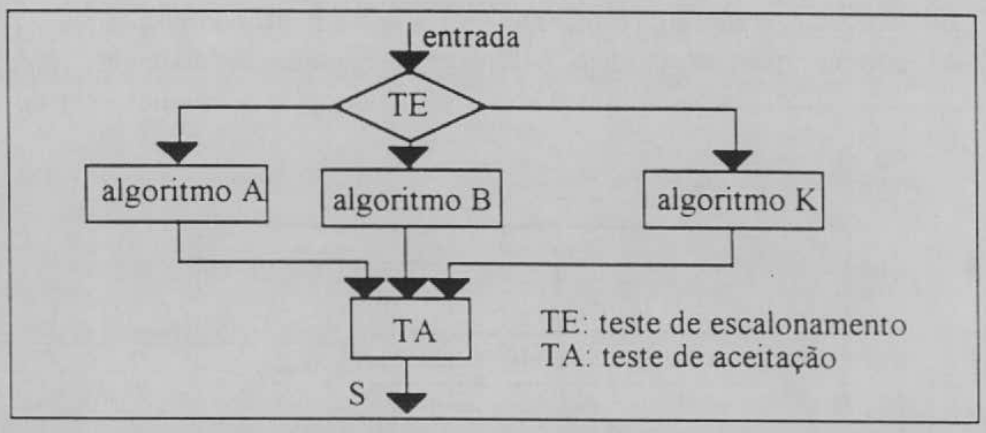

Figura 4: Estrutura interna de um nó do MR

As faltas de projeto, quando da estruturação de uma aplicação na forma de um esquema MR, podem se manifestar no teste de escalonamento, nos algoritmos de aplicação (base das replicações), no teste de aceitação e no votador. As faltas transitórias embora possam se manifestar nos mesmos elementos citados para faltas de projeto, por ser uma falta de hardware, nos limitaremos a analisar as consequências da ocorrência em termos de nó (ou estação), e portanto as implicações se reduzirão só no votador. No sentido de simplificar a visão do MR consideraremos o teste de escalonamento e o teste de aceitação incorporados em cada réplica. A figura 4 mostra cada componente (nó) do MR na estruturação de suas funcionalidades.

ERROS	DESCRIÇÃO
TIPO 1	o teste de escalonamento selecionou uma lista de algoritmos contendo elementos que não são apropriados para a classe do dado de entrada, e um destes algoritmos foi executado (erro detectado no teste de aceitação);
TIPO 2	o algoritmo de aplicação gera um resultado incorreto e o teste de aceitação julga o resultado como correto (erro não detectado);
TIPO 3	o algoritmo de aplicação correspondente a última alternativa da classe gera um resultado correto e o teste de aceitação classifica o resultado como incorreto (erro detectado);
TIPO 4	o algoritmo de aplicação gera um resultado incorreto e o teste de aceitação julga o resultado como incorreto (erro detectado);
TIPO 5	ocorre quando o procedimento de restauração não pode recuperar o estado do MR (estado de entrada) após uma execução de um algoritmo mal escalonado ou incorreto;
TIPO 6	o votador não fornece um resultado de consenso, pois não existe maioria (erro detectado)
TIPO 7	o votador fornece um resultado de consenso, porém este resultado é incorreto (erro não detectado)
TIPO 8	ocorre quando é ativada falta de projeto no procedimento de votação. O erro produzido pode ser resultado da determinação de um grupo de votação incorreto ou então da escolha de um resultado a partir de um grupo incorreto para o voto majoritário.

Tabela 1: Tipos de erros no esquema MR

Com base nas descrições acima, nós identificamos na tabela 1 os tipos de erros de uma estrutura MR. O tipo 1 está relacionado com faltas no teste de escalonamento. Os tipos 2 e 3 estão associados a faltas de projeto no teste de aceitação. O tipo 4 é resultado de faltas de projeto no algoritmo. O tipo 5 implica em faltas de projeto no suporte de execução. O tipo 6 está relacionado com faltas transitórias dos nós onde as réplicas são executadas. O tipo 7 é resultado da combinação de faltas de projeto do teste de escalonamento ou algoritmo de aplicação e também do teste de aceitação e finalmente o tipo 8 está relacionado com faltas de projeto do procedimento de votação. Exteriormente ao grupo, os erros visíveis são os de tipo 6, 7 e 8. Estes tipos de faltas podem ser estendidos para os esquemas citados no item 2. A tabela 2 resume a ocorrência dos tipos de erros descritos nos diversos esquemas.

Com base nos tipos de erros acima das tabelas 1 e 2 foi desenvolvido um simulador que permite determinar o número de erros detectados e não detectados de cada um dos esquemas de tolerância a faltas citados neste texto. A importância em determinar o número de erros detectáveis e não detectáveis pelos esquemas está em explicitar a eficiência de cada um destes em situações quantitativas de erros significativas.

5. Simulação

A eficiência dos esquemas de tolerância a faltas depende também do projeto de seus componentes. Em alguns casos, por exemplo, testes em mecanismos de detecção de erro (teste de aceitação) podem garantir que dentro de uma espaço do espectro de

dados os erros gerados são detectados com absoluta precisão, entretanto fora deste espaço a detecção de erros não é tão precisa. O mesmo ocorre para os algoritmos e outros componentes. Desta forma, os esquemas de tolerância a faltas não são infalíveis e o próprio comportamento do esquema varia de acordo com os parâmetros e premissas no projeto do esquema.

	Esquemas de Tolerância a Faltas				
Erros	MR	DRB	NVP	NSCP	CRB
TIPO 1	X				
TIPO 2	X	X		X	X
TIPO 3	X	X		X	X
TIPO 4	X				
TIPO 5	X	X		X	X
TIPO 6	X		X	X	X
TIPO 7	X		X	X	X
TIPO 8	X		X		X

Tabela 2: Tipos de erros nos diferentes esquemas de Tolerância a Faltas

Considerando estes fatos, o processo de simulação desenvolvido permite quantificar o número de erros detectados e não detectados pelos esquemas de tolerância a faltas considerados, levando em conta a confiabilidade de seus componentes. A correção destes componentes será dada por valores obtidos conforme funções associadas (item 5.1).

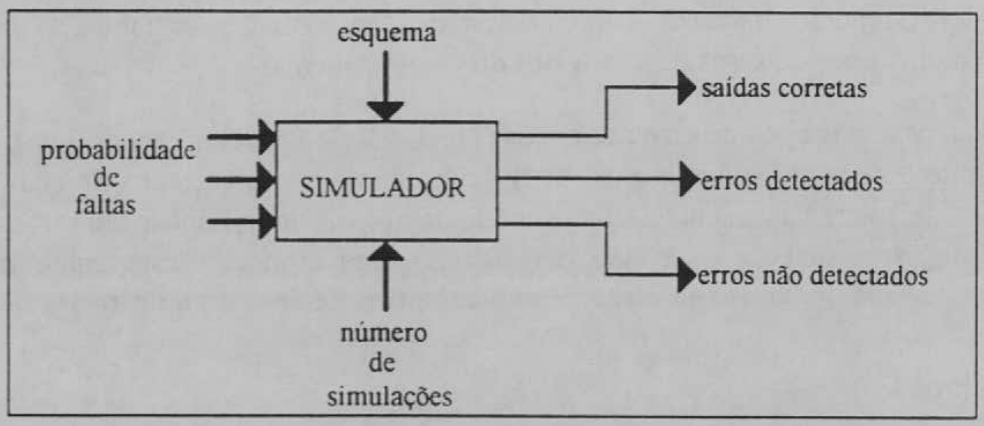

Figura 5: Processo de simulação

O processo de simulação adotado se apresenta esquematizado na figura 5, onde os parâmetros de entrada, que configuram o simulador, determinam o esquema de tolerância a faltas e as probabilidades associadas aos seus componentes que serão usados na simulação. A partir deste ponto, "n_sim" experimentos serão realizados no teste de eficiência do esquema. Em cada experimento, o estado de correção dos componentes do esquema será determinado em geração pseudo-randômica. Nos próximos itens são apresentados respectivamente, a representação do estado de correção dos componentes do MR e a evolução deste estado de correção na dinâmica do processo de simulação. A simulação dos outros esquemas de tolerância a faltas segue a mesma linha de raciocínio.

5.1. Representação do estado de correção do MR

De forma a simular o comportamento do modelo MR, nós introduzimos as seguintes variáveis e vetores:

Teste de escalonamento:

A variável "e" determina a ordenação de algoritmos que são indicados para o processamento de um certo dado de entrada e o valor de "e" indica o início desta ordenação (o algoritmo mais preciso). Para efeito de simulação, o grafo de hierarquia de especialização é representado através de uma lista, com k componentes, sendo k o número total de algoritmos na hierarquia. Consequentemente, quando o teste de escalonamento escolhe o algoritmo de índice j, tal que $1 \leq j \leq k$, os algoritmos de índices $< j$ serão alternativas (ordenadas segundo seus graus de adequação ao dado de entrada) ao algoritmo j quando da falha deste. A variável "d" é introduzida no sentido de sinalizar a correção do teste de escalonamento. Considerando que existe para um dado de entrada um valor ideal de "e", denominado de e_{ideal}, temos que:

$$d = \begin{cases} 0 \text{ para } e \leq e_{ideal} \text{ (a seleção é correta)} \\ 1 \text{ para } e > e_{ideal} \text{ (a seleção é incorreta)} \end{cases}$$

Potencialmente os diferentes nós sobre os quais o esquema MR é configurado, estão sujeitos a faltas transitórias o que pode implicar em testes de escalonamento diferentes, o que faz com que os valores de "e" possam assumir valores diferentes para os nós do MR. Desta forma, o "estado" de escalonamento do MR é dado pelo vetor:

$$TE^G = [d_1, d_2, \ldots d_N]$$

onde d_i representa a correção ou não do escalonamento no nó "i" do MR.

Algoritmo:

O vetor $M = [m_1, m_2, \ldots m_k]$ representa, através de seus componentes m_i, o estado de correção na execução dos algoritmos em um ciclo de processamento de um mesmo nó do esquema MR. O valor de m_i para um ciclo de processamento específico é dado por:

$$m_i = \begin{cases} 0, \text{ o algoritmo executa corretamente para o dado de entrada;} \\ \neq 0, \text{ o algoritmo executa de maneira incorreta para o dado de entrada.} \end{cases}$$

Os algoritmos de um mesmo nó podem assumir diferentes estados de correção, em função dos ciclos de processamento. Isto pode ser explicado, por exemplo, por códigos que apresentam "bugs" (erros de programação) que podem ou não serem ativados nos diferentes ciclos de processamento. O comportamento temporal errado em um ciclo, para aplicações tempo real, também se encaixa neste modelo.

O estado dos algoritmos do MR, em um ciclo de processamento é representado através de N vetores M, ou seja $M^G = \{M_1, M_2, ..., M_N\}$, onde M_i para $1 \leq i \leq N$ representa os estados dos algoritmos do nó de índice i do grupo MR.

teste de aceitação:

O vetor $TA = [a_1, a_2, ...a_n]$ representa, através de seus componentes a_i, o "estado" corrente do teste de aceitação, em ciclo de processamento, de um nó do MR. O valor de a_i é dado por:

$$a_i = \begin{cases} 0, \text{ se o teste de aceitação é correto} \\ 1, \text{ se o teste de aceitação é incorreto} \end{cases}$$

sendo assim, o conjunto de testes de aceitação do MR é representado através de N vetores TA, ou seja $TA^G = \{TA_1, TA_2, ..., TA_N\}$.

5.2. Evolução do estado de correção no processo de simulação

Os parâmetros de entrada do simulador dependem do esquema de tolerância a faltas a ser analisado. No caso específico do MR os parâmetros de entrada são:

- $P(F_{ALG})$: probabilidade de faltas nos algoritmos
- $P(F_{TE})$: probabilidade de ocorrência de faltas no teste de escalonamento;
- $P(F_{TA})$: probabilidade de faltas no teste de aceitação;
- $P(F_{hard})$: probabilidade de faltas de hardware (faltas transitórias);
- $P(F_{VOT})$: probabilidade de faltas no votador

As faltas transitórias, como citado anteriormente, podem se manifestar no teste de escalonamento, no algoritmo ou no teste de aceitação. No sentido de simplificar, limitaremos os efeitos destas faltas ao nível de estações (nós). Outros parâmetros do simulador no caso do MR são:

- "N": número de nós do MR;
- "k": número total de algoritmos;

- "x": número total de resultados que um algoritmo pode produzir. Por exemplo, para funções booleanas "x" é igual a 2;
- "n_sim" número de simulações desejado.

Estes parâmetros são importantes na geração de valores aleatórios que são determinantes em cada experimento de simulação. Estes valores são gerados a partir das seguintes funções:

- **rand (b):** esta função retorna, com probabilidade uniforme, um número aleatório entre 1 e b;
- **randômico(b):** esta função retorna "0" com probabilidade **b** e "1" com probabilidade **b-1**;
- **distrib(b, p_1, p_2):** se b igual a "0" esta função retorna o parâmetro p_1, senão retorna p_2. Normalmente os parâmetros p_1 e p_2 são funções.

As expressões que a partir dos parâmetros e das funções citados acima, determinam a cada experimento o estado dos componentes do MR são dadas na tabela 3.

	nó	grupo
teste de escalonamento	$e = \text{distrib}(1 - P(F_{TE}), e_{ideal}, \text{rand}(k))$;	p/ $0 \le i \le N-1$ $d_i = \text{distrib}(1 - P(F_{hard}), d, \text{rand}(k))$
Algoritmo	p/ $0 \le j \le k-1$ $m_j = \text{randômico}\ (1 - P(F_{ALG\,j}))$	p/ $0 \le i \le N-1$ e $0 \le j \le k-1$ $m_j^i = \text{distrib}(1 - P(F_{hard}), m_j, \text{randômico}(x))$
teste de aceitação	p/ $0 \le j \le k-1$ $a_j = \text{randômico}(1 - P(F_{TA}))$	p/ $0 \le i \le N-1$ e $0 \le j \le k-1$ $a_j^i = \text{distrib}(1 - P(F_{hard}), a_j, \text{randômico}(1))$
votação	----------	$v = \text{randômico}\ (1 - P(F_{VOT}))$

Tabela 3 : cálculo das correções dos componentes do MR

5.3. A estrutura do programa de simulação

Uma vez definido o esquema de tolerância a faltas a ser simulado e os parâmetros associados, o programa de simulação executa "n_sim" simulações, equivalente na prática a "n_sim" ciclos de processamentos. Em cada simulação são atribuídos valores às variáveis definidas anteriormente. Analisando estes valores é possível identificar as condições associadas aos erros definidos no item 4 deste texto. Na figura 6 é apresentada a estrutura simplificada do programa de simulação do esquema MR. Para não sobrecarregar em demasia a figura, nós representamos apenas a estrutura para o esquema MR, embora o mesmo programa simule os outros esquemas também.

```
Programa: simulador de esquemas de tolerância a faltas
Parâmetros de entrada: esquema, P(F_TE), P(F_k),
                       P(F_TA), P(F_hard), N, n_sim, k, x
Parâmetros de saída: número de erros det. e não-det.
inicio {
   caso:
   esquema == MR
           mr();
      esquema == DRB
   drb();
    :
   esquema == PNV
   pnv();
}
mr() {
    para i:=0 até n_sim {
      para i:=0 até k-1 && j:=0 até N-1 {
        gera os valores de e[i], m[i,j], a[i,j];
        se e[i] > e_ideal então
                  r[i]:=exceção;
        senão se m[i,j]==a[i,j] então
                  r[i]=ok;
        senão se m[i,j] != 0 && a[i,j]==1 então
                  r[i]:= m[i,j]      /* erro não detectado */
        senão r[i]=erro_det;
      }
      encontra o valor de maior frequência entre os r[i]
      se a maioria é ok então
                  resul_corretos++;
      senão se maioria é erro_det então
                  erro_det++;
            senão erro_n_det++;
    }
    imprime os valores: result_corretos, erro_det e
erro_n_det;
}
```

Figura 6: Estrutura do programa de simulação

5.4 Resultados dos experimentos

Os experimentos realizados tiveram por objetivo fornecer uma base quantitativa de comparação e de análise dos esquemas de tolerância a faltas, apresentados no item 2, com o modelo MR proposto no artigo. Em cada simulação, foram realizados 1 milhão de experimentos, número que consideramos satisfatório no sentido de determinar uma tendência.

A tabela 4 apresenta os resultados considerando as probabilidades de falta de projeto no votador e no teste de aceitação com valor atribuido de 0.005 e a

probabilidade de faltas transitórias com valor de 0.0001. Estes valores atribuídos para faltas de projeto estão dentro da faixa dita de baixa confiabilidade (pior caso), conforme especificado em [Butler 93] no que se refere a probabilidade de componentes de software. O valor de 0.00001 assumido para faltas transitórias determina uma situação semelhante de probabilidade de faltas àquela apresentada em [Kopetz 90].

Prob. de faltas nos algoritmos	Esquemas de Tolerância a faltas										
	NMR	PNV		CRB		DRB		NSCP		MR	
	n_det	det	n_det	det	n_det	det	n_det	det	n_det	det	n_det
0.01	10041	296	62	0	39	4	61	95	4	5	14
0.02	19952	1073	123	2	112	15	99	385	40	3	39
0.03	29836	2428	287	13	274	40	139	860	88	3	80
0.04	39847	4235	477	28	443	89	206	1478	178	5	102
0.05	49737	6615	700	55	667	157	291	2256	256	3	108
0.06	59717	9157	1039	120	982	263	340	3250	343	2	125
0.07	69560	12434	1464	177	1303	434	401	4379	466	2	139
0.08	79513	16387	1882	261	1772	615	453	5753	605	2	162
0.09	89510	20350	2457	400	2363	824	521	7422	796	3	189
0.1	99434	24907	2942	504	2876	1176	529	9222	1000	5	203

Prob. de faltas de projeto (teste de aceitação e votador): 0.005

Prob. de faltas transitórias: 0.00001

Tabela 4: Resultados dos experimentos

Dos resultados obtidos pode-se verificar facilmente que o modelo MR apresentou um desempenho melhor em termos de erros não detectados. O esquema NSCP apresentou valores semelhantes de erros não detectados para baixas probabilidades de faltas nos algoritmos. Com base nestes resultados nós concluímos que o número de erros não detectados diminui nos esquemas considerados quando os resultados gerados pelos algoritmos de aplicação são submetidos a um teste de consistência, no caso o teste de aceitação. Os resultados de erros não detectados do DRB e CRB confirmam esta observação.

O melhor desempenho do MR em relação ao esquema CRB que contém os mesmos mecanismos de detecção de erro (teste de aceitação e voto) pode ser explicado na colocação do teste de aceitação. No MR o teste de aceitação é realizado na saída dos algoritmos, o que minimiza o número de resultados errados que chegam na entrada do votador, onde o mecanismo de voto realizado é o de votação exata (mais simples). No CRB os valores de saída dos algoritmos são aplicados diretamente no votador, que se utiliza de uma votação inexata que pode chegar a um valor de consenso errado a partir de saídas erradas, mesmo não sendo majoritárias [Lorczak 89]. O teste de aceitação no CRB só é realizado quando não é possível obter um valor de consenso.

Os resultados do MR em relação aos outros se explica pelo fato deste esquema apresentar dois mecanismos de detecção de erros. O NSCP se apresenta com melhores resultados que o DRB porque o seu teste de aceitação corresponde a uma comparação de resultados. Estes (NSCP e DRB) apresentam melhores resultados que o PVN e o CRB pelas características citadas acima de votação inexata. Esta particularidade da votação inexata faz com que o CRB, apresente valores próximos do PVN em relação a erros não detectados. Neste dois últimos esquemas a diferença de erros detectados se explica pelo teste de aceitação presente no CRB.

Nos erros detectados considerados no processo de simulação não estão incluídos erros mascarados, ou seja, as saídas errôneas dos algoritmos que quando submetidas a uma votação, por serem minoritárias, são mascaradas. Muito embora a presença de erros minoritários (mascarados: não visíveis externamente), uma votação onde ocorre consenso corresponde sempre a uma saída correta para um observador externo. A não inclusão de erros mascarados em erros detectados explica na tabela 4 porque esquemas com votador (MR, CRB, PVN) apresentam valores na coluna *det* menores que os da coluna *n_det*.

Os valores de erros detectados são extremamente baixos devido ao uso da votação exata que detecta apenas erros decorrentes de faltas transitórias. Os valores baixos desta coluna estão em sintonia com o valor de probabilidade assumido para estas faltas (0.00001). Por fim, para o modelo NMR não é apresentada coluna de erros detectados pelo mesmo motivo, ou seja, este esquema apresenta votação exata e portanto detecta apenas faltas transitórias e os valores de erros detectados são extremamente baixos. A coluna de erros não detectados do NMR mostra o que é óbvio: a inutilidade do mesmo para faltas de projeto. A variação do número de erros detectados e não detectados pode ser melhor observada nas figuras 7 e 8.

6. Outras Considerações

O esquema MR foi implementado no LCMI/DEEL/UFSC (Laboratório de Controle e MicroInformática da Universidade Federal de Santa Catarina) usando uma plataforma UNIX, em uma rede local com 6 estações SUN. O modelo de implementação baseou-se em réplicas ativas e não privilegiadas. Nesta implementação foram usados os protocolos de difusão de mensagens CBCAST e ABCAST [Birman 89], respectivamente, nas comunicações cliente/grupo MR e entre réplicas no grupo MR na implementação do votador. A hierarquia de especificação foi constituída de algoritmos de busca conforme indicado na figura 2. Os testes realizados nesta implementação comprovam a eficiência já indicada pela simulação do item 5.

O desempenho do esquema MR poderia ser contestado por fazer uso de técnica de recuperação de erro em retrocesso ("backward recovery"), ou seja se os resultados de uma replicação não passam pelo teste de aceitação o estado inicial do MR no ciclo é restaurado e uma nova replicação é ativada. O que nós podemos dizer é que o teste de escalonamento minimiza a ocorrência destes retrocessos uma vez que seleciona os algoritmos mais adequados para cada dado de entrada. Isto não ocorre em esquemas

como o RB, DRB e CRB, onde a hierarquia é sempre a mesma e sujeita a qualquer dado de entrada.

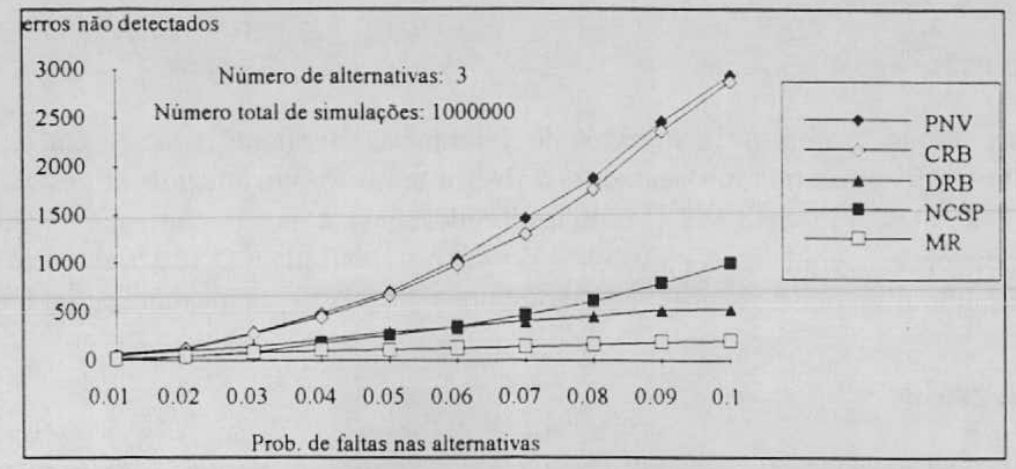

Figura 7: Resultados relativos a tabela 4 (erros não detectados)

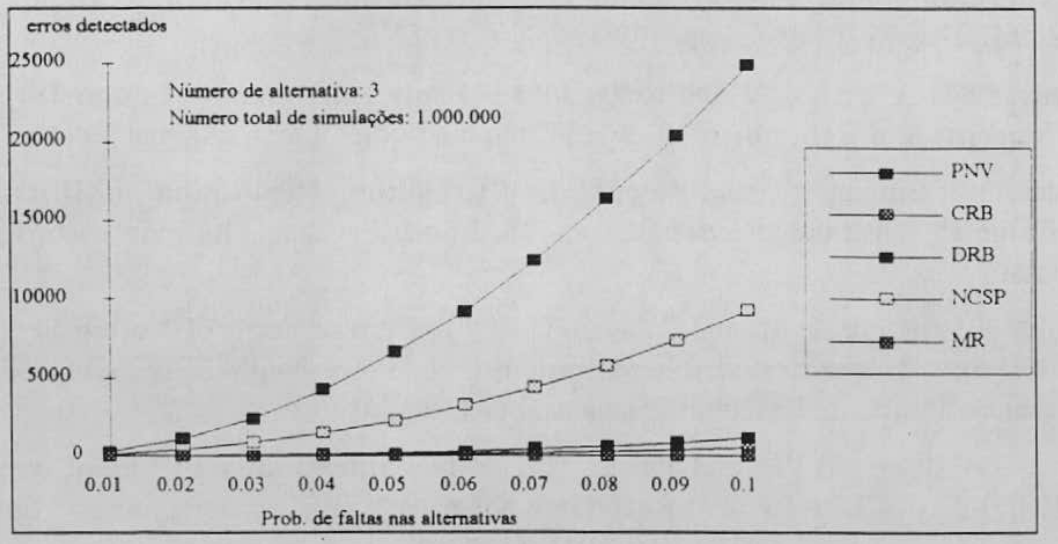

Figura 8: Resultados relativos a tabela 4 (erros detectados)

O esquema MR incluído dentro de uma noção de grupo está sendo usado para expressar programas distribuídos com alta confiabilidade. No modelo de programação que estamos adotando, um sistema distribuído (programa distribuído) é formado por grupos MR: clientes e servidores são expressados em grupos MR. Este trabalho se insere na linha do apresentado em [Tully 90], onde processos do programa distribuído são colocados na forma de "Triades" ("Triple Modular Redundancy" - TMR).

O fator relevante do esquema MR é a flexibilidade da seleção de redundâncias adequadas para tratar cada dado de entrada. Isto permite que se construam servidores confiáveis, implementando determinadas funções, porém atendendo faixas mais amplas

de necessidades de processamento. Um exemplo disto é o servidor que realiza buscas (figura 2). Isto nos parece perfeitamente adequado à natureza de um sistema distribuído. Os outros esquemas não apresentam tal flexibilidade.

7. Conclusão

Este artigo tratou de esquemas de tolerâncias de faltas usados em sistemas distribuídos e sistemas multiprocessadores. Neste texto foi introduzido um esquema de tolerância a faltas, o modelo MR (Múltiplas Replicações), cuja característica é a ativação de redundâncias adequadas aos atributos do dado de entrada. O texto apresentou um estudo comparativo onde os resultados são obtidos através de uma simulação numérica.

8. Bibliografia

[Anderson 81] Anderson, T. and Lee, P.A., **"Fault Tolerance - Principles and Practice"**, Prentice-Hall, 1981.

[Avizienis 77] Avizienis, T. and Chen, L., **"On the Implementation of N-Version Programming for Software Fault-Tolerance During Program Execution"**, Proc. COMPSAC 77, Chigaco, pp. 149-155, Nov, 1977.

[Avizienis 84] Avizienis, A. and Kelly, J.P.J., **"Fault Tolerance by Design Diversity: Concepts and Experiments"**, IEEE Computer, vol. 17, nº 8, August 1984, 67-80.

[Birman 89] Birman, K. and Joseph, T., **"Exploiting Replication in Distributed Systems"**, Distributed Systems, Cap. 15, Edited by Sape Mullender, ACM Press, pg. 319-367.

[Blough 90] Blough, D.M. and Sullivan, G.F., **"A Comparation of Voting Strategies for Fault Tolerant Distributed Systems"**. In Proc. Ninth IEEE Symposium on Reliable Distributed Systems, Alabama, pg. 136-145.

[Butler 93] Butler, R.W., and Finelli, G.B., **"The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software"**, IEEE Transactions on Software Engineering, Vol. 19, N° 1, Jan. 1993, pp. 3-12.

[DELTA_4 90] **"DELTA_4 Architecture Guide"**, Delta-4 Consortium, December 1990. (Delta-4 Document N° G90.050/I1/R).

[Giandomenico 90] Giandomenico, F.L. and Strigini, **"Adjudicators for Diversity Redundant Components"**, in Proc. Ninth IEEE Symposium on Reliable Distributed Systems, Alabama, pg. 114-123.

[Kim 88] Kim, K.H., **"Approaches to Implementation of a Repairable Distributed Recovery Block Scheme"**, 18 th Internation Symposium of Fault-Tolerant Computing, Pittsburgh, June 1988.

[Kopetz 90] Kopetz, H., Kantz, H., Grunsteidl, G., Puschner, P., Reisinger, J., **"Tolerating Transient Faults in Mars"**, 20th Symposium on Fault Tolerant Computing, Newcastle upon Tyne, UK, June 1990.

[Laprie 87] Laprie, J.C., Arlat, J., Beounes, C., Kanoun, K., Hourtolle, C., **"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] Laprie, J.C., Beounes, C., Karnoun, K., **"Definition and Analysis of Hardware-and-Software Fault-Tolerant Architectures"**, PDCS Esprit Basic Research Action, 23, May, 1990.

[Lorczak 89] Lorczak, P.R., Caglayan, A.K. and Eckhardt, D.E., **"A Theoretical investigatin of Generalized Voters for Redundant Systems"**, in Proc 19th FTCS, Chicago, Illinois, June 1989, pg. 444-451.

[Mori 86] Mori, K., et. al., **"Autonomous Decentralized Sotware Structure and Its Application"**, Proc. Fall Joint Comp. Conf., Dallas, Texas, Nov. 1986, pp. 1056-1063.

[Nacamura 92] Nacamura Júnior, L., **"Proposição de um Modelo de Tolerância a Faltas baseado em alta replicação"**, Monografia submetida ao exame de qualificação para doutorado em Engenharia Elétrica, UFSC, Brasil, Abril 1992.

[Randell 75] Randell, B., **"System Structure for Software Fault Tolerance"**, IEEE Transaction on Software Engineering, June 1975, pg. 220-232.

[Schneider 90] Schneider, F.B., **"Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial"**, ACM Computing Surveys, Vol. 22, N° 4, Dec. 1990, pp. 299-319.

[Scoth 87] Scoth, P.K., Gault, J.W. and McAllister, D.F., **"Fault-Tolerance Software Reliability Modeling"**, IEEE Transactions on Software Engineering, Vol. SE-13, Nº 5, May 1987.

[Shin 89] Shin, K.K and Dolter, J.W., **"Alternative Majority-Voting Methods for Real-Time Computing Systems"**, IEEE Transactions on Reliability, Vol. 38, Nº1, April 1989, pg. 58-67.

[Tully 90] Tully, A., **"Preventing State Divergence in Replicated Distributed Systems"**, Ph.D. Thesis, The Univesity of Newcastle upon Tyne Computing Laboratory, Sep. 1990

[Wensley 78] Wensley, J.H., et al., **"SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control"**, Proc. IEEE, vol. 60, pp. 1240-1245, Oct, 1978.

[Yu 91] Yu, A.C and Lin, K.-J, **"Recovery Manager for Real-Time Imprecise Computations"**, RTSS Workshop on Architectural Aspects of Real-Time Systems, Dec. 1991.