DSEE Universidade Estadual de Campinas
FACULDADE DE ENGENHARIA ELÉTRICA
DEPARTAMENTO DE SISTEMAS DE ENERGIA ELÉTRICA

UNICAMP

Análise *On-line* da Estabilidade Transitória de Sistemas Elétricos de Potência usando Ambientes de Computação Distribuída

Tese apresentada à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas, como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica.

por

Marcelo Stehling de Castro

Graduado em Engenharia Elétrica - FE/UFJF

em outubro de 1995 perante a banca examinadora

André Luiz Morelato França	Orientador
Ildemar Decker	UFSC
Fujio Sato	FEE/UNICAMP

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

C279a

Castro, Marcelo Stehling de
Análise on-line da estabilidade transitória de sistemas elétricos de potência usando ambientes de computação distribuida / Marcelo Stehling de Castro.--Campinas, SP: [s.n.], 1995.

Orientador: André Luiz Morelato França.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica.

1. Processamento paralelo (Computadores). 2. Sistemas de energia elétrica - Estabilidade. 3. Sistemas de energia elétrica - Estimação de estado. I. França, André Luiz Morelato. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica. III. Título.

Resumo

O objetivo desta dissertação é investigar a viabilidade de se utilizar ambientes de computação distribuída para avaliar de forma *on-line* o nível de segurança dinâmica de sistemas elétricos de potência.

No primeiro capítulo são apresentados alguns conceitos básicos sobre análise e determinação da segurança dinâmica em sistemas de energia elétrica bem como uma revisão bibliográfica de trabalhos e publicações relacionados com a área.

No segundo capítulo é descrito o problema abordado bem como são apresentados os métodos de solução utilizados para efetuar a avaliação da segurança. A partir de uma lista de contingências dinâmicas, calcula-se a estabilidade transitória relativa a cada contingência usando-se simulação no domínio do tempo. Margens de estabilidade ou instabilidade são obtidas aplicando-se técnicas baseadas na teoria da função transitória da energia.

O terceiro capítulo apresenta os ambientes computacionais distribuídos utilizados para avaliar o desempenho da implementação da análise *on-line.*

O quarto capítulo descreve a implementação do método de análise *on-line* da estabilidade transitória em ambientes distribuídos utilizando o *software* **PVM** para gerenciar a comunicação e a coordenação dos processos entre as diversas máquinas do ambiente.

No quinto capítulo são apresentados e comentados, para diversos ambientes distribuídos, os resultados de desempenho obtidos utilizando-se uma rede elétrica real Sul-Sudeste com 320 barras e 46 geradores. Os resultados mostram que é possível analisar de 100 a 200 contingências, na rede acima, usando modelo detalhado para geradores em cerca de 5 minutos.

As conclusões contidas no sexto capítulo indicam que é viável a solução do problema usando sistemas homogêneos de computação distribuída, baseados na tecnologia **RISC** atual e *softwares* tipo **PVM** para gerenciar os processos.

Abstract

In this thesis a network of several computers, working as a virtual parallel machine, is used to perform on-line power systems transient stability assessment based on time-domain simulation over a list of contingencies.

The whole distributed computing system is operated and controlled by **PVM** (Parallel Virtual Machine) software.

Performance test results are presented using a real-world power system database with 320 buses and 46 generators, showing that from 100 to 200 contingencies can be analyzed in about 5 minutes, depending on the distributed environment.

From the implementation and test results, the following conclusions can be drawn: on-line transient stability assessment is feasible when implemented on homogeneous distributed computing environments and PVM package shows promise to control efficiently distributed environments.

Agradecimentos

- Prof. André Luiz Morelato França pela excelente orientação, paciência, amizade, ensinamentos técnicos e científicos ao longo deste período no qual trabalhamos juntos.
- Prof. Ariovaldo Verandio Garcia pelo suporte técnico prestado, paciência, amizade, ao longo destes anos.
- Aos amigos do Laboratório de Sistemas de Energia Elétrica/DSEE, pela convivência prestativa e solidária, mas acima de tudo pelo carinho com que sempre me trataram;
- Professores do Departamento de Sistemas de Energia Elétrica - UNICAMP, pela amizade, colaboração e as excelentes condições de trabalho que fornecem aos estudantes de pós-graduação;
- A todos que contribuíram para que a realização deste trabalho se tornasse possível;
- A equipe de suporte do CENAPAD pelo auxílio na implementação do programa nas máquinas IBM;
- CAPES pelo apoio financeiro;
- Pesquisa desenvolvida com o auxílio do CENAPAD-SP (Programa Centro Nacional de Processamento de Alto Desempenho em São Paulo), projeto UNICAMP / FINEP - MCT.

Há homens que lutam um dia e são bons;

Há outros que lutam um ano e são melhores;

Há aqueles que lutam muitos anos e são muito bons;

Más há os que lutam toda a vida, esses são imprescindíveis.

"Berthold Brecht"

Dedico este trabalho aos meus pais Paulo e Myrthes,

meus irmãos Paulo e Heloísa,

minha namorada Welma e meus amigos.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Análise On-Line de Segurança Dinâmica

1.1 Introdução

A operação segura e econômica de um sistema elétrico de potência é permanentemente um objetivo a ser alcançado. Este objetivo não é fácil de ser atingido, tal a magnitude e a complexidade do problema, requerendo o uso de avançadas técnicas de análise, otimização e controle, e a disponibilidade de computadores de alto desempenho. Não obstante, os centros de controle das empresas de energia elétrica (**EMS** - *Energy Management System*) vêm evoluindo ao longo do tempo e incorporando melhores recursos computacionais que têm permitido a implantação de novas funções de monitoramento, análise e controle. Uma dessas funções é a análise de segurança dinâmica.

Um sistema de energia elétrica está operando de forma "segura" quando é capaz de atender à carga (objetivo primordial) sem violar limites operativos pré-estabelecidos, mesmo se submetido a contingências imprevisíveis. Portanto, o conceito de segurança está intimamente ligado à idéia de sobrevivência após uma perturbação significativa. Chama-se de "análise de segurança" ao conjunto de procedimentos realizados para determinar se, e em que grau, um sistema elétrico de potência é seguro em relação a uma lista de contingências. Se pelo menos uma contingência acarretar violação em limites operativos, o sistema será considerado inseguro. Somente tem sentido analisar a segurança de sistemas que estejam no "estado normal", ou seja, toda a carga está atendida e nenhum equipamento

está sobrecarregado. Portanto, a primeira tarefa de um processo de análise de segurança é identificar se o sistema está em estado normal ou de emergência (monitoramento de segurança). Se o sistema estiver no estado normal, a análise de segurança determina se o sistema está "seguro" ou "inseguro" com respeito a um conjunto de contingências possíveis. Se estiver "inseguro", ou seja, se existir pelo menos uma contingência que possa causar violações, devem ser encontradas ações de controle preventivas para levar o sistema a um estado seguro (corretivamente seguro).

A análise de segurança dinâmica de um sistema de energia elétrica, composto por uma rede interligada que inclui transmissão e geração, é uma tarefa que exige um grande esforço computacional. Este tende a crescer à medida que aumenta a dimensão do sistema em estudo, o número de interconexões, ou quando se deseja uma operação mais precisa. Construir melhores centros de controle (**EMS**) representa um grande desafio para a indústria associada a geração, transmissão e distribuição de energia elétrica.

Uma meta a ser alcançada é que os resultados dos cálculos de segurança dinâmica sejam implementados automaticamente pelo **EMS** sem que haja intervenção do operador humano dos sistemas de potência. Entretanto, existem atualmente pouquíssimos exemplos de esquemas de análise com restrições de segurança operando em malha fechada. A maioria das instalações já feitas tem se caracterizado por adotar um modo de operação interativo, nos quais os resultados são apresentados na forma de dados e avisos ao operador humano que os aceita, modifica ou ignora de acordo com a sua experiência em engenharia. O número de implementações totalmente automáticas aumentará naturalmente à medida que a confiança das empresas do setor elétrico na exatidão dos resultados obtidos pela análise de segurança também aumentar.

Sistemas de potência com controle que inclui restrições de economia-segurança ainda é um campo de pesquisa recente no qual muitos dos aspectos práticos bem como os conceituais ainda não foram completamente estabelecidos [10]. Visando dar uma idéia geral da área, neste capítulo apresenta-se um conjunto de conceitos e definições usuais relacionados com a segurança dinâmica; o papel do método da função de energia transitória (**TEF**) no estudo da estabilidade transitória dentro de um sistema de análise de segurança dinâmica *on-line*; uma visão geral das atuais pesquisas e implementações indicadas na literatura existente; e como poderia ser feita a determinação da segurança dinâmica no futuro.

1.2 Análise de Segurança

O **NERC** (*North American Electric Realiabilty Council*) define segurança como "prevenção de desligamentos em cascata quando o sistema de transmissão está submetido a perturbações severas" [20]. Os desligamentos em série não ocorrerão se o sistema foi planejado e está operando de forma tal que as seguintes condições, no sistema de transmissão, são satisfeitas em todo instante:

1. nenhum equipamento ou circuito de transmissão está sobrecarregado;
2. não há barras com tensão fora dos limites (usualmente dentro de $\pm 5\,\%$ do valor nominal);
3. se após a ocorrência de alguma perturbação, dentro de um conjunto previamente especificado, resultarem condições de operação em regime permanente aceitáveis.

A análise de segurança é conduzida para verificar se as condições acima são satisfeitas. As duas primeiras condições precisam apenas de uma análise em regime permanente, a terceira requer análise transitória (programa computacional de estabilidade transitória). Recentemente descobriu-se que alguns dos fenômenos de instabilidade de tensão são dinâmicos em natureza e precisam de novas ferramentas de análise.

Em geral a análise de segurança está relacionada com a resposta do sistema de potência às perturbações. Na análise em regime permanente é suposta uma nova condição de operação. Na análise de segurança dinâmica a própria transição em si é de interesse, permitindo que a análise confirme se a transição levará a uma condição de operação aceitável ou não.

Atualmente a capacidade computacional dos centros de controle tem limitado a análise de segurança a cálculos em regime permanente, nos quais as condições de regime pós-contingência são obtidas e usadas para verificar os limites de fluxos ou tensões frente a possíveis violações. Isso significa que a dinâmica do sistema é ignorada, não permitindo saber se o estado de pós-contingência seria alcançado sem perda de sincronismo em parte do sistema. Ainda que o estado de regime permanente pós-contingência fosse obtido corretamente, é preciso saber o modo real de instabilidade para determinar-se alguma ação remediadora. Dessa forma, fica clara a necessidade de uma análise dinâmica.

1.3 Análise de Segurança Dinâmica

A análise de segurança tem sido classificada em "estática" e "dinâmica". Na análise de segurança estática as restrições operativas que se deseja monitorar referem-se à sobrecarga em equipamentos (transformadores e linhas de transmissão) e verificação de limites de tensão em barramentos. Supõe-se que o sistema elétrico atinja um novo ponto de operação em regime permanente, após submetido a uma perturbação (desligamento de linha, por exemplo). A ferramenta computacional básica utilizada na análise estática é um programa de fluxo de carga. Inúmeras técnicas tem sido usadas para selecionar as contingências mais severas e realizar a análise de segurança propriamente dita reduzindo o número de fluxos de carga necessários. Quando a análise de segurança é estendida de tal forma a incluir a otimização de uma função objetivo (custo de operação, por exemplo), então se deve resolver um problema de fluxo de carga ótimo.

Na análise de segurança dinâmica o que se deseja saber primordialmente é se o sistema elétrico irá atingir um ponto de operação que seja aceitável, após a ocorrência de uma contingência (desligamento de linha, eliminando um curto-circuito, por exemplo). Analisa-se o transitório eletromecânico sofrido pelos geradores a fim de verificar se o sincronismo tende a ser mantido, ou ocorrerá uma instabilidade que conduzirá o sistema ao colapso. Um programa de estabilidade transitória é a ferramenta computacional básica utilizada na análise de segurança dinâmica, durante a qual precisa ser repetidamente rodado. Em geral, as restrições operativas de origem dinâmica são expressas em termos de limites de transferência de potência em determinado grupo de linhas de transmissão ou conjunto de usinas geradoras. Isto significa que se esses limites forem ultrapassados durante a operação pelo menos uma contingência poderá, se ocorrer, levar o sistema à instabilidade. Trata-se portanto de uma ação preventiva. Por outro lado, ao ser detectado um caso instável, devido a uma certa contingência, pode-se implantar uma ação corretiva que consiste usualmente em reduzir automaticamente a potência ativa de certos geradores (redespacho de geração), se a contingência ocorrer. Como a quantidade de geração a ser cortada e o local depende de cada perturbação, o resultado é uma tabela de procedimentos colocada a disposição do operador do sistema. Esta tabela permite ao operador ajustar os mecanismos automáticos de corte de geração em função de cada contingência.

Os estudos de segurança dinâmica dos sistemas elétricos de potência são importantes para o planejamento, operação e controle adequado desses sistemas. Com o crescimento da complexidade e do tamanho dos sistemas, as empresas elétricas necessitam

de ferramentas computacionais de análise de segurança mais poderosas e rápidas. Na atualidade a maior parte das empresas elétricas no mundo realizam análise de segurança dinâmica *off-line* embora a necessidade de se realizar análise *on-line* seja evidente.

1.4 Análise *On-line*

A análise de segurança é classificada como *on-line* quando a determinação do estado e nível de segurança faz parte das tarefas rotineiras de monitoramento, análise e controle executadas pelo centro de controle do sistema. Em outras palavras, a análise de segurança é parte integrante de uma ou mais malhas de controle em tempo real do sistema. Na análise *on-line* as reais condições operativas do sistema podem ser levadas em conta, e os limites operativos e ações de controle podem ser atualizados periodicamente.

Quando a análise de segurança é feita em modo *off-line* opta-se por estudar somente os piores casos, tendo em vista o tamanho da rede elétrica e o grande número de contingências a analisar. Isto leva ao estabelecimento de limites operativos imprecisos e conservativos, válidos somente para situações genéricas de carga leve, média ou pesada, por exemplo. Comumente o operador do sistema depara-se com situações não previamente estudadas e, portanto, as instruções operativas disponíveis serão de pouca valia. Com análise *on-line* a operação do sistema se dá de modo muito mais realista e confiável, pois se baseia nas informações as mais recentes possíveis.

As ações básicas da análise da segurança *on-line* são: monitoramento, avaliação e melhoria da segurança, como descrito abaixo.

1. **Monitoramento da segurança** : Usando medições do sistema em tempo real, identifica se o sistema está em estado normal ou não. Se ele estiver em estado de emergência precisará de um controle de emergência. Se ocorrer perda de carga será necessário um controle restaurativo;
2. **Avaliação da Segurança** : Se o sistema está em estado normal, determinar se o sistema está seguro ou inseguro com respeito a um conjunto de contingências, bem como estabelecer índices ou margens de segurança;
3. **Melhoria da Segurança** : Se inseguro, determinar ações corretivas que devem ser tomadas para que o sistema se torne seguro. Se seguro, determinar as ações preventivas para evitar que o sistema se torne inseguro.

A operação *on-line* segura e econômica de um sistema elétrico de potência é um objetivo que vem sendo almejado há bastante tempo. Os centros de controle das empresas de energia elétrica vêm evoluindo ao longo do tempo e incorporando melhores recursos computacionais que têm aberto o caminho para a implantação *on-line* de novas funções de monitoramento, análise e controle. Atualmente, vários centros de controle realizam análise de segurança estática (análise de contingências) em modo *on-line*. Já a implementação de análise *on-line* de segurança dinâmica está em um estágio bem menos avançado devido a três motivos principais:

a) o problema dinâmico é muito mais complicado, exigindo modelos mais complexos;

b) falta de métodos de análise de segurança que sejam eficientes e confiáveis;

c) capacidade computacional insuficiente dos centros de controle para realizar a análise a tempo.

1.5 Decomposição em Subproblemas

A análise *on-line* de segurança dinâmica pode ser decomposta em vários subproblemas, que interagem entre si na forma mostrada na Figura 1.1.

Na análise *on-line* o objetivo é determinar se um futuro ponto de operação será seguro ou não, partindo-se de medidas em tempo-real do sistema, detecção de configuração e estimação de estado que fornecem o caso básico presente. Em estudos *off-line* os casos básicos são previamente selecionados para representar os cenários de pior caso. Análise de segurança só faz sentido se o sistema estiver no estado normal, ou seja, toda a carga atendida e nenhuma restrição operativa violada.

Previsão de Carga

A análise *on-line* de segurança dinâmica é uma análise prospectiva, isto é, visa determinar se um ponto de operação futuro será estável (sendo portanto um estado operativo seguro), ou instável (levando o estado do sistema a ser classificado como inseguro). Sua finalidade é dar ao operador informações atualizadas sobre limites operativos de transferência e ações preventivas de controle que ajude na tomada de decisões de forma antecipada. Deste modo, fazer previsão de carga e geração torna-se parte essencial na análise.

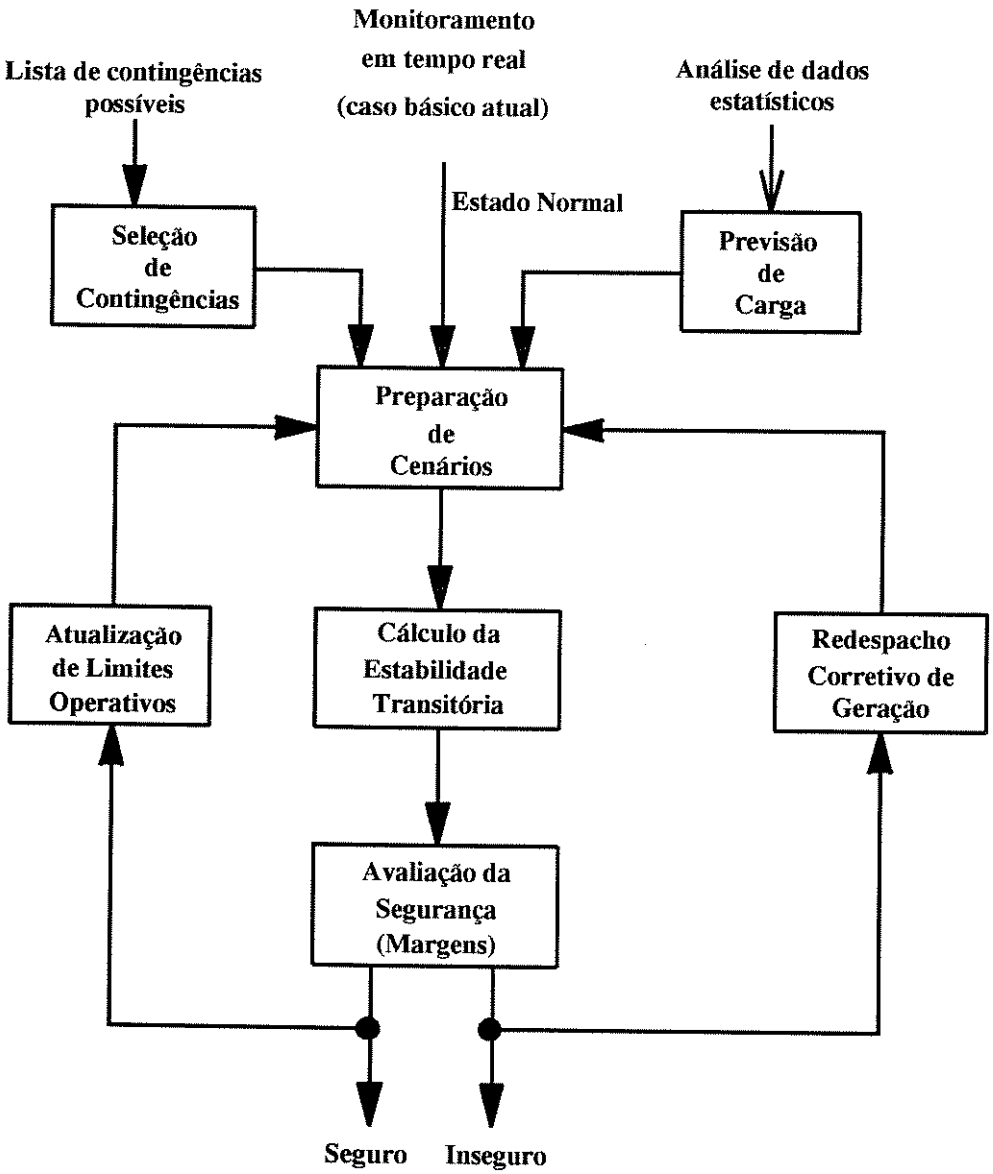

Figura 1.1: Subproblemas da análise *on-line* de segurança dinâmica

Imagina-se aqui um horizonte da previsão de 15 a 30 minutos (previsão de curto prazo), que corresponde ao ciclo em que a análise de segurança deve ser repetida.

Seleção de Contingências

Em sistemas elétricos de grande dimensão o número de contingências possíveis pode facilmente chegar à ordem de milhares. Como não há tempo para analisar todas as contingências em detalhe a solução consiste em filtrar da lista aquelas que comprovadamente (ou muito provavelmente) não ameaçarão a segurança da rede. Esta seleção de contingências deve basear-se em métodos rápidos e aproximados para ser efetiva. Poucos trabalhos têm sido publicados nesta área no caso dinâmico, ao contrário do caso estático onde existem métodos muito eficientes. Neste caso, a aplicação de métodos diretos (métodos que analisam o comportamento de uma determinada função e concluem sobre a estabilidade, sem simulação no tempo) tem se mostrado muito atraente.

Preparação de Cenários

Este módulo recebe informação de vários tipos e cuida de preparar convenientemente as bases de dados que serão utilizadas durante o cálculo da estabilidade transitória. Fornece o caso básico futuro. Como funciona como uma espécie de gerente da análise de segurança pode ser dotado de uma certa dose de inteligência artificial.

Cálculo da Estabilidade Transitória

Este subproblema corresponde ao que normalmente faz um programa de cálculo da estabilidade transitória angular, ou seja, determina se existem geradores instáveis para cada perturbação. Embora qualquer método possa ser usado (direto, híbrido ou simulação) é mais confiável utilizar, como será feito neste trabalho, simulação no domínio do tempo por integração numérica. Desta maneira, evitam-se restrições na modelagem de geradores, seus controles e outros componentes. Por outro lado, a simulação passo-a-passo pode ser muito demorada, levando este módulo a ser excessivamente oneroso em termos computacionais.

Avaliação da Segurança

Aqui os resultados das simulações são analisados e o sistema é classificado como seguro ou inseguro. O sistema (ponto de operação) é considerado inseguro se pelo menos uma contingência puder levar um ou mais geradores à instabilidade. Dentro deste módulo também se inclui a obtenção de margens ou índices de estabilidade ou instabilidade, dependendo da situação. A margem de estabilidade quantifica a distância que um sistema estável está da instabilidade, enquanto que a margem de instabilidade quantifica a distância que um sistema instável está da estabilidade.

Atualização de Limites Operativos

Em geral, as redes elétricas têm limites operativos dinâmicos expressos através do máximo carregamento de um conjunto de linhas de transmissão ou máxima geração de um grupo de usinas (fluxo de interface). Se o sistema estiver dinamicamente seguro pode-se estudar o relaxamento dessas restrições e verificar até onde os limites podem ser relaxados. Freqüentemente isto é realizado através de sucessivos redespachos de geração incrementais em pontos localizados até que um estado inseguro seja alcançado.

Redespacho Corretivo de Geração

Se for detectado que o sistema está dinamicamente inseguro pode-se estudar medidas preventivas ou corretivas para levá-lo ao estado seguro. Uma ação corretiva (atua após a ocorrência de uma contingência) que tem se mostrado efetiva consiste no corte de potência dos geradores que tendem à instabilidade (perda de sincronismo). Portanto, deve-se obter a mínima quantidade de geração a ser reduzida capaz de levar o sistema a um estado seguro. Isto corresponde a realizar redespachos incrementais de geração até se obter um esquema de corte de geração, que deverá ser implantado nos mecanismos automáticos de controle de geração.

1.6 Bibliografia na Área

Como se sabe, a execução *on-line* da análise de segurança dinâmica é um problema complexo e que envolve várias áreas de conhecimento. Este trabalho aborda apenas parte do problema maior, ou seja, investiga a possibilidade de resolver os subproblemas de cálculo da estabilidade transitória e avaliação da segurança utilizando sistemas de computação distribuída. Desse modo, a revisão bibliográfica realizada abrange somente esses temas.

Uma visão geral sobre análise *on-line* de segurança de sistemas de potência pode ser encontrada nas referências [10, 20] . A primeira aborda problemas e técnicas relativas à análise estática, enquanto que a segunda abrange também aspectos da análise dinâmica.

Vários métodos têm sido propostos nos últimos anos para calcular rapidamente a estabilidade transitória e viabilizar análises de segurança *on-line*. Dentre esses destacam-se os métodos diretos (Lyapunov e função da energia transitória), métodos híbridos (misturam simulação e métodos diretos), métodos de reconhecimento de padrões e métodos usando redes neurais artificiais. Nenhum desses métodos rápidos tem se mostrado suficientemente confiável para ser adotado como norma nos centros de controle, como pode ser claramente visto nas referências [16, 17]. Quando se depara com sistemas reais (com sua grande diversidade de equipamentos a modelar e uma seqüência complexa de chaveamentos) o único método confiável ainda é a simulação no domínio do tempo. Por esta razão este método é adotado neste trabalho. Os métodos baseados na função da energia transitória (*transient energy function* - **TEF**) têm apresentado resultados promissores e muita pesquisa continua sendo feita neste campo [22, 8, 12, 13, 31, 18]. Esta teoria tem sido reconhecida como uma ferramenta útil no subproblema de seleção de contingências dinâmicas [4], bem como na obtenção de margens de estabilidade [2]. Algumas empresas têm implantado sistemas de análise *on-line* de segurança dinâmica recentemente [1, 16, 17], cujo desempenho está sendo avaliado experimentalmente.

Capítulo 2

Problema Abordado e Métodos de Solução

2.1 Formulação do Problema

O problema focalizado neste trabalho é a implementação da avaliação de segurança dinâmica em ambientes de computação distribuída, visando investigar a viabilidade de realizá-la *on-line*. Dentre todos os subproblemas que compõem a análise *on-line* de segurança dinâmica, este trabalho aborda apenas os subproblemas de cálculo da estabilidade transitória e avaliação da segurança, que são os mais significativos em termos de esforço computacional.

Em poucas palavras, o problema resolvido neste trabalho pode ser enunciado como segue:

Dada uma lista de contingências dinâmicas e um ponto de operação (caso básico futuro), classificar o sistema em seguro ou inseguro em relação ao conjunto de contingências. Se inseguro, calcular a margem de instabilidade. Se seguro, calcular a margem de estabilidade.

O objetivo principal da dissertação é implementar a solução desse problema em ambientes de computação distribuída e analisar seu desempenho, tendo em vista aplicações *on-line*. O uso de sistemas de computação distribuída deve ser visto como um meio de acelerar a análise através da decomposição do problema em tarefas independentes que serão

realizadas em paralelo. Deseja-se investigar o potencial de ambientes distribuídos como uma alternativa ao uso de computadores paralelos na busca de alto desempenho. Uma vantagem dos ambientes distribuídos em relação aos computadores paralelos é a facilidade de atualização do *hardware* envolvido à medida que novos processadores vão aparecendo no mercado, mantendo-se porém o mesmo *software*. Já um computador paralelo constitui uma máquina em si e não se pode mudar o processador usado sem ter de reprojetá-lo, embora alguns computadores paralelos mais recentes apresentem uma arquitetura mais flexível, como por exemplo o **SP2** da **IBM**. Outro atrativo de sistemas distribuídos consiste na possibilidade de misturar computadores de diferentes arquiteturas formando sistemas heterogêneos, inclusive computadores paralelos.

2.2 Cálculo da estabilidade transitória

A análise da estabilidade transitória associada a cada contingência é realizada através de um programa de simulação no domínio do tempo usando integração numérica passo-a-passo. Embora os métodos diretos baseados nas teorias de Lyapunov e da função da energia transitória (**TEF**) tenham recentemente evoluído muito , ainda não são inteiramente confiáveis e sofrem de restrições na modelagem dos componentes dos sistemas de energia elétrica, o que dificulta a análise quando se deseja estender a simulação além da primeira oscilação.

O programa de simulação utiliza o esquema de solução alternado entrelaçado [23], no qual os conjuntos de equações algébricas e diferenciais são resolvidos separadamente, em processos iterativos distintos, porém com iterações sucessivamente entremeadas. A Figura 2.1 ilustra o esquema em sua parte essencial para um passo de integração.

O sistema de equações algébricas é expresso na forma $I = YV$ complexa, onde **Y** representa a matriz admitância nodal. As equações diferenciais são integradas numericamente pelo método trapezoidal implícito. O programa permite a representação de modelos detalhados de máquinas síncronas (até modelo IV na classificação de Young), sistemas de excitação, reguladores de velocidade, motores de indução e diversos modelos de carga. Deve ser dito que o programa de estabilidade não foi desenvolvido nesta dissertação, utilizando-se um programa em uso no **DSEE**. Porém, foi necessário realizar modificações e desenvolver novas subrotinas para incluir a avaliação de segurança e obtenção das margens de estabilidade.

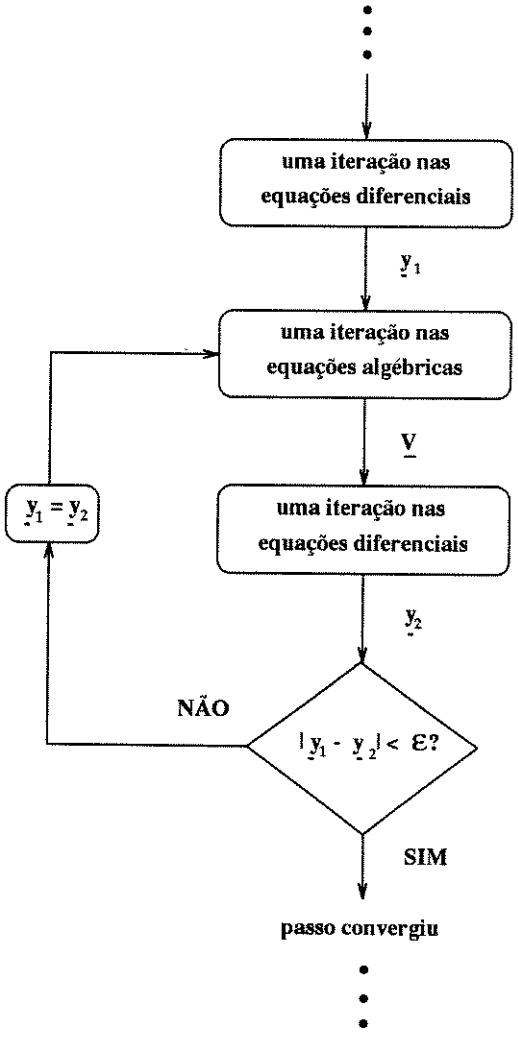

Figura 2.1: Parte essencial do esquema alternado entrelaçado implícito

A detecção de geradores *instáveis* é feita monitorando-se a variação no tempo dos ângulos dos rotores de cada gerador em relação ao centro de ângulos (**COA**) de todas as máquinas. Um gerador é classificado como instável se seu rotor sofrer uma variação angular muito grande, ou seja, maior que um limiar previamente fixado. O valor desse limiar depende evidentemente do sistema em estudo e de alguma calibração experimental. Por exemplo, no sistema-teste usado neste trabalho verificou-se que o limiar de 130 graus elétricos é adequado.

2.3 Avaliação da segurança

O procedimento seguido para avaliar a segurança e obter as margens de estabilidade baseia-se no Método Híbrido proposto por G.A. Maria [15] e modificado por C.K. Tang [11]. Essa abordagem combina simulação no tempo com o método da Função da Energia Transitória (**TEF**). As idéias básicas do método **TEF** podem ser encontradas na bibliografia citada na seção 1.6 do Capítulo 1.

A idéia fundamental do Método Híbrido é analisar a trajetória dos ângulos dos rotores ao longo do tempo, após a eliminação da falta, em hiperespaço de estado que fornece a energia potencial dos geradores. Se o sistema for instável, sua trajetória irá cruzar a Superfície Limite de Energia Potencial (**PEBS**) dentro do período de simulação. Se for estável, o cruzamento não se verificará. De acordo com a teoria **TEF**, a Superfície Limite (**PEBS**) é definida por um conjunto de pontos onde a função energia potencial atinge um máximo máximo direcional tomando-se todas as direções a partir do **SEP** e portanto representa a borda no poço de energia potencial centrado em um ponto de equilíbrio estável (**SEP**) da configuração final do sistema. Assumindo sistema conservativo, os pontos de máxima energia potencial correspondem a pontos de mínima energia cinética, e portanto um modo de determinar o ponto de cruzamento da trajetória com a **PEBS** , no caso instável, é descobrir quando a energia cinética atinge um mínimo. As Figuras 2.2 e 2.3 visam ilustrar as idéias acima.

O procedimento geral usado para obter-se a avaliação de segurança consiste de dois estágios, como descrito a seguir. Deve-se ressaltar que o Estágio 1 é aplicada a cada contingência de forma independente e que no Estágio 2 realiza-se uma análise coordenada dos resultados obtidos no estágio anterior. Esta característica de independência foi explorada para a implementação em ambientes de computação distribuída, conforme será mostrado

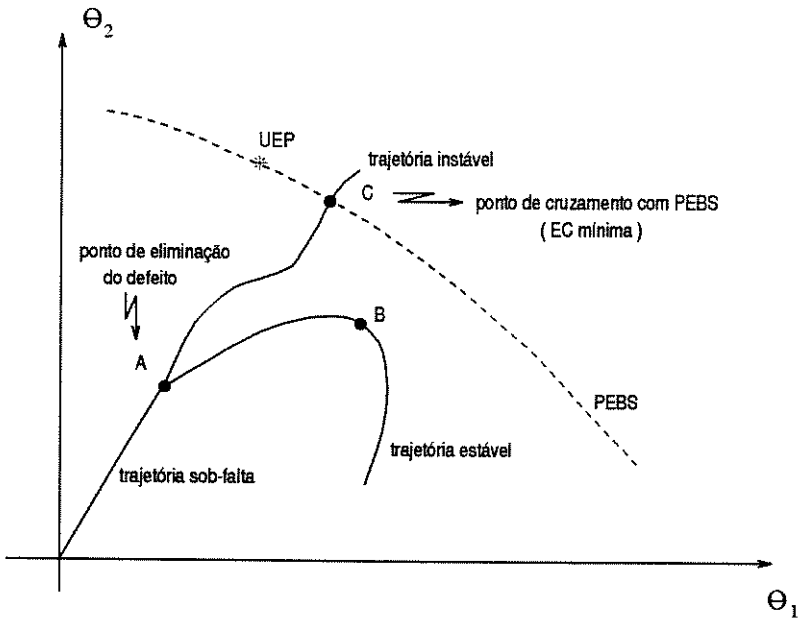

Figura 2.2: Exemplo de trajetórias estável e instável em caso de duas máquinas

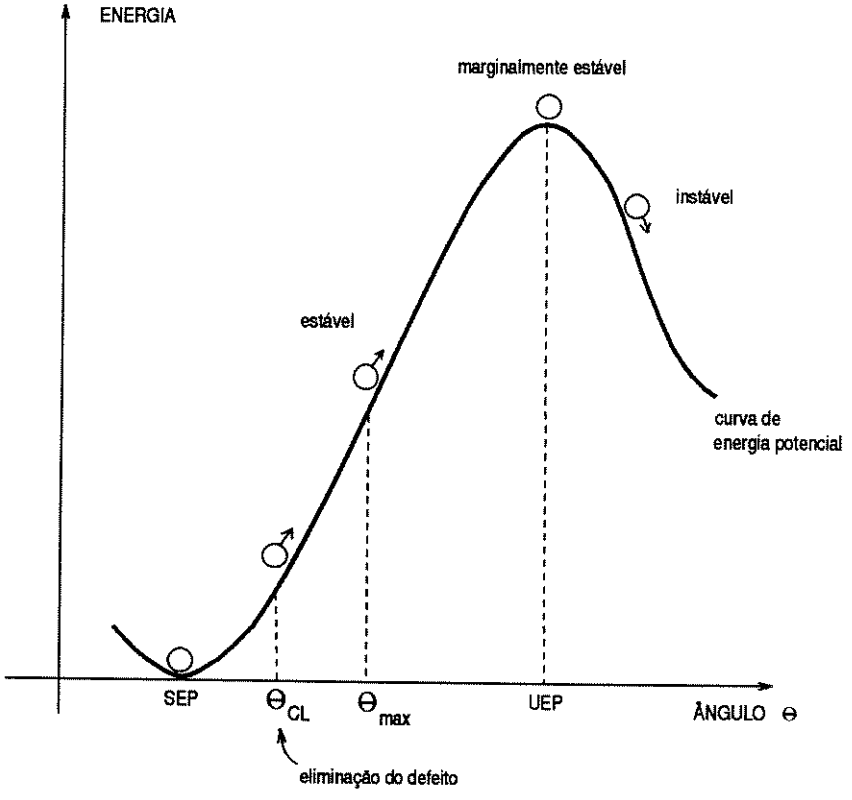

Figura 2.3: Conceito básico da abordagem TEF

no Capítulo 4, onde o Estágio 1 foi entregue a diversos processos diferentes que analisam uma das contingências e encaminham sua resposta a um processo coordenador responsável pela análise global e pela entrega do resultado final sobre a estabilidade do sistema e as respectivas margens de estabilidade ou instabilidade.

Estágio 1 : Análise de estabilidade para cada contingência

Para cada passo de integração:

- Calcular a energia cinética de todo o sistema relativa ao centro de ângulos, usando a expressão:

$$E_c = \frac{1}{2} \sum M_i \tilde{w}_i^2$$

onde M_i é a constante de inércia e $\tilde{w}_i$ é a velocidade angular em relação ao COA do gerador i e n é o número total de geradores.

- Se E_c não atingiu um mínimo, vá para o próximo passo de integração.
- Se E_c atingiu um mínimo:
  - Verifique se algum gerador é instável (detectando se o ângulo do rotor ultrapassou o limiar) ;
  - Em caso afirmativo (pelo menos um gerador instável) então classifique o caso como *instável*, calcule a *margem de instabilidade* (**MI**) e pare a simulação ;
  - Em caso negativo (todos geradores estáveis) então classifique o caso como *estável*, calcule a *margem de estabilidade* (**ME**) e pare a simulação.

Estágio 2 : Análise de segurança global

Analisando os resultados da Estágio 1 para todas as contingências, classifique o ponto de operação em *seguro* ou *inseguro*, de acordo com o seguinte critério:

- Se pelo menos uma contingência apresentar caso instável, então classifique o ponto de operação como *inseguro*;
- Se todas as contingências apresentarem casos estáveis, então classifique o ponto de operação como *seguro*;

2.4 Obtenção das margens de estabilidade e instabilidade

O conceito de máquinas avançadas é utilizado para calcular as margens de estabilidade e instabilidade. Desta forma, este assunto será abordado inicialmente.

Máquinas Avançadas

Define-se como *máquinas avançadas* aquelas que mais tendem a se separar do resto do sistema, para uma dada perturbação. Em geral, são os geradores mais próximos eletricamente da perturbação e isto é mais verdadeiro quanto maior é o sistema de potência.

O critério utilizado neste trabalho para obter as máquinas avançadas baseou-se em uma generalização do comportamento dinâmico de um sistema formado por um gerador ligado a uma barra infinita. Uma máquina será considerada avançada se, no ponto de E_c mínima, seu ângulo de rotor (em relação ao COA) estiver mais perto do ângulo crítico do que do ângulo de equilíbrio estável pós-falta.

Definindo-se

$$DPC = \mid \pi - \theta_{SEP_2} - \theta_m \mid$$

como a distância ao ponto crítico e

$$DPE = \mid \theta_m - \theta_{SEP_2} \mid$$

como a distância ao ponto estável pós-falta, o critério para considerar um gerador avançado pode ser expresso por:

$$DPC < DPE$$

A Figura 2.4 ilustra a aplicação do critério.

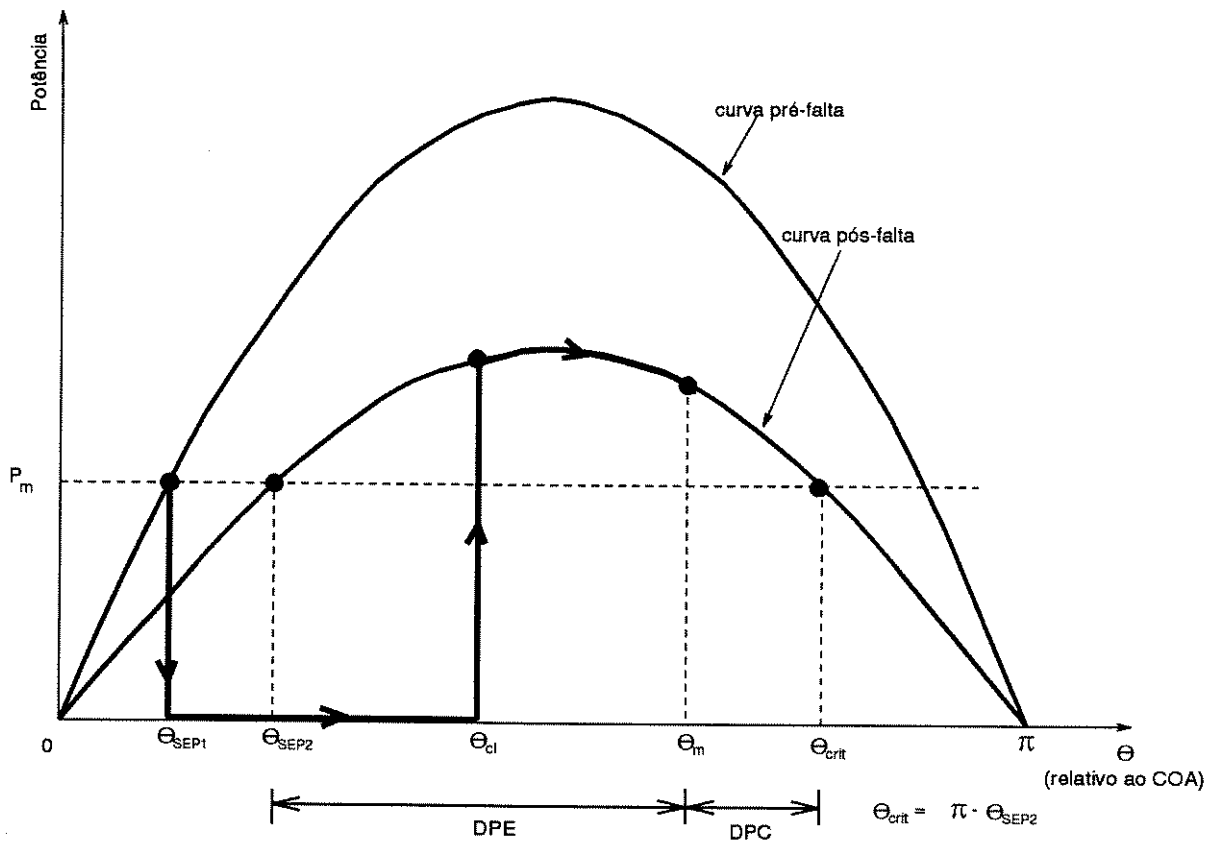

Figura 2.4: Conceito de máquina avançada

Margem de Instabilidade (**MI**)

A Margem de Instabilidade é um índice quantitativo do grau de severidade da perturbação e visa dar uma indicação numérica de quão longe o sistema está da estabilidade. A obtenção da Margem de Instabilidade neste trabalho segue basicamente o procedimento utilizado nas referências [11] e [15] para casos instáveis.

Margem de Instabilidade (**MI**) é definida como **a energia cinética corrigida associada às máquinas avançadas no ponto de energia cinética mínima.**

O conceito de energia cinética corrigida e sua importância são descritos nas referências [2, 3, 5].

Desse modo, a Margem de Instabilidade é obtida calculando-se a expressão:

$$MI = E_c^{corr} = \frac{1}{2} M_{eq} \tilde{w}_{eq}^2$$

onde:

$$M_{eq} = \frac{M_a M_b}{M_a + M_b}$$

$$M_a = \sum_{i=1}^{NA} M_i \qquad i = 1, 2, \ldots, NA$$

$$M_b = \sum_{i=1}^{NB} M_i \qquad i = 1, 2, \ldots, NB$$

e **NA** é o número de máquinas avançadas e **NB** o número de máquinas restantes.

$$\tilde{w}_{eq} = \tilde{w}_a - \tilde{w}_b$$

$$\tilde{w}_a = \sum_{i=1}^{NA} \frac{M_i}{M_a} \tilde{w}_i$$

$$\tilde{w}_b = \sum_{i=1}^{NB} \frac{M_i}{M_b} \tilde{w}_i$$

Deve ser ressaltado que a Margem de Instabilidade é calculada usando-se somente resultados obtidos durante a simulação no domínio do tempo.

Margem de Estabilidade (**ME**)

A Margem de Estabilidade também fornece o grau de severidade da perturbação no caso estável. É um índice numérico que indica quão longe da instabilidade o sistema está.

Neste trabalho, a Margem de Estabilidade é definida como **a variação da energia potencial da máquina mais avançada entre o ponto de energia cinética mínima e o ponto crítico.**

Seguindo essa definição a Margem de Estabilidade, deve-se destacar que a **ME** é função de uma integral dependente do caminho seguido sistema, podendo-se fazer uma aproximação através das seguintes expressões:

$$ME = \Delta E_p = \int_{\theta_{m_i}}^{\theta_{crit}} \phi_i^{\theta_m} d\theta = \frac{1}{2}\phi_i^{\theta_m}(\pi - \theta_{SEP2} - \theta_{m_i})$$

onde:

$$\phi_i = P_{m_i} - P_{e_i} - \frac{M_i}{M_i} P_{COA}$$

$$P_{COA} = \sum_{k=1}^{n} (P_{m_k} - P_{e_k})$$

e o índice i refere-se à máquina mais avançada.

As Figuras 2.5 e 2.6 visam ilustrar o procedimento e mostrar o grau de aproximação utilizado na forma de cálculo para a margem de estabilidade.

Mais uma vez deve-se ressaltar que a Margem de Estabilidade é calculada usando-se somente resultados obtidos durante a simulação no domínio do tempo.

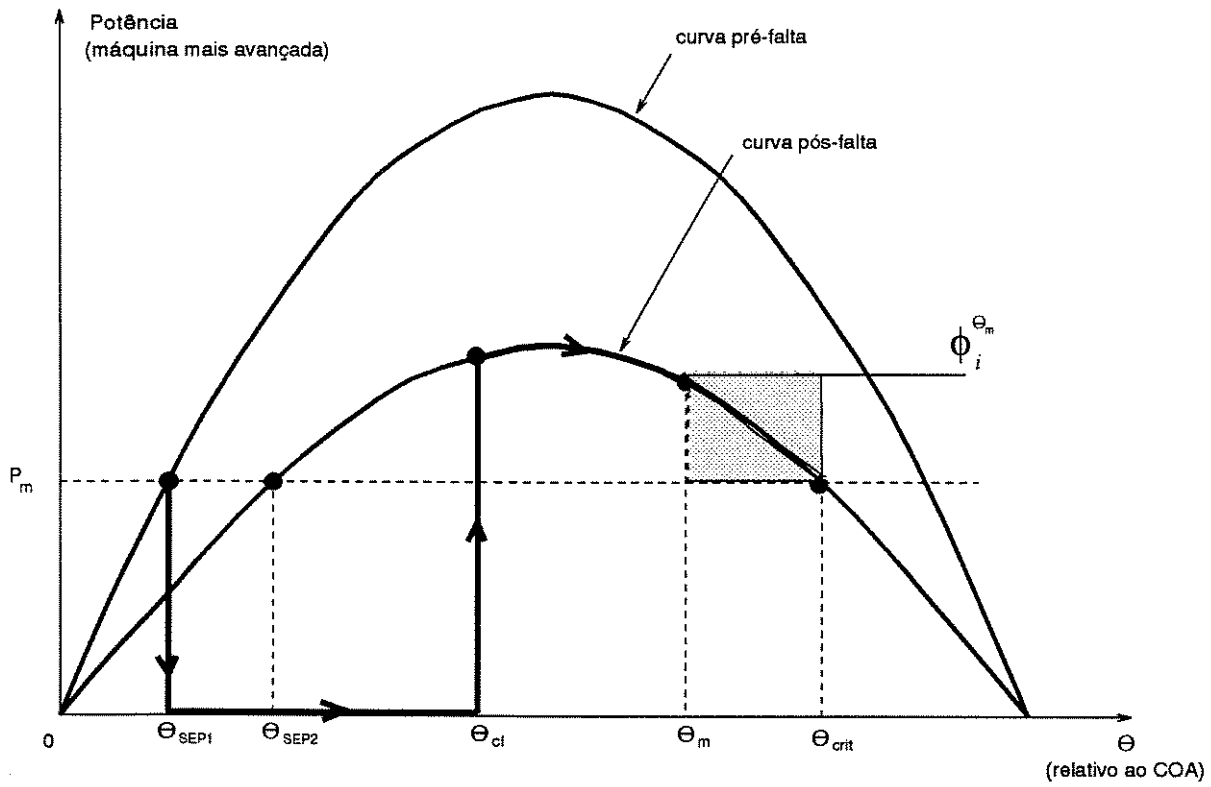

Figura 2.5: Definição da margem de estabilidade

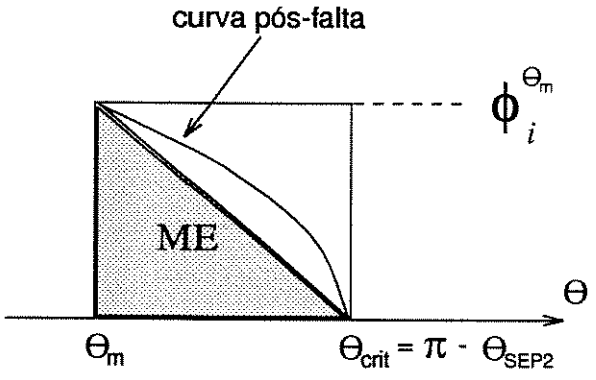

Figura 2.6: Aproximação feita no cálculo da E_p

Exemplo Ilustrativo

Com o objetivo de deixar mais claro os procedimentos de obtenção das margens, inclui-se neste item um exemplo ilustrativo com as seguintes características:

- Sistema Sul-Sudeste 320 barras, 470 linhas, 46 geradores representados por modelo IV com sistema de excitação;
- Contingência: curto trifásico na barra 304 com eliminação através do desligamento da linha 302-304 no instante t_{cl} ; a barra 304 está eletricamente muito próxima do Gerador 2;
- Passo de integração: 5 ms e ângulo máximo para detectar instabilidade: 130 graus.

Os resultados de várias simulações com diferentes tempos de eliminação são mostrados na tabela 2.1, onde t_{cl} indica o tempo de eliminação, E_c^{min} dá o valor da energia cinética total mínima, t_{min} indica o instante no qual a energia cinética é mínima, **Ger.av.** indica quais geradores estão avançados em t_{min}, **Avaliação** indica a classificação da contingência e Margem dá o valor da margem de estabilidade ou instabilidade. Na Figura 2.7 mostra-se a evolução da energia cinética total no tempo, para o caso t_{cl}=0.225s e na Figura 2.8 a trajetória do ângulo do rotor do Gerador 2 (instável) em relação ao COA.

t_{cl} (s)	E_c min (pu)	t_{min} (s)	Ger. av.	Avaliação	Margem (pu)
0.200	2.66	0.55	-	estável	ME= -1.94
0.210	1.91	0.60	2	estável	ME= -1.42
0.215	1.28	0.65	2	estável	ME= -0.64
0.220	1.65	0.63	2	instável	MI= 0.41
0.225	3.16	0.55	2	instável	MI= 1.12
0.230	4.45	0.50	2	instável	MI= 1.45

Tabela 2.1: Resultados para diferentes tempos de eliminação

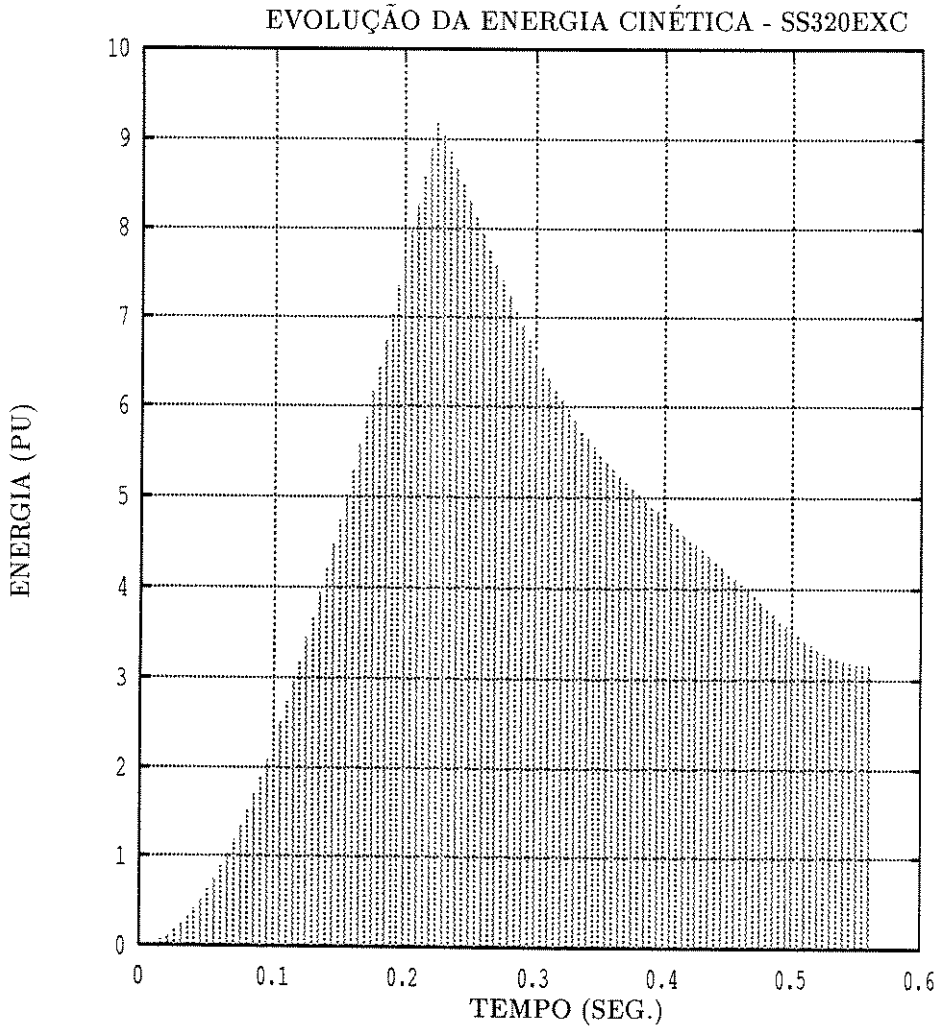

Figura 2.7: Gráfico da energia cinética para um caso instável - $t_{cl} = 0.225s$

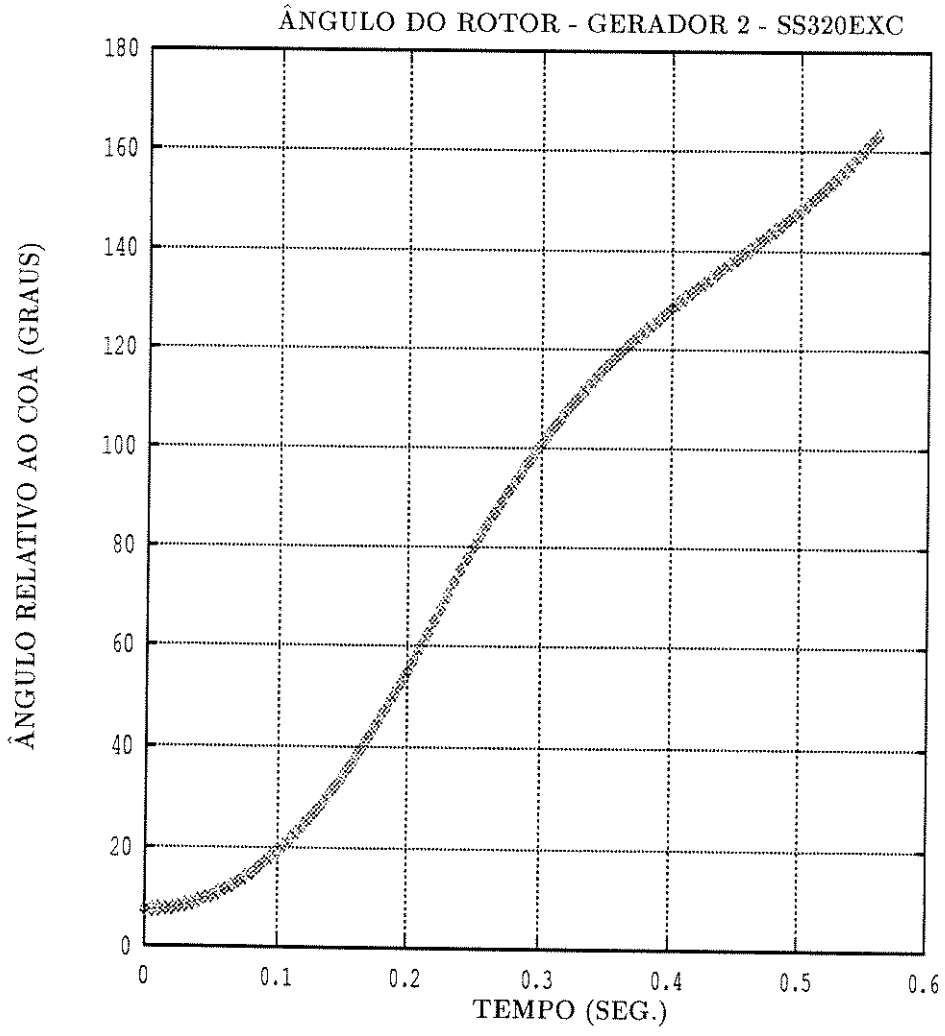

Figura 2.8: Trajetória do ângulo do rotor para um caso instável - $t_{cl} = 0.225s$

Capítulo 3

Ambientes de Computação Distribuída

3.1 Introdução

Hoje em dia tornou-se possível o uso de ambientes de trabalho que integram diferentes computadores, os quais vão desde estações de trabalho, computadores pessoais e até supercomputadores conectados por um sistema de comunicação como ilustrado na Figura 3.1. A este conjunto de computadores interligados dá-se o nome genérico de ambiente de *computação distribuída.* Com isto surgiu a idéia de utilizar esses recursos computacionais disponíveis para resolver um único problema paralelo. A vantagem dos ambientes de computação distribuída é sua versatilidade que contrasta com a rigidez de computadores paralelos que, em última análise, constituem uma única máquina projetada com processadores, formas de intercomunicação e arquiteturas definidas.

A flexibilidade dos ambientes distribuídos se origina no agrupamento de várias máquinas, com diferentes configurações de *hardware* e de formas de interligação, o que permite substituir partes do equipamento pelo que há de mais novo no mercado, possibilitando a expansão da memória **RAM**, a troca dos processadores, o aumento do disco rígido, a mudança do tipo de interconecção de, por exemplo, *Ethernet* para **FDDI**, não havendo problema em desligar temporariamente parte do equipamento ou a necessidade de mudar o código do programa fonte. Um **DCE** (*Distributed Computing Enviroment*) pode ser reutilizado como uma máquina paralela virtual, com arquitetura **MIMD** de memória distribuída

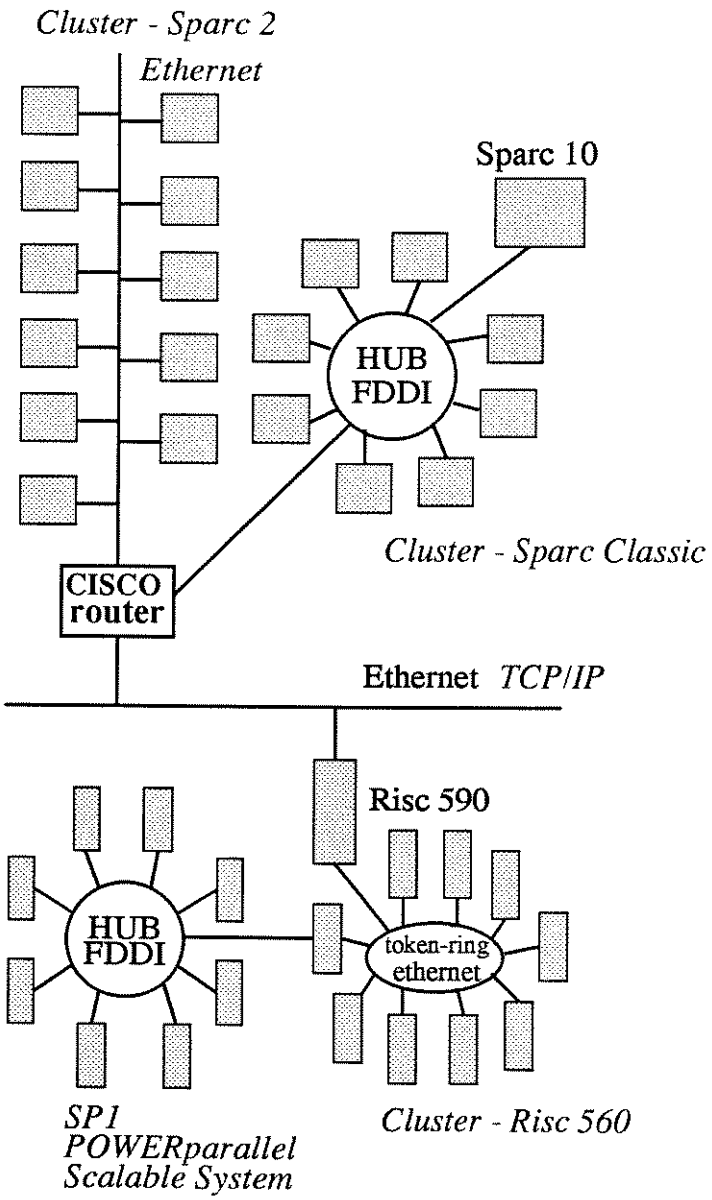

Figura 3.1: Exemplo de um ambiente de computação distribuída

e comunicação entre os processadores através de troca de mensagens. Por essa razão, é adequado para problemas de granularidade maior (grão grosso) , como a aplicação realizada neste trabalho.

O objetivo desta dissertação foi o de investigar o comportamento de um programa de simulação para avaliação da segurança dinâmica de um sistema elétrico em um ambiente de computação distribuída. Basicamente os testes ocorreram em ambientes com duas arquiteturas diferentes de computadores: estações de trabalho **Sun**, situadas em dois laboratórios do **DSEE** e **IBM**/Risc6000, situadas no Centro Nacional de Processamento de Alto Desempenho (**CENAPAD**) localizado no campus da Unicamp. Os modelos variaram como descrito abaixo:

- Sun: Sparc 2, Sparc 10;
- IBM/RISC6000: 370 (nós do SP1), 55L, 560, 590.

Além disso, as redes usadas detinham diferentes formas e o meios de interconecção, variando de cabos coaxiais *Ethernet* **LAN** e *Ethernet token-ring* **LAN**, até cabos de fibra ótica numa ligação *token-ring* **LAN** (*Fiber Distributed Data Interface*/**FDDI**).

Uma descrição completa do ambiente do **CENAPAD** pode ser vista através do seguinte endereço Internet:
`http://www.cenapad.unicamp.br/CENAPAD/Informacoes/Hardware/diagrama.html`.

Através desse endereço acessa-se a página conforme mostrado na Figura ??, bastando dar um clique no objeto desejado para se obter maiores informações.

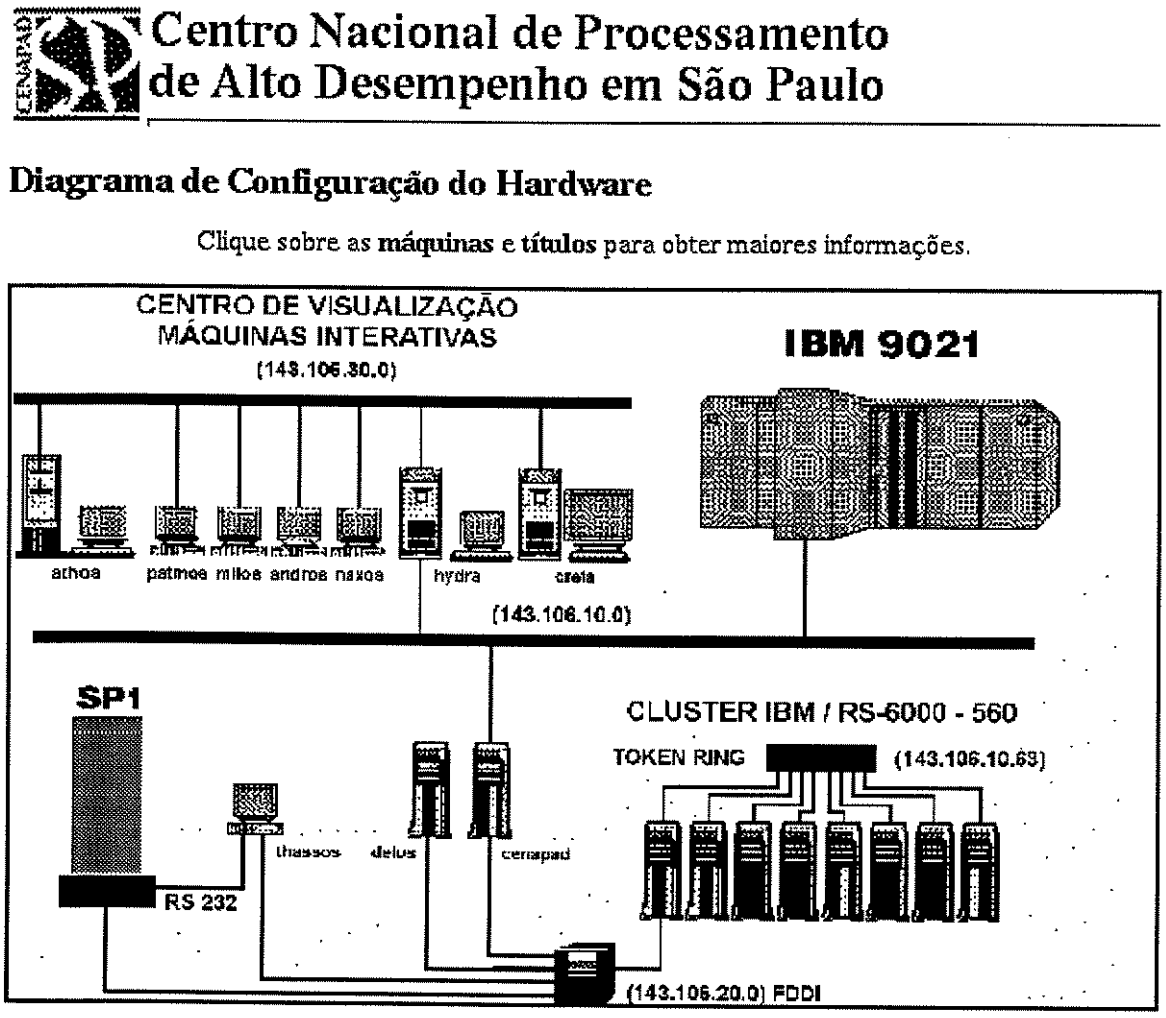

Figura 3.2: Performance no DCE # 1 na Fase 2 com 10 hosts

3.2 Descrição dos Ambientes de Computação Distribuída

Em seguida, descreve-se resumidamente as características de cada ambiente distribuído utilizado para avaliar o desempenho da análise *on-line*.

3.2.1 DCE # 1 : Sparc2 conectadas via Ethernet

Este ambiente de computação distribuída, descrito na Figura 3.3, consiste de 11 estações de trabalho Sun-Sparc, sendo uma do modelo 470 com 32 Mbytes de RAM e dez do modelos Sparc 2 com 16 Mbytes de RAM, velocidade de *clock* igual a 33.0 MHz, conectados por cabos coaxiais *Ethernet* **LAN**, possuindo uma banda de passagem de 10 Mbps (valor nominal).

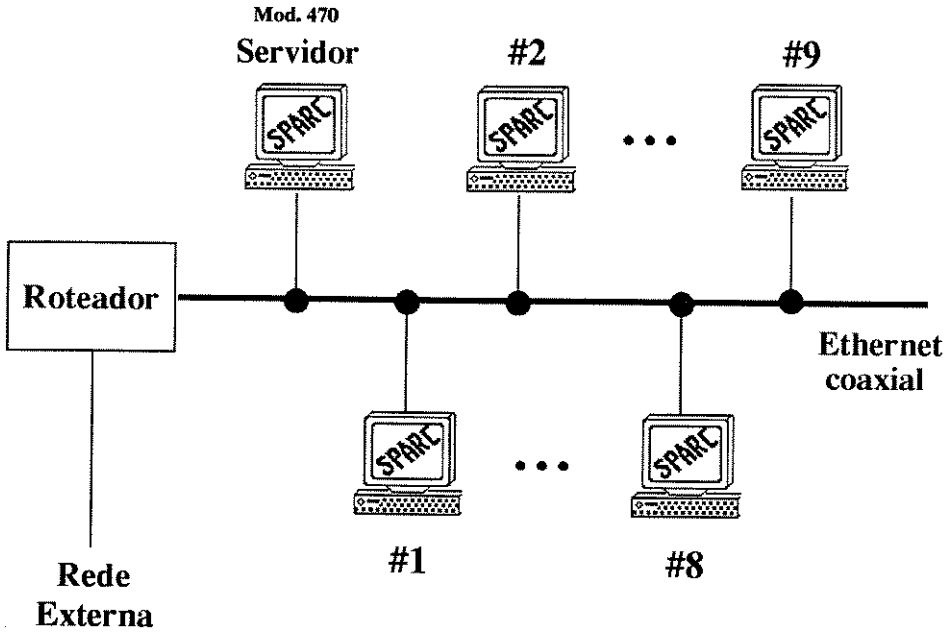

Figura 3.3: Visão geral da estrutura do DCE # 1

3.2.2 DCE # 2 : Risc6000 conectadas via Ethernet

A Figura 3.4 mostra um *cluster* homogêneo de oito estações de trabalho com processadores IBM POWER RISC/6000 (uma do modelo 590 com velocidade de *clock*: 66.7 MHz e sete do modelo 560 com *clock*: 50 MHz) com 128 Mbytes de memória RAM, interconectadas por uma rede de cabos coaxiais *Ethernet token-ring* **LAN** com uma banda nominal de passagem de 16 Mbps.

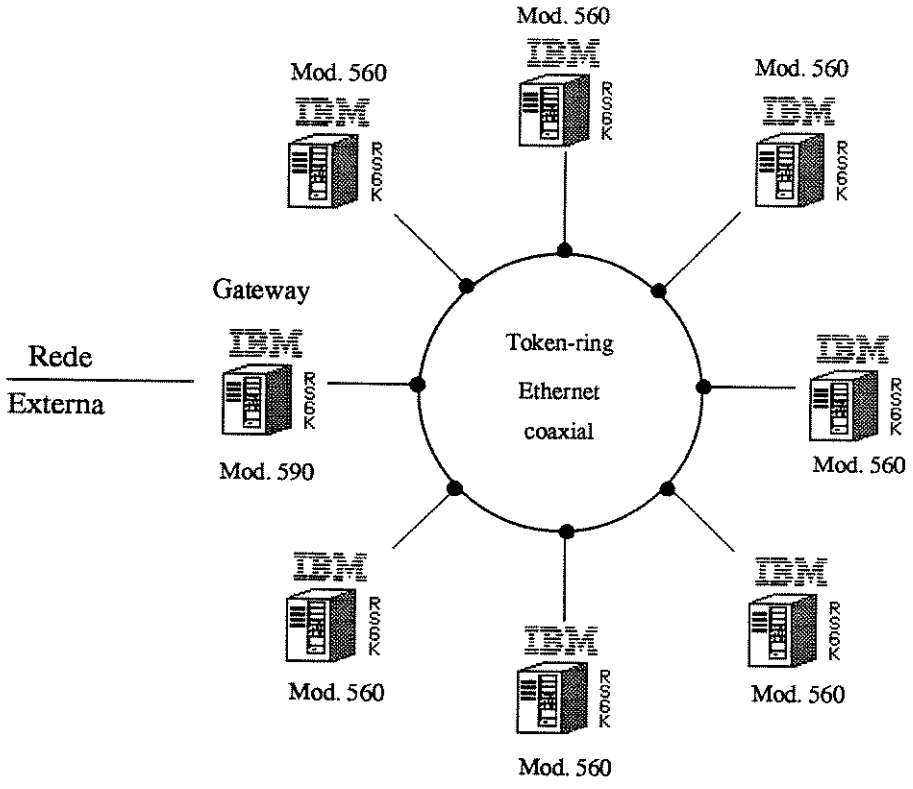

Figura 3.4: Visão geral da estrutura do DCE # 2

3.2.3 DCE # 3 : Risc6000 conectadas via FDDI

Outro ambiente utilizado nos testes é um *cluster* composto de oito estações de trabalho **IBM** (modelo 370 com processador IBMPOWER Risc/6000) que constituem os nós de um supercomputador SP1 (*Scalable PowerParallel System* 1). Cada nó possui 256 Mbytes de RAM, uma velocidade de *clock* igual a 62.6 MHz e atingindo uma performance de pico igual a 125 Mflops, sendo interconectados via um *hub (active wire center)* através de cabos de fibra ótica usando *token-ring* **LAN FDDI** e trabalhando com uma banda nominal de passagem de 100 Mbps, conforme mostrado na Figura 3.5. Este ambiente contém as máquinas mais rápidas entre todas as usadas nos testes.

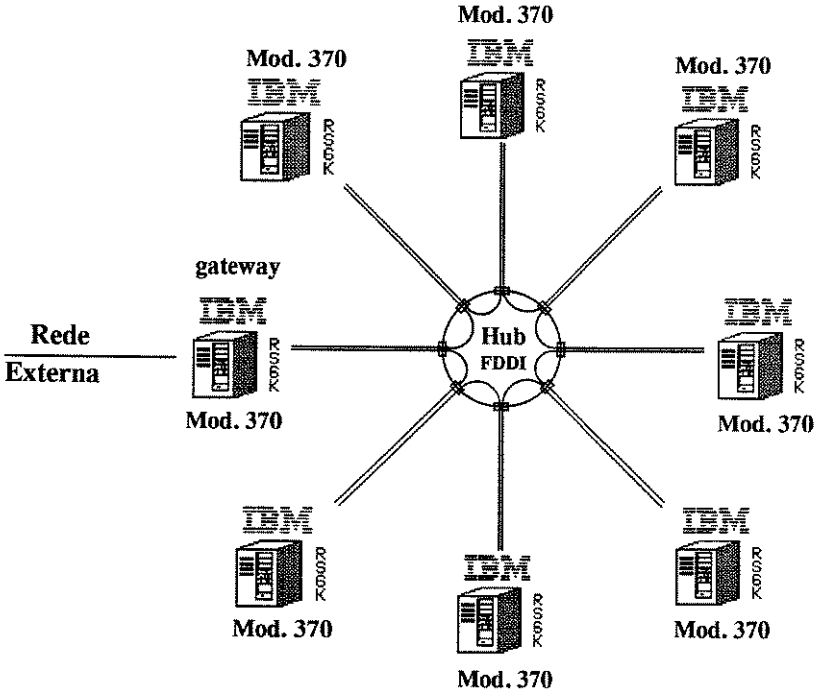

Figura 3.5: Visão geral da estrutura do DCE # 3

3.2.4 DCE # 4 : Risc6000 & SP1

Este ambiente consiste na união dos dois ambientes citados anteriormente formando uma nova configuração que contém 16 máquinas, aumentando desta forma o poder computacional agregado. Note-se que as máquinas têm a mesma arquitetura mas desempenhos diferentes. A Figura 3.6 ilustra este ambiente e mostra as estações locais do **LSEE**, pertencentes ao **DSEE**, a partir de onde é feito o acesso ao **CENAPAD**.

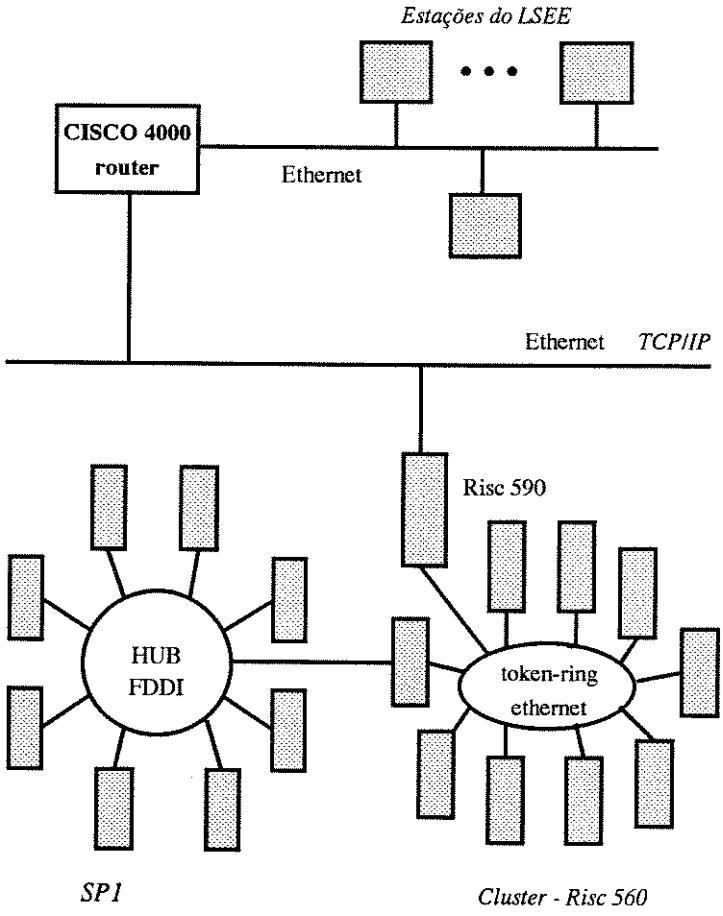

Figura 3.6: Visão geral da estrutura do DCE # 4

3.2.5 DCE # 5 : Sparc Classic conectadas via FDDI

Este ambiente de computação distribuída, descrito na Figura 3.7, consiste de nove estações de trabalho Sun-Sparc, sendo uma modelo 10 , com 32 Mbytes de RAM, velocidade de *clock* igual a 40.0 MHz e oito modelo Sparc Classic com 16 Mbytes de RAM, velocidade de *clock* igual a 50.0 MHz conectadas por cabos de fibra ótica a um *hub (active wire center)* e usando um protocolo **FDDI** com banda de passagem de 100 Mbps (valor nominal).

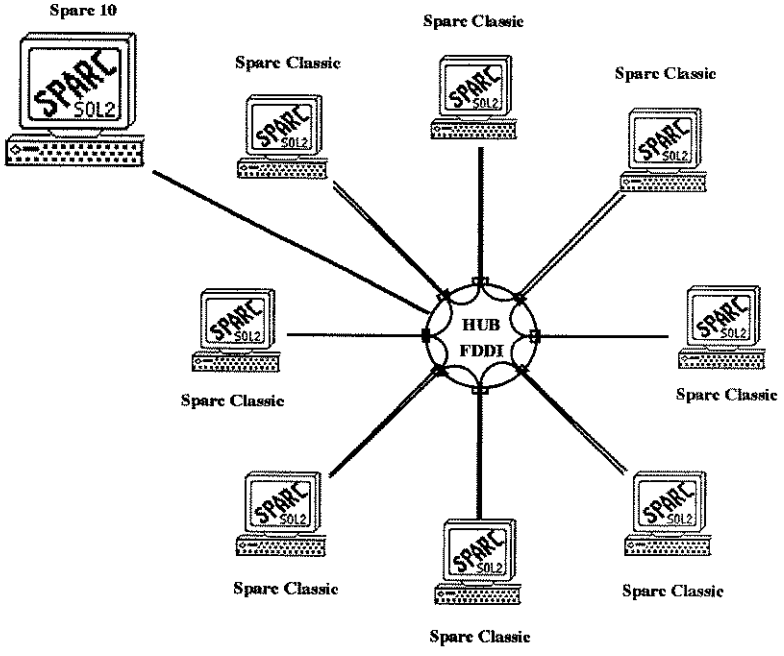

Figura 3.7: Visão geral da estrutura do DCE # 5

3.2.6 DCE # 6 : Risc6000 & SP1 & Sparc2 & Sparc Classic

Este ambiente distribuído resulta do agrupamento de todas as estações disponíveis constituindo uma máquina paralela virtual com 36 *hosts*. Ele corresponde a agregar três dos ambientes já citados: **DCE** # 1 + **DCE** # 4 + **DCE** # 5. Vale a pena ressaltar algumas particularidades para este ambiente: a distância física entre as estações que formam o **DCE** # 4 em relação aos outros é de aproximadamente 600 metros. Agrupou-se máquinas com sistemas operacionais diferentes, bem como arquiteturas e modelos diferentes, constituindo um ambiente altamente heterogêneo, conforme ilustrado na Figura 3.8.

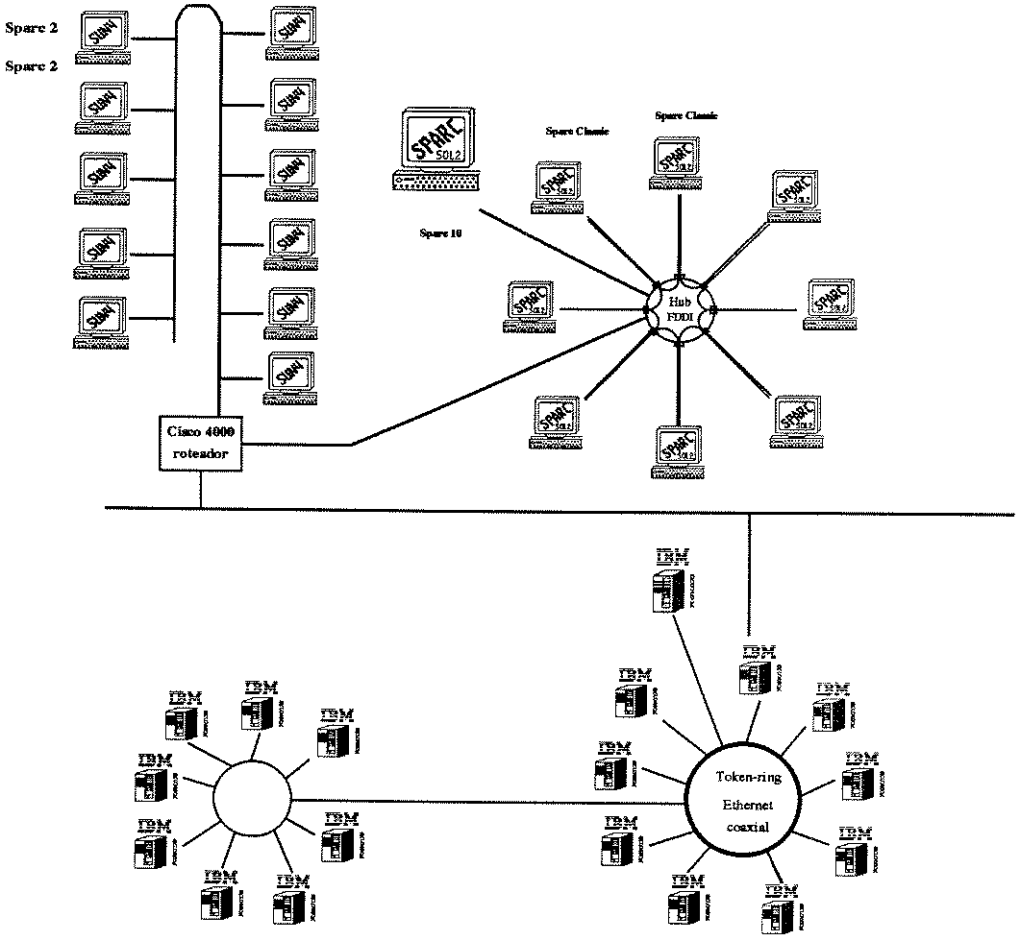

Figura 3.8: Visão geral da estrutura do DCE # 6

Capítulo 4

Implementação

4.1 PVM como *Software* de Controle

Um fator crucial na utilização de ambientes distribuídos para a solução de um único problema é como gerenciar a alocação, coordenação e controle das tarefas (processos) realizados em paralelo nos diversos processadores. Felizmente existem disponíveis alguns *softwares* que permitem fazer esse gerenciamento em alto nível sem programar diretamente em UNIX. Neste trabalho, optou-se por utilizar o **PVM**, na versão 3.3.7, como *software* de controle por sua fácil disponibilidade, portabilidade e desempenho.

PVM (*Parallel Virtual Machine*) é um *software* desenvolvido para permitir a comunicação entre uma rede heterogênea composta por computadores seriais, paralelos ou vetoriais, fazendo com que eles funcionem como um único e grande computador paralelo (máquina virtual). Cada computador que compõe esta máquina virtual é denominado *host*. O **PVM** pode ser configurado para agregar diversos tipos de arquitetura de computadores, incluindo processadores seqüenciais, vetoriais e computadores multiprocessados, possuindo ainda um código portável para arquiteturas que vierem a surgir no futuro. O pacote de bibliotecas e programas do **PVM** vem sendo desenvolvido e aperfeiçoado no Laboratório Nacional de Oak Ridge desde de 1989, sendo de domínio público [14].

Quando se trata de um computador paralelo multiprocessado, cada processador é exatamente igual aos outros no que se refere a poder de processamento, desempenho, programas instalados e velocidade de comunicação. Isto não é válido quando se usa uma rede interligada. Cada computador que compõe a rede pode ser de arquitetura diferente

(diferentes fabricantes) ou possuir diferentes compiladores. Quando um programador deseja explorar uma rede de computadores em conjunto, ele deve lidar com diferentes tipos de heterogeneidade:

- arquitetura;
- formato de dados;
- velocidade de processamento;
- carga da máquina;
- carregamento da rede.

O conjunto de computadores disponíveis pode incluir variados tipos de arquitetura desde PC 386/486 até estações de trabalho de alto desempenho, multiprocessadores de memória compartilhada, supercomputadores vetoriais e processadores massivamente paralelos. Para cada tipo de arquitetura há um método ótimo de programação. Acrescente-se ainda o fato do usuário ter que lidar com problemas de hierarquia de decisões de programação. A máquina paralela virtual em si, pode ser composta por computadores paralelos. Mesmo quando os *hosts* forem apenas estações de trabalho seriais, ainda haverá o problema da incompatibilidade entre os formatos binários e da necessidade de compilar os programas fontes em cada máquina.

Formatos de dados em computadores diferentes normalmente são incompatíveis. Tal incompatibilidade é um ponto importante em computação distribuída pois dados enviados por um computador **A** podem ser ilegíveis para um computador **B** que venha a recebê-los. Pacotes de "troca de mensagens" desenvolvidos para ambientes heterogêneos devem garantir que todos os computadores compreendam os dados trocados entre si. Infelizmente os primeiros sistemas baseados em troca de mensagens desenvolvidos especificamente para alguns tipos de computadores massivamente paralelos não são adequados para computação distribuída pois não incluem suficiente informação na codificação ou decodificação da mensagem que permita a outros interpretá-las.

Mesmo que o conjunto de computadores seja formado por estações de trabalho com o mesmo formato de dados, ainda resta o problema de heterogeneidade devido às diferenças de velocidade de processamento. Um exemplo simples é o problema de executar-se tarefas paralelas em uma máquina virtual composta por um supercomputador e uma estação de trabalho. O programador deve ter o cuidado de não deixar o supercomputador

inativo aguardando o envio de dados de uma estação de trabalho, para então prosseguir a sua execução. Outra dificuldade é que uma rede de computadores pode possuir vários usuários executando compartilhadamente diferentes tipos de programas, que requisitam parcial ou totalmente os recursos disponíveis (memória, disco rígido), fazendo a carga de cada máquina variar dramaticamente. Como resultado desses fatores descritos acima tem-se que o poder computacional efetivo, mesmo usando-se máquinas idênticas, pode variar dentro de uma larga faixa.

Assim como a carga da máquina varia devido a diversos fatores, o tempo gasto para enviar uma mensagem pode variar também, dependendo da taxa de utilização da rede no instante de envio, o que é imposto pelos demais usuários da rede. Este tempo de envio torna-se importante quando uma tarefa é mantida inativa aguardando uma mensagem oriunda de outra máquina, e mais ainda quando o algoritmo paralelo é sensível ao tempo de recebimento de respostas para efetuar algum sincronismo. Fica evidente que, em se tratando de computação distribuída, o efeito das heterogeneidades pode aparecer até mesmo em simples programas.

Apesar das muitas dificuldades causadas pela heterogeneidade, enumeradas acima, a computação distribuída pode oferecer inúmeras vantagens:

- usando o *hardware* existente, o custo computacional pode ser bem pequeno;
- o desempenho pode ser otimizado pela distribuição de cada tarefa individual para a arquitetura mais apropriada;
- pode ser explorada a característica de executar aplicações especiais que só possam ser inicializadas em plataformas específicas pertencentes a uma máquina virtual;
- os recursos do computador virtual podem crescer ou ir sendo alterados aos poucos, tirando proveito das tecnologias de redes e de computadores mais modernas existentes no mercado;
- o desenvolvimento de programas pode ser feito utilizando-se um ambiente familiar, onde o programador pode utilizar editores, compiladores e depuradores que estão disponíveis individualmente em cada máquina;
- pode ser implementada uma tolerância a falha, quer seja devido à perda de um computador que integra a máquina virtual ou perda da conexão entre um computador e a rede, ou pelo atraso excessivo em retornar o resultado de um cálculo ocasionado

pelo compartilhamento da **CPU** do computador por outros processos de outros usuários. Para tanto, podem ser utilizadas as rotinas de `pvmfnrecv ( )` para temporizar a recepção esperando durante um intervalo de tempo máximo pelos resultados e tomando a atitude de redirecionar os cálculos originariamente enviados para outro computador que esteja disponível.

Todos esses fatores descritos para reduzir o tempo de desenvolvimento de uma aplicação, os testes, o processo de depuração, de modo a diminuir os custos e possibilitar o melhor resultado final são os pontos explorados pelo **PVM** .

4.1.1 Noções sobre o PVM

Um processo UNIX pode ser considerado um exemplo de um programa (arquivo executável) que é executado pelo sistema operacional de um computador. No **PVM** , uma *tarefa* é definida como sendo uma unidade de computação de maneira análoga a um processo UNIX. Em poucas palavras, pode-se dizer que o **PVM** fornece um mecanismo de gerenciamento para tarefas permitindo que estas sejam inicializadas, se comuniquem e façam um sincronismo entre si. As tarefas **PVM** comunicam-se baseadas no paradigma de **troca de mensagem**, incluindo toda e qualquer conversão de dados que se faça necessária devido a diferentes representações de ponto flutuante ou inteiro usado pelos computadores que compõem a máquina virtual.

As aplicações dos usuários, que podem ser escritas em Fortran 77 ou em Linguagem C ou C++, podem ser paralelizadas como um conjunto de **tarefas cooperantes**. O **PVM** possui uma biblioteca para rotinas padrões de interface que contêm subrotinas que um usuário pode chamar dentro do seu programa (Fortran ou C) para troca de mensagens, inicialização de processos, coordenação de tarefas e modificação na configuração da máquina virtual. Essas subrotinas devem ser inseridas no código fonte do programa paralelo em locais apropriados. Outra exigência é que o novo código fonte deve ser compilado e carregado com as bibliotecas do **PVM** . Do ponto de vista do usuário, o **PVM** pode ser visto como uma linguagem de alto nível (construída em cima do kernel do UNIX) que fornece uma completa interface de programação para ambientes de computação distribuída e, por isso, tem obtido uma grande aceitação na comunidade de computação de alto desempenho.

Uma vez inicializado, o processo mestre do **PVM** configura a máquina virtual (definida previamente pelo usuário) e dispara um *daemon* em cada *host* (computador que

compõe a máquina virtual). Um *daemon* é um processo que executa em *background* um procedimento específico ou simplesmente espera a ocorrência de um determinado evento. Um exemplo clássico de um processo *daemon* é o serviço de correio eletrônico. O *daemon* do **PVM** (pvmd) configurado em cada *host* atua como um roteador e controlador de mensagens, provendo uma eficiente forma de comunicação entre tarefas de usuários e outros *daemons*, conforme ilustrado na Figura 4.1 . Todo *daemon* é dependente da arquitetura da máquina na qual irá rodar, porém formatos de dados incompatíveis entre *hosts* são automaticamente convertidos. Mais especificamente, o procedimento de comunicação entre processos no **PVM** é baseado no conceito de *sockets*, normalmente utilizado em programação UNIX para redes de computadores [30]. Este mecanismo é implementado em todos os três tipos de conexão utilizados: entre *daemons*, entre *daemon* e tarefa, e entre tarefas.

Deve ser ressaltado que um usuário do **PVM** não precisa necessariamente saber muito sobre como o funcionam internamente as rotinas do **PVM** , porém um conhecimento mais detalhado destas pode ser de grande valia no desenvolvimento e aperfeiçoamento da sua aplicação. Um dos fatores que torna o **PVM** atrativo é o fato que suas bibliotecas de rotinas primitivas são tudo que o usuário necessita para programar sua aplicação em um ambiente de computação distribuída. Uma descrição sucinta das principais rotinas do **PVM** pode ser encontrada no Apêndice B. Maiores detalhes podem ser encontrados na referência [14].

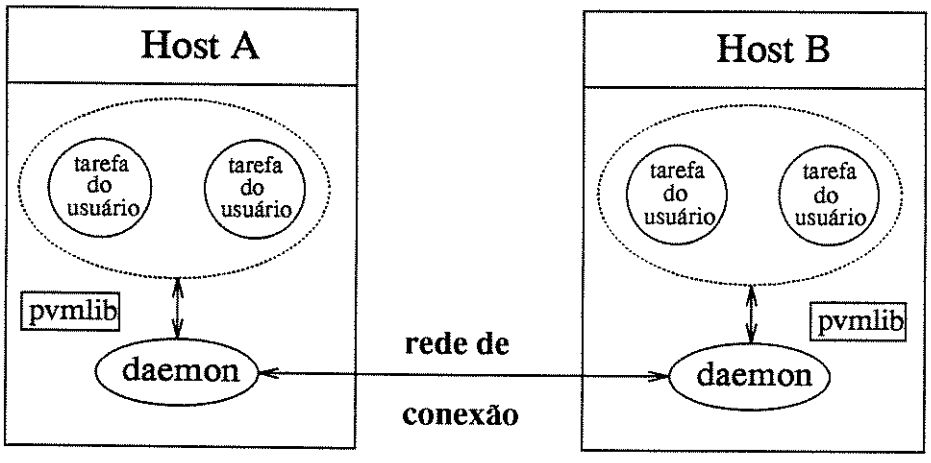

Figura 4.1: Mecanismo de comunicação entre processos no PVM

4.2 Implementação - Fase 1

A implementação teve duas fases. Em ambas, atribuiu-se uma contingência a cada processo, que avaliou a respectiva estabilidade transitória. Na Fase 1, cada processo simula a respectiva contingência durante um intervalo de tempo previamente fixado (1.15 segundos para o modelo clássico e 5.00 segundos para o modelo detalhado) e escreve um arquivo de saída. Na Fase 2, cada processo simula a respectiva contingência até encontrar um ponto de energia cinética mínima, quando então interrompe a simulação e calcula as margens de estabilidade ou instabilidade.

Em ambas as fases, o modelo de programação escolhido para a implementação do programa paralelo foi o **SPMD** (*Single Program Multiple Data*). Neste modelo, o mesmo programa é executado em todas os processos, mas cada um trabalha com seus próprios dados. O diagrama de blocos da Figura 4.2 mostra os diversos passos do programa na Fase 1.

Conforme mostrado no diagrama de blocos, a primeira etapa de qualquer execução de um programa em paralelo é a de configurar a máquina virtual (**VM**), ou seja, disparar um processo de controle *daemon* em cada computador ou estação de trabalho pertencente a **VM**. Nem todas as estações configuradas serão necessariamente utilizadas para realizar o processamento, pois algumas poderão ser descartadas caso seja conveniente, por exemplo, se for detectado que uma determinada estação está sobrecarregada com muitos processos. Para configurar utiliza-se um arquivo denominado `hostfile` (veja exemplo na Figura 4.3). Feito isso basta executar o programa principal que se encarregará de criar o processo-pai. Este processo é o primeiro a ser criado e funciona como uma espécie de supervisor, porém também analisa uma contingência, não ficando ocioso. O processo-pai verifica através da rotina `pvmfconfig( )` (ver detalhes no Apêndice B) em que máquinas configuradas poderá inicializar processos-filho criando então N processos-filho por máquina (nos testes realizados os valores atribuídos a N foram 2,4,8,12,16). Observe-se que o código executado no processo-pai e nos processos-filhos é o mesmo.

A partir desse ponto do código executável, o processo-pai adere ao grupo dinâmico (ver Apêndice A) recebendo o número "0" (zero) , efetua a leitura dos dados da rede e do fluxo de carga contidos em um arquivo e processa a montagem das matrizes e vetores a serem usados na simulação. Estes vetores são transmitidos para todos os processos-filho que compõem o grupo dinâmico através da rotina `pvmfbcast( )` do **PVM** que permite aos

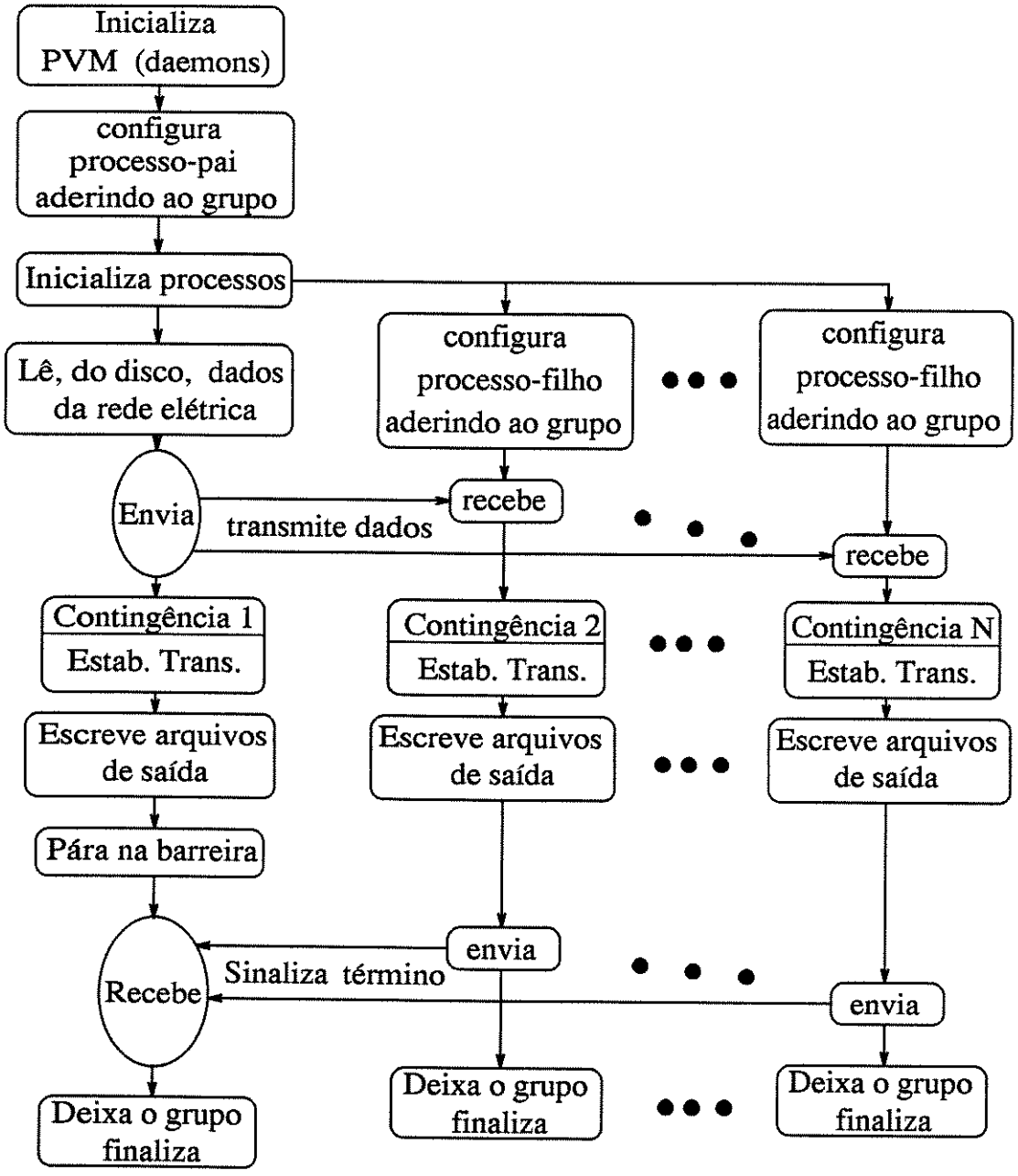

Figura 4.2: Diagrama de blocos contendo detalhes de implementação da Fase 1

membros de um grupo dinâmico trocarem dados entre si de uma maneira muito eficiente. Esta rotina envia os dados para todos os processos-filho de uma só vez.

De posse dos dados, cada processo, quer seja um processo-filho ou o processo-pai, efetua o cálculo de uma contingência diferente. O mecanismo pelo qual cada processo descobre qual contingência deve simular, é o seguinte: de posse da lista de barras do sistema recebido do processo-pai, o processo-filho simula um curto-circuito trifásico, sem desligamento de linha, com eliminação no instante t_{cl}= 0,30 segundos, aplicado na barra $1 + n_{grp}$, onde n_{grp} é número do processo no grupo dinâmico. Como o número de um processo no grupo é único, garante-se que cada contingência simulada será diferente. Note que nesta Fase 1 não é fornecida explicitamente uma lista de contingências para os processos.

Uma vez efetuado o cálculo da estabilidade e a avaliação de segurança, cada processo-filho escreve o seu respectivo arquivo de saída, envia um sinal para o processo-pai e em seguida abandona o grupo finalizando a execução. O processo-pai, após efetuar o cálculo da estabilidade e a avaliação de segurança da sua contingência, fica aguardando a recepção de mensagem de todos os filhos que compõem o grupo e então escreve os tempos gastos em cada parte do programa antes de terminar a sua execução.

Nesta Fase 1 optou-se por deixar que cada processo individualmente escrevesse seus resultados da simulação (t_{max} = 5 segundos) em arquivos próprios, de modo a verificar se os resultados da simulação em paralelo coincidiam com aqueles obtidos por uma versão serial do programa. Com este procedimento buscou-se testar a confiabilidade do **PVM** , que mostrou ser adequada. A Tabela 4.1 apresenta o tamanho (em Kbytes) de alguns arquivos de saída para um exemplo no qual foram inicializados 120 processos (10 máquinas x 12 processos por máquina). A Tabela 4.2 mostra o tamanho de alguns arquivos de medida de tempo, indicando que estes são bem menores que os arquivos de saída. Se for computada a soma do tamanho de todos os arquivos (resultados + tempos) tem-se uma noção do *overhead* de escrita envolvido quando 120 processos tentam escrever em um mesmo *winchester*.

A diferença de tamanho do arquivo `file.031.1`, em relação aos outros, pode ser explicada pelo fato que esta contingência terminou prematuramente, obedecendo ao critério de parada que encerra a simulação uma vez detectado grandes valores do ângulo de rotor.

Neste exemplo fica ressaltada a característica de independência entre os processos (paralelismo), bem como o assincronismo durante a fase de cálculos. Assincronismo este que ocasiona tempos de simulação diferentes para cada processo, fazendo com que o

Arquivo gerado	Tamanho em bytes
file.000.1	4346
file.001.1	4346
⋮	⋮
file.030.1	4346
file.031.1	1187
file.032.1	4346
⋮	⋮
file.118.1	4346
file.119.1	4346

Tabela 4.1: Tamanho dos arquivos de saída de resultados

Arquivo gerado	Tamanho em bytes
time.000.1	318
time.001.1	318
⋮	⋮
time.118.1	318
time.119.1	318

Tabela 4.2: Tamanho dos arquivos de medida de tempos

processo mais lento seja determinante para definir o tempo total de execução do programa. Porém, na média, os processos tenderam a levar o mesmo tempo de execução, sendo balanceados por natureza. Desta forma, para balancear-se a carga na **VM** optou-se por alocar um mesmo número de processos por *host* (no caso de máquinas homogêneas) e diminuir ou aumentar no caso de máquinas mais lentas ou rápidas, respectivamente.

A medição de tempos para as várias etapas do programa foi de grande importância para mostrar quais estavam sendo ineficientemente paralelizadas e poderiam ser melhoradas. Uma discussão mais detalhada será feita na seção 4.4.

```
# This symbol is used to make comments
# CONFIGURACAO PARA O DSEE
# configuration used for my run of pvm 3.3.5
# location: $HOME

# bellow you put the name of the machines

tambacu
pacu
baiacu
jau
tilapia
bagre
piau
tucunare
maguro
pirarucu
#salmon
```

Figura 4.3: Exemplo de um hostfile

Na Fase 1, diferentes testes foram realizados para possibilitar uma inferência sobre o comportamento da execução do programa, usando-se diferentes tipos de máquinas:

- uma rede de 10 Sparcs modelo 2 conectadas por Ethernet
  (ver **DCE** # 1 na Figura 3.3);
- um *cluster* de 8 RISC/6000 modelo 560 conectadas por **FDDI**
  (ver **DCE** # 2 na Figura 3.4).
- um **SP1** com 8 nós modelo 370 conectadas por **FDDI**
  (ver **DCE** # 3 na Figura 3.5).

Também a partir desses testes investigou-se o número total de processos a serem criados, o qual depende da capacidade do computador usado (em nosso caso limitou-se a 12 processos por máquina, por ser esse um número limite que permite uma execução segura e sem falhas) e que corresponderá ao número de contingências a serem simuladas. Um dos objetivos deste trabalho também foi o de encontrar o número ótimo de casos por *host* que nos levasse a um tempo mínimo de execução de cada processo-filho e, conseqüentemente, ao menor tempo total de execução. Uma discussão mais detalhada será apresentada no Capítulo 5

4.3 Implementação - Fase 2

O diagrama de blocos da Figura 4.4 mostra os diversos passos do programa na Fase 2. Do conhecimento adquirido na Fase 1 foi possível sanar uma série de ineficiências, por exemplo, o acesso a disco, e implementar uma metodologia mais completa para avaliação de segurança que incluiu o cálculo de margens de estabilidade ou instabilidade. Nesta fase cada processo-filho recebe a lista completa de contingências enviada pelo processo-pai, onde cada contingência corresponde a um curto-circuito trifásico, eliminado em diferentes instantes de tempo através do desligamento de linha, ou seja, a eliminação do defeito é feita supondo-se o desligamento de uma linha em um determinado instante de tempo (t_{cl}) durante a simulação. Cada processo simula a contingência correspondente à posição $1 + n_{grp}$ da lista , onde n_{grp} é número do processo no grupo dinâmico.

Após a simulação, que termina assim que um valor de energia cinética mínima for encontrado, cada processo-filho transmite para o processo-pai o resultado da análise de estabilidade de sua contingência: estável ou instável, bem como as margens. Este, por sua vez, processa todas as respostas recebidas e avalia o estado global de segurança: seguro ou inseguro, escrevendo em um único arquivo o resultado final. Desta forma, foi eliminado o dispendioso tempo gasto para escrever os arquivos de saída de cada processo-filho.

Pelo programa exemplo da Figura C, contido no Apêndice C, é possível ter uma noção das rotinas básicas do **PVM** que foram utilizadas no programa desta tese e que se encontram explicadas resumidamente no Apêndice B. Caso o leitor tenha maior interesse as melhores fontes de consulta são as referências [14, 21].

Outra grande diferença nesta fase é que a análise do estado do sistema passou

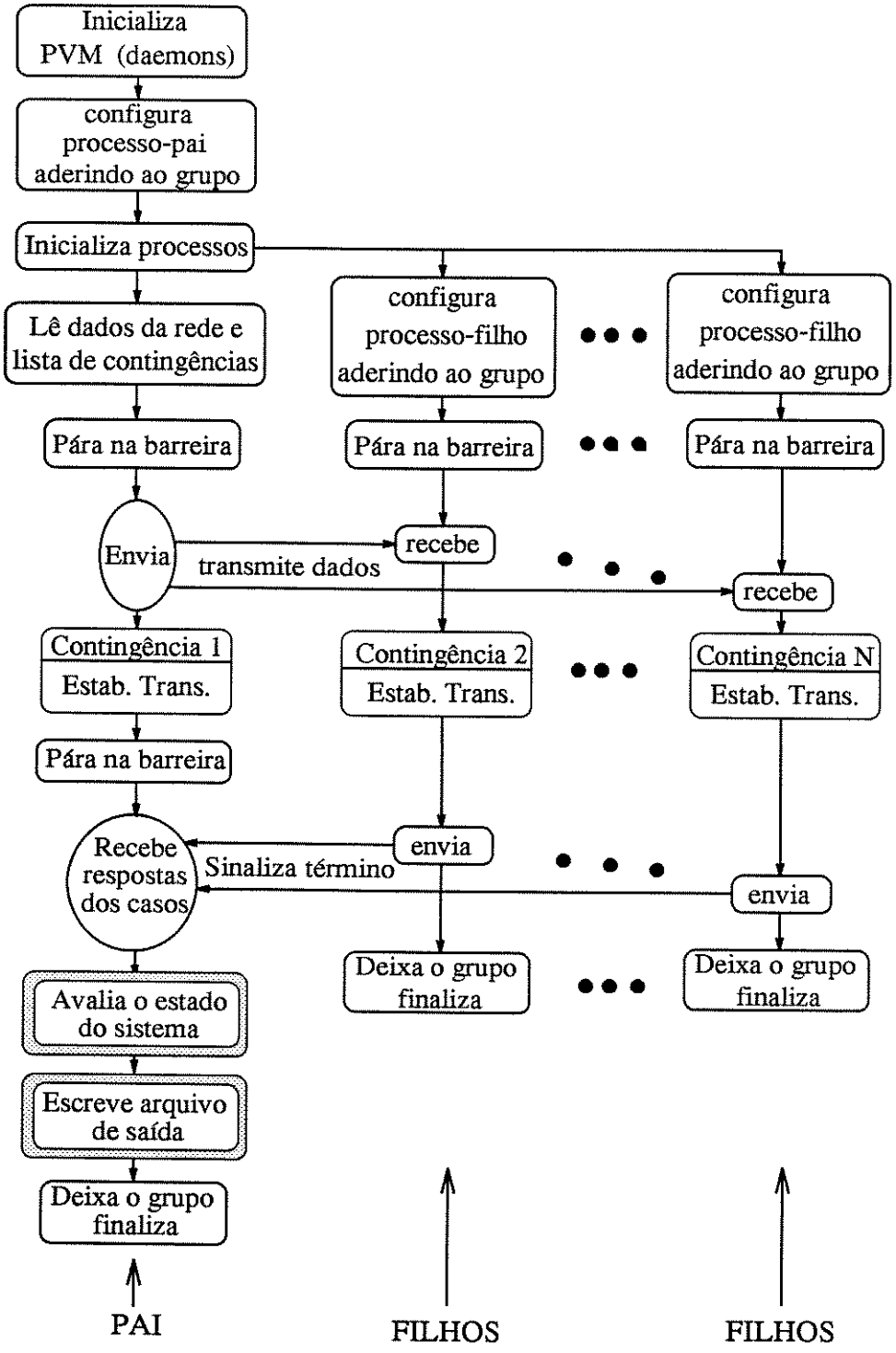

Figura 4.4: Diagrama de blocos contendo detalhes de implementação da Fase 2

a fornecer não apenas dois **estados binários** (estável ou instável), mas incorporou o cálculo de margens de estabilidade ou instabilidade através da análise da função energia transitória no domínio do tempo. Essa abordagem também permitiu interromper a simulação assim que fosse detectado um valor de energia cinética mínima, e portanto agora o intervalo de simulação é variável para cada contingência. Este aspecto, associado ao fato de apenas o processo-pai escrever um arquivo de saída, propiciou enormes ganhos no tempo total de processamento.

Conforme citado no Capítulo 2, o problema de análise de estabilidade transitória pode ser divido em dois estágios independentes, onde os processos-filho ficam responsáveis pelo cálculo de uma contingência que, uma vez concluído, envia a classificação da estabilidade do sistema através de uma variável binária `NSTATE`, onde o valor "0" significa estável e o valor "1" instável. Além desta variável, cada filho transmite também alguns vetores contendo informações para o pai, por exemplo, o seu número no grupo, o valor da energia cinética mínima, as margens de estabilidade ou instabilidade e o número dos geradores avançados, dentre outros. De posse dos dados de todos os processos-filho, o processo-pai, em um segundo estágio, avalia o estado global do sistema para a dada lista de contingências.

Para ilustrar o tipo de informação entregue ao operador, a Figura 4.5 ilustra parte da saída escrita em arquivo pelo processo-pai.

Diferentes tipos de testes foram realizados para medir o desempenho do programa para sistemas reais simulando-se inúmeras contingências e verificando-se a coerência dos resultados obtidos. Dois ambientes de simulação contendo máquinas diferentes foram utilizados nesses testes:

- uma rede de 10 Sparcs modelo 2 conectadas por Ethernet
  (ver **DCE** # 1 na Figura 3.3) ;
- uma rede composta por 8 Sparcs modelo Classic e uma servidora modelo 10 conectadas por **FDDI** através de um *hub*
  (ver **DCE** # 5 na Figura 3.7).

```
==========================================================================
PROGRAMA DE ESTABILIDADE TRANSITORIA - DEPTO. DE SIST. DE ENERGIA ELETRICA/UNICAMP==

NUMERO DE -
---------

BARRAMENTOS                          = 320
LINHAS                               = 470
CARGAS NAO-LINEARES                  =   0
MAQUINAS SINCRONAS                   =  46
SISTEMAS DE EXCITACAO                =  46
REGULADORES DE VELOCIDADE            =   0
MOTORES DE INDUCAO                   =   0
LINHAS MONITORADAS                   =   0
COMPENSADORES ESTATICOS DE REATIVO   =   0
SINAIS ESTABILIZADORES DE POTENCIA   =   0
SINAIS ESTABILIZADORES DOS COMPENSADORES =  0

                    CHAVEAMENTOS ESPECIFICADOS
                    --------------------------

 TEMPO   SWITCH   BUS to BUS   CT     RESISTENCIA   REATANCIA   SUSCEP    TAP
         IN/OUT                NO       P.U.          P.U.

 0.0000  IN      162      0    0      0.00000      0.00000   0.00000   0.0000
 0.3000  OUT     162      0    0      0.00000      0.00000   0.00000   0.0000
 0.3000  OUT     162    163    0      0.09000      0.20960   0.04880   0.0000

Father number:    0    Hostname = pirarucu
EVERY GENERATOR IS STABLE
TSM =  0.413092406    pu
Ecmin =  0.292402206    pu
Timemin = 0.43000000    sec
None Generator is advanced
TSM =  0.413092406    pu

                    CHAVEAMENTOS ESPECIFICADOS
                    --------------------------

 TEMPO   SWITCH   BUS to BUS   CT     RESISTENCIA   REATANCIA   SUSCEP    TAP
         IN/OUT                NO       P.U.          P.U.

 0.0000  IN      190      0    0      0.00000      0.00000   0.00000   0.0000
 0.3000  OUT     190      0    0      0.00000      0.00000   0.00000   0.0000
 0.3000  OUT     190    191    0      0.00000      0.14800   0.15620   0.0000

Son number:    7    Hostname = tambacu
EVERY GENERATOR IS STABLE
TSM =  0.082798578    pu
Ecmin =  0.600488907    pu
Timemin = 0.42000000    sec
None Generator is advanced

                    CHAVEAMENTOS ESPECIFICADOS
                    --------------------------

 TEMPO   SWITCH   BUS to BUS   CT     RESISTENCIA   REATANCIA   SUSCEP    TAP
         IN/OUT                NO       P.U.          P.U.

 0.0000  IN      402      0    0      0.00000      0.00000   0.00000   0.0000
 0.3000  OUT     402      0    0      0.00000      0.00000   0.00000   0.0000
 0.3000  OUT     402    405    0      0.00000      0.03000   7.66580   0.0000

Son number:   35    Hostname = baiacu
EVERY GENERATOR IS STABLE
TSM = -1.868945580    pu
Ecmin = 35.865055176    pu
Timemin = 0.34500000    sec
Gen.    52  advanced    dpc =  56.000449    dpe =  -3.823965

                    CHAVEAMENTOS ESPECIFICADOS
                    --------------------------

 TEMPO   SWITCH   BUS to BUS   CT     RESISTENCIA   REATANCIA   SUSCEP    TAP
         IN/OUT                NO       P.U.          P.U.

 0.0000  IN      503      0    0      0.00000      0.00000   0.00000   0.0000
 0.3000  OUT     503      0    0      0.00000      0.00000   0.00000   0.0000
 0.3000  OUT     503    403    0      0.00000      0.01370   0.00000   1.0000

Son number:   36    Hostname = tilapia
EVERY GENERATOR IS STABLE
TSM = -3.855631072    pu

Ecmin =  8.142531179    pu
Timemin = 0.38000000    sec
None Generator is advanced

                    CHAVEAMENTOS ESPECIFICADOS
                    --------------------------

 TEMPO   SWITCH   BUS to BUS   CT     RESISTENCIA   REATANCIA   SUSCEP    TAP
         IN/OUT                NO       P.U.          P.U.

 0.0000  IN      406      0    0      0.00000      0.00000   0.00000   0.0000
 0.3000  OUT     406      0    0      0.00000      0.00000   0.00000   0.0000
 0.3000  OUT     406    407    0      0.00000      0.01800   1.03540   0.0000

Son number:   38    Hostname = tilapia
EVERY GENERATOR IS STABLE
TSM =  0.000000000    pu
Ecmin =  0.000000000    pu
Timemin = 0.00000000    sec
None Generator is advanced


Simulation Date: Fri Jul 14 22:16:05 1995
There are   10 machines in the configuration
Starting   40 tasks
  40   Time execution to Father: (s) =              205.47

--------------------------------------------------------------------------
```

Figura 4.5: Resultados da execução na Fase 2

4.4 Detalhes sobre a Implementação

Deve ser ressaltado que durante a fase de implementação e testes deste trabalho, houve alterações nos ambientes computacionais utilizados, não só em nível de hardware como de política de utilização das máquinas no **CENAPAD**. Em linhas gerais, as topologias descritas no Capítulo 3 permaneceram as mesmas, entretanto algumas das máquinas foram substituídas por modelos mais novos e poderosos. O próprio programa sofreu algumas alterações que serão comentadas nesta seção, as quais propiciaram ganhos de desempenho no tempo total de execução. Devido a esses fatores, durante a apresentação dos resultados no Capítulo 5 será notado que alguns ambientes distribuídos não puderam ser usados.

4.4.1 Distribuição do arquivo de dados

Um dos aspectos a ser levado em conta pelo leitor é que o problema tratado nesta dissertação envolve algumas particularidades. Para o cálculo de estabilidade transitória foi usado um programa já existente que realiza:

1. a leitura de um arquivo inicial de dados contendo as condições iniciais do sistema elétrico;
2. o cálculo intensivo para resolução de equações algébricas e diferenciais;
3. e a escrita de arquivos de saída com o resultado de cada caso simulado.

Devido a estas características, torna-se importante em ambientes distribuídos de computação, a forma pela qual é feita a distribuição dos dados pelos processos, pois isto pode influenciar o desempenho em termos de tempo total de execução.

Foram testadas as seguintes filosofias de distribuição visando minimizar o tempo gasto com a leitura e a inicialização de dados:

- replicar o arquivo de entrada de dados de maneira a permitir que cada processo leia em um arquivo distinto. Este procedimento, adotado no início da implementação, comprovou não ser uma boa idéia devido ao fato desse arquivo possuir um tamanho razoavelmente grande quando da simulação de sistemas reais (Sul-Sudeste 320 barras), causando uma demora até criar todos os novos arquivos. A Tabela 4.3 mostra os tamanhos de alguns sistemas usados nos testes;

Arquivo de dados	Tamanho em bytes
sist3	1252
sist3n	1573
ss320	54714
ss300m4	58207
ss320exc	78128

Tabela 4.3: Tamanho dos arquivos de dados

- liberar a leitura do arquivo de dados para todos os processos-filho, uma vez que o sistema operacional UNIX permite um acesso simultâneo (concorrente) e assíncrono. Porém, devido ao grande número de processos-filho, o tempo gasto para ler a informação do arquivo de entrada de dados foi muito grande, representando uma parcela considerável em relação ao tempo total de execução.
- permitir que apenas o processo-pai efetuasse a leitura do arquivo de dados, montasse as matrizes do sistema em estudo e, tendo em vista que os dados são iguais para todos os filhos, transmitisse estas informações para todos os processos-filho. Esta foi a melhor solução encontrada para melhorar o desempenho da parte de **leitura e escrita** do programa e, portanto, diminuir o tempo total de execução.

4.4.2 Balanceamento de carga

Devido às características inerentes aos sistemas operacionais baseados em UNIX, quanto maior a memória **RAM** de uma máquina, mais rápido o sistema irá gerenciar a execução de processos, uma vez que não precisará paginar na memória, podendo manter um grande volume de dados na própria **RAM** sem precisar gravar e recuperar do disco rígido. Tentando tirar proveito disto, optou-se por inicializar o maior número de processos por máquina quanto possível, o que deu excelentes resultados, permitindo simular mais casos em um tempo menor. Mas essa filosofia se mostrou inaplicável quando foram usados **ambientes heterogêneos** de computação distribuída contendo máquinas com características de *hardware* muito distintas, o que exigiu o estabelecimento de um critério de balanceamento de carga entre os processadores. Partiu-se então para uma distribuição proporcional de processos por máquina de acordo com a característica de cada ambiente, resumida na Tabela 4.4.

No **DCE** #6 as estações **Sun** se mostraram lentas demais para rodar os seus processos quando comparadas às máquinas **IBM**, pois mesmo colocando-se apenas um processo por máquina na arquitetura **Sun** contra dez processos por máquina na **IBM**, estas ainda assim terminavam primeiro e ficavam paradas aguardando a conclusão dos casos designados para a arquitetura **Sun**. Parece não haver nenhuma vantagem em utilizar ambientes distribuídos muito heterogêneos para resolver um mesmo problema.

No caso de máquinas não tão heterogêneas em desempenho foi realizado um balanceamento na carga dos processadores seguindo um critério heurístico elaborado a partir da Tabela 4.4, onde através de um teste usando uma versão serial do programa obteve-se uma amostra de como cada máquina se comporta. Usando esse critério, durante testes em um **DCE** que agrupou as estações do **DSEE**, foram inicializados quatro processos nas Sparc Classic contra apenas três nas Sparc2, o que permitiu minimizar o tempo em que as máquinas mais rápidas ficavam inativas, fazendo com que todos os processos terminassem aproximadamente ao mesmo tempo. Porém, devido ao fato da rede do **DCE** 5 estar em fase de implantação não foi possível prosseguir nos testes. Havendo um desempenho similar entre todas as suas máquinas, no **DCE** # 4 cada estação recebeu o mesmo número de processos. Apresenta-se ainda, apenas para uma comparação de desempenho, a execução em um PC 486 com 32 Mbytes de RAM e velocidade de *clock* igual a 40 MHz.

Arquitetura	Tempo serial (s)	Tempo paralelo (s)	Carga da máquina
Sparc2	13.31	14.01	0.00
Sparc Classic	11.69	12.35	0.00
Sparc10	5.48	6.11	0.00
RISC6000	2.44	—	1.22
SP1	1.15	—	0.32
PC 486	60.00	—	0.00

Tabela 4.4: Performance em diferentes ambientes

4.4.3 Compilação otimizada

Pensar em computação distribuída de alto desempenho significa ter em mente o objetivo de melhorar o desempenho como um todo da execução de uma aplicação. Dessa

forma, não basta apenas se preocupar em paralelizar o programa, mas também otimizá-lo tanto em nível de implementação do algoritmo e dos paradigmas de programação quanto em nível de compilação e "linkagem" , procurando tirar o máximo possível do *hardware* disponível. Para tanto, uma pesquisa no manuais dos compiladores [28, 27, 29, 26, 25, 24] visando escolher opções de compilação otimizadas, permitiu ganhos significativos no tempo total de execução, qualquer que seja o ambiente de máquinas utilizado, como pode ser visto na Tabela 4.5.

Tempo	Cluster IBM		
(segundos)	default	otimizado	ganho %
leitura	1.46	1.41	3.42
cálculos	58.36	39.95	31.55
escrita	0.77	0.47	38.96
total	60.59	41.84	30.95
	Sparc 2		
	default	otimizado	ganho %
leitura	8.81	8.10	8.06
cálculos	7.97	5.13	35.63
escrita	1.35	0.86	36.30
total	18.05	13.30	26.32
	Sparc Classic		
	default	otimizado	ganho %
leitura	6.40	4.72	26.25
cálculos	7.02	4.60	34.47
escrita	1.05	0.54	48.57
total	14.45	9.88	31.63

Tabela 4.5: Comparação entre as diversas opções de compilação

Lista-se abaixo as principais opções de otimização usadas na compilação para criar o programa executável desenvolvido nesta dissertação.

- `-O2` otimização dos *loops* do programa;
- `-dalign` padroniza as variáveis do programa;
- `-cg89` ajusta operações aritméticas de acordo com o processador de ponto flutuante instalado.

4.5 Portabilidade do Programa

Uma das vantagens da utilização de rotinas de troca de mensagens do tipo fornecido pelo **PVM** é que elas tornam o programa altamente portável entre arquiteturas diferentes (RISC 6000 - **IBM**, **Sun**, **PC**). A única dificuldade reside nas subrotinas ou funções que efetuem chamadas de tempo no sistema ou então que venham a usar um recurso computacional específico do *hardware* da arquitetura em questão.

Neste trabalho, as únicas funções que apresentaram incompatibilidade foram:

1. medição de tempo;
2. escrita em arquivos (alguns dos atributos do `WRITE` do Fortran 77 variaram da RISC 6000 em relação ao PC 486 e a **Sun**);
3. o local (*PATH*) onde se encontram instaladas as rotinas do **PVM** variam entre as arquiteturas exigindo alteração no código fonte dos programas;

Outro fato a salientar é que para cada ambiente de arquitetura diferente é necessário compilar e "linkar" todos os programas e rotinas no próprio ambiente.

Capítulo 5

Resultados em Diferentes Ambientes Distribuídos

5.1 Introdução

O objetivo deste capítulo é apresentar ao leitor uma série de resultados mostrando o comportamento do programa nos diferentes ambientes distribuídos já descritos no Capítulo 3. Cada um destes ambientes, por possuir diferentes características de *hardware*, proporcionou diversas possibilidades de testes com resultados bem particulares.

Durante os testes buscou-se encontrar o número máximo de processos que deveria ser alocado em cada *host*. Os conceitos de mono e multi-programação encontrados em [6, 7] sugerem alocar tantos processos quanto possível numa máquina, até um limite a partir do qual seria necessário ao sistema operacional efetuar "*paging*" e "*swap*", salvando um processo no disco rígido e recuperando para a memória, de forma a efetuar os processos em compartilhamento de tempo. Porém, para a aplicação de computação distribuída desenvolvida neste trabalho, notou-se que a paginação e o "*swap*" podem ser vantajosos, pois os processos são todos disparados no início da execução do processo-pai, que efetua a leitura dos dados e através de um *broadcast* os envia para os processos-filho. Ou seja, os processos-filho de uma máquina são criados de uma só vez e ficam disputando a CPU da máquina. Isto é melhor do que inicializar cada processo-filho separadamente e esperar a liberação da CPU.

Dependendo do sistema operacional utilizado para gerenciar a rede de computadores, o número de processos que podem ser abertos em uma mesma estação pode variar. No caso do SunOS 5.3 o número máximo de processos que podem ser abertos por máquina é menor que no SunOS 4.1.1, devido ao fato de o primeiro reservar uma parcela maior da memória disponível para o sistema operacional.

O tamanho da tabela de processos do sistema bem como a quantidade de memória disponível em cada máquina também são fatores de grande influência no número máximo de processos. Quanto mais memória **RAM** disponível, maior será o número de processos que poderão ser abertos naquele *host* sem que haja necessidade de paginação ou "swap", conforme explicado em [6].

Diante disso, o número ideal de processos a serem alocados em cada máquina, ou seja, o número de processos que resulta em um melhor desempenho tomando-se a relação entre o tempo total de computação e o número de contingências a analisar, também variou entre os diversos **DCE** , o que será mostrado no decorrer deste capítulo.

Inicialmente foram feitas simulações utilizando um sistema com três barras para facilitar a verificação da confiabilidade dos resultados obtidos e a calibração do programa. A seguir foram utilizados dois sistemas elétricos reais:

1. Sistema Sul-Sudeste com 320 barras, 470 linhas e 46 geradores representados por modelo clássico (ss320);
2. Sistema Sul-Sudeste com 320 barras, 470 linhas e 46 geradores representados por modelo detalhado, incluindo sistema de excitação (ss320exc).

Os modelos descritos acima resultaram em tempos de computação bem diferentes, uma vez que o modelo detalhado envolve muito mais cálculos e portanto um tempo bem maior de processamento.

Sabe-se que um dos "gargalos", quando se trata de computação de alto desempenho, é o tempo de leitura e escrita no disco rígido. À medida que a velocidade dos processadores aumenta o tempo gasto para processar operações lógicas e aritméticas decresce, ao passo que o aumento na velocidade de leitura e escrita em disco fica limitado devido aos movimentos mecânicos envolvidos nestas operações. Logo, se o leitor comparar os resultados dos diferentes **DCE** irá comprovar que foi possível, usando máquinas rápidas,

diminuir o tempo de cálculo, porém o tempo de escrita e leitura não variou tanto de um ambiente para outro.

Em todos os resultados apresentados neste capítulo foram feitas médias para cinco execuções sucessivas do programa, de forma a minimizar a influência de alterações bruscas na carga do ambiente, pois estes são compartilhados por diversos usuários. Dessa forma, nem sempre a soma dos tempos de leitura, mais inicialização, mais cálculos e mais escrita dará um valor igual ao tempo total apresentado.

5.2 Resultados da Fase 1

Conforme descrito no Capítulo 4, na Fase 1 da implementação optou-se por deixar cada processo escrever, em arquivos individuais, o seu respectivo tempo de execução e a sua resposta para a contingência simulada. Além disso, outro fator que levou a elevados tempos de computação e **I/O** foi manter fixo o intervalo de tempo de simulação em 1.11 segundos para o modelo clássico e 5 segundos para o modelo detalhado, utilizando um passo de integração de 20 ms para o modelo clássico e de 5 ms para o modelo detalhado.

5.2.1 Resultados para o DCE # 1: Sparc2 com Ethernet

Os resultados obtidos para os dois sistemas simulados apresentaram tempos bem diferentes, como era de se esperar, e o pequeno poder computacional deste **DCE** leva o tempo total de solução a ser muito grande. Na Tabela 5.1 estão mostrados os resultados da simulação de ambos os modelos. Optou-se por variar o número de *hosts* e verificar o desempenho do programa, plotando-se os gráficos das Figuras 5.1 e 5.2 para facilitar a visualização dos resultados.

Através dos gráficos das Figuras 5.1 e 5.2 pode-se perceber que quanto maior o número de *hosts* configurados, maior será o número de casos simulados (contingências) para um mesmo intervalo de tempo. Nota-se também que, à medida que mais *hosts* vão sendo usados, a redução no tempo de solução vai saturando, o que é característico do processamento paralelo.

O limite de 120 casos (12 casos x 10 *hosts*) foi imposto pelo *hardware* deste **DCE**, pois não se deve esquecer que além dos 12 casos por máquina criados pelo **PVM** existem

Casos Simulados	Tempo decorrido (segundos)				
	2 Hosts	4 Hosts	6 Hosts	8 Hosts	10 Hosts
2	19.78				
4	31.04	21.05			
6			21.52		
8	51.71	32.61		25.06	
10					25.35
12			33.72		
16	93.02	53.03		36.91	
20					39.14
24	129.65		55.08		
32		96.80		58.93	
40					60.40
48		137.76	99.92		
64				106.34	
72			142.28		
80					107.85
96				161.01	
120					167.78

Tabela 5.1: Tempos para o modelo clássico no DCE #1

Casos Simulados	Tempo decorrido (segundos)				
	2 Hosts	4 Hosts	6 Hosts	8 Hosts	10 Hosts
2	489.18				
4	911.63	593.94			
6			544.17		
8	1505.31	903.08		544.61	
10					609.50
12			904.29		
16	2706.02	1505.39		908.87	
20					908.79
24	3492.84		1512.63		
32		2716.01		1454.74	
40					1492.53
48			2850.73		
64				2422.90	
72			3790.51		
80					2750.49
96				3767.40	
120					3838.78

Tabela 5.2: Tempos para o modelo detalhado no DCE # 1

os referentes ao sistema operacional (por exemplo *daemons* de *email*, impressão, etc) e os processos de outros usuários, que uma vez somados podem atingir o número máximo de processos permitidos na estação. Este limite de 12 processos/máquina visa assegurar que o **PVM** consiga abrir os seus processos na máquina.

Em todos os testes apresentados para este **DCE** foi possível tornar o ambiente distribuído dedicado somente à execução do programa, pois este foi executado durante a madrugada com todas as máquinas livres (sem qualquer usuário ou processos em *batch*), disputando a **CPU** apenas com os *daemons* do sistema operacional.

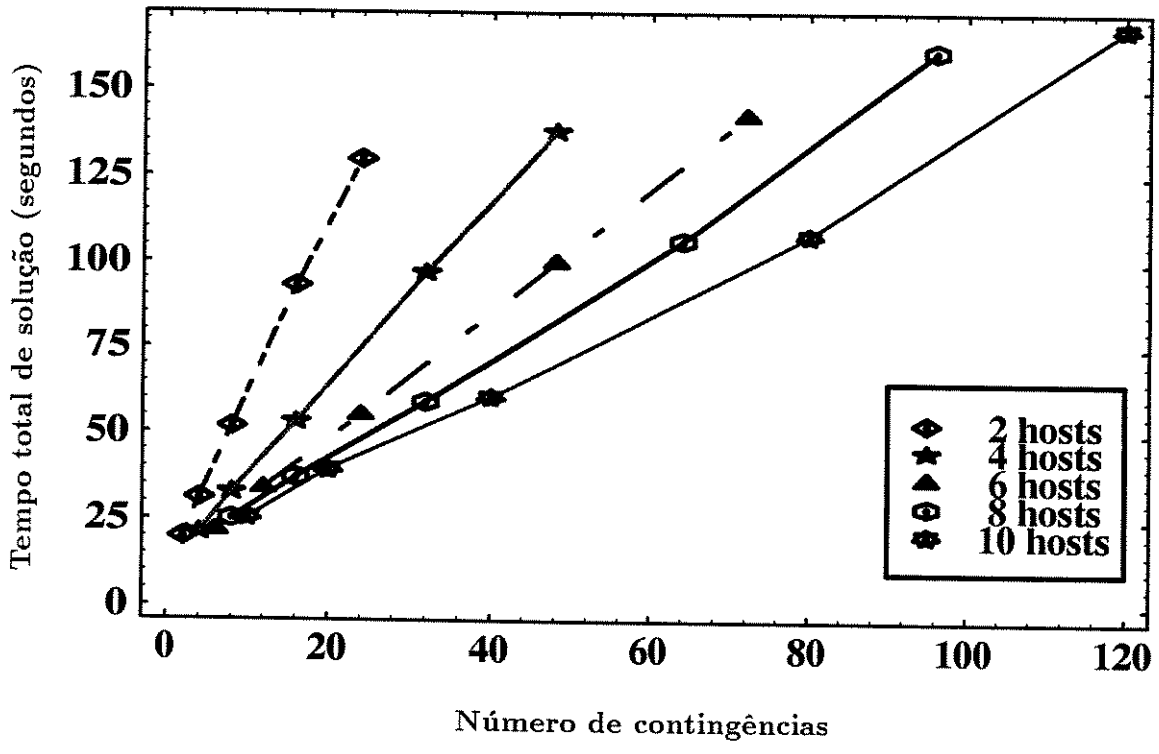

Figura 5.1: Performance no DCE # 1 usando o modelo clássico

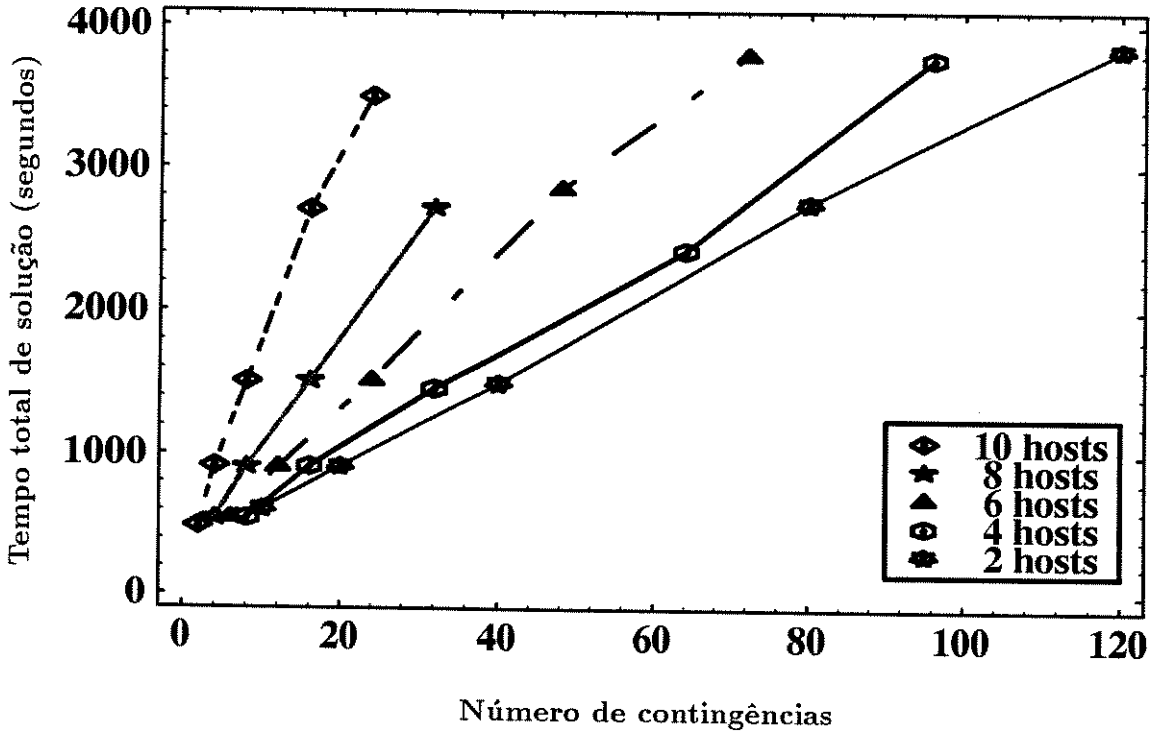

Figura 5.2: Performance no DCE # 1 usando o modelo detalhado

5.2.2 Resultados para o DCE # 2: Risc6000 com FDDI

Este ambiente possui *hosts* e rede bem mais rápidos que o **DCE** #1. Pode-se notar, através das Tabelas 5.3 e 5.4, que o tempo referente à simulação de 96 contingências foi 2.3 vezes menor para o modelo clássico em relação ao **DCE** # 1. Quanto ao modelo detalhado, a diferença foi de 4.54 vezes. Por esses valores vê-se que o ganho computacional da solução distribuída tende a ser mais elevado ainda quando se usam máquinas de alto desempenho, já que a maior parte do tempo é gasto em cálculos de ponto flutuante.

Casos Simulados	Tempo decorrido (segundos)			
	2 Hosts	4 Hosts	6 Hosts	8 Hosts
2	9.89			
4	12.48	11.39		
6			11.92	
8	17.68	15.91		11.94
12			13.87	
16	27.84	18.32		13.72
24	36.04		17.93	
32		27.03		19.74
48		42.82	30.39	
64				44.98
72			50.84	
96				69.88

Tabela 5.3: Tempos para o modelo clássico no DCE # 2

Observando o gráfico da Figura 5.3 nota-se que a curva referente a oito *hosts* toca a curva de seis *hosts*. Isto não era um resultado esperado, pois conforme já visto, quanto mais *hosts* melhor o desempenho da solução distribuída. A explicação reside no fato de que, neste caso particular, as máquinas estavam sendo compartilhadas por vários usuários executando diferentes processos, o que afeta o desempenho do programa e leva a resultados um pouco piores do que os teoricamente esperados.

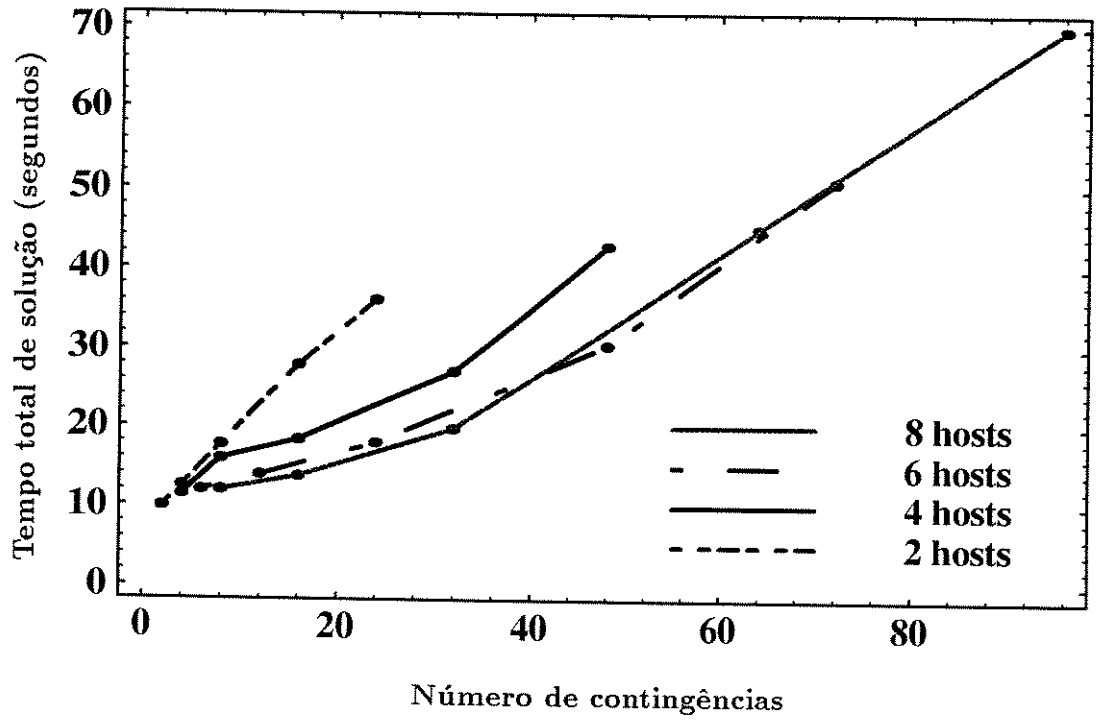

Figura 5.3: Performance no DCE # 2 usando o modelo clássico

Casos Simulados	Tempo decorrido (segundos)			
	2 Hosts	4 Hosts	6 Hosts	8 Hosts
2	196.06			
4	260.55	196.59		
6			203.37	
8	363.63	269.16		205.26
12			262.12	
16	610.56	381.90		263.40
24			363.83	
32		690.87		381.00
48			601.83	
64				585.06
72			784.12	
96				828.62

Tabela 5.4: Tempos para o modelo detalhado no DCE # 2

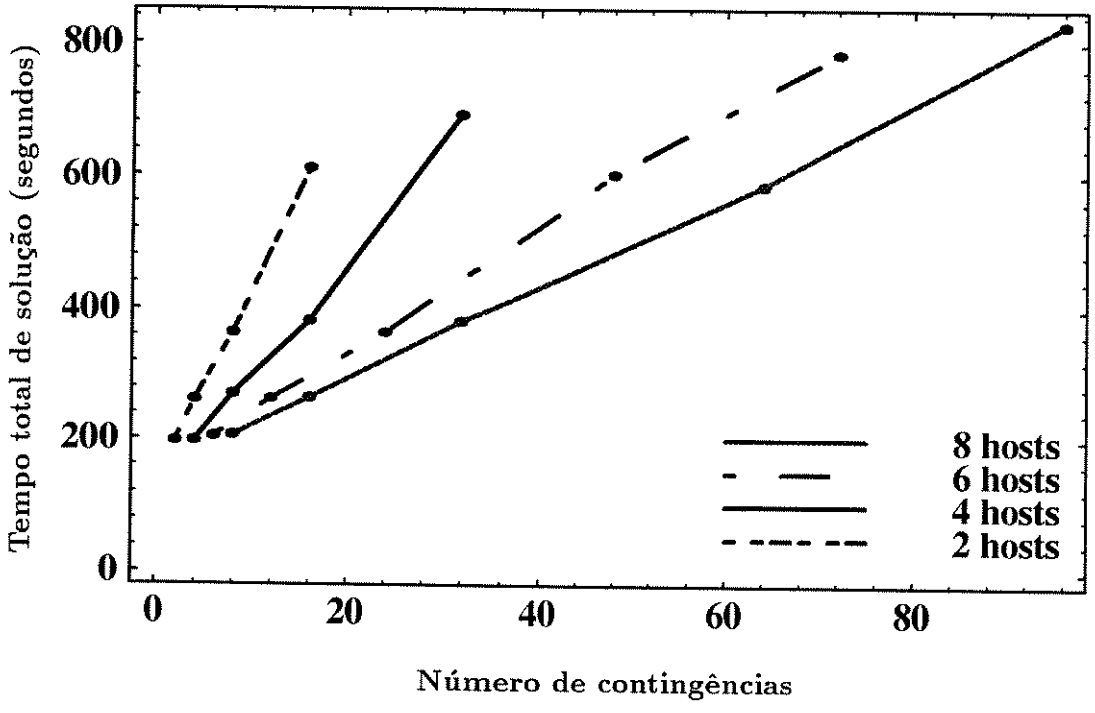

Figura 5.4: Performance no DCE # 2 usando o modelo detalhado

5.2.3 Resultados para o DCE # 3: SP1 com FDDI

Neste teste, a carga da rede pode ser vista na Tabela 5.5, através da qual pode-se ter uma noção do comportamento das máquinas quando em uso compartilhado. Através do comando UNIX rup obtém-se a informação sobre a carga média do processador de determinada máquina da rede. Esses valores referem-se à taxa de utilização da máquina produzida pelos processos executados no último minuto. Considerou-se valores de carga média inicial e final aqueles medidos antes e após a execução do programa, respectivamente.

Máquina	Carga média inícial	Carga média final
cenapad	2.52	4.25
n01.cna.unicamp.br	6.00	7.20
n02.cna.unicamp.br	5.02	7.54
n03.cna.unicamp.br	5.05	7.10
n04.cna.unicamp.br	4.97	6.55
n05.cna.unicamp.br	5.99	7.82
n06.cna.unicamp.br	5.92	7.52
n07.cna.unicamp.br	4.00	5.83
n08.cna.unicamp.br	4.07	6.02
alamo.cna.unicamp.br	2.25	4.03
frejo.cna.unicamp.br	2.36	4.52
imbuia.cna.unicamp.br	1.94	4.09
mogno.cna.unicamp.br	5.10	7.77
peroba.cna.unicamp.br	2.18	4.76
pinho.cna.unicamp.br	4.65	6.01
delos.cna.unicamp.br	2.17	5.48

Tabela 5.5: Carga média das máquinas

Ao analisar os resultados o leitor deve ter em mente que as ações de leitura, comunicação e escrita são naturalmente seriais, sendo que apenas o processamento dos cálculos é paralelizado.

Dessa forma no **DCE # 3**, conforme mostrado na Tabela 5.6, para 8 *hosts* e 96 contingências, o tempo total foi de 47.22 segundos, enquanto que para oito contingências foi de 10.32 segundos. Mantendo-se o número de *hosts* igual a oito e aumentando o número de

casos em doze vezes, o esperado seria um tempo igual a 123.84 segundos ($12 * 10.32$), logo o ganho foi de 262.26 %. Isto se deve à paralelização dos cálculos de ponto flutuante e à forma de distribuição dos dados iniciais através de apenas uma sessão de troca de mensagens entre os processos-filho e o processo-pai.

Casos Simulados	Tempo decorrido (segundos)			
	2 Hosts	4 Hosts	6 Hosts	8 Hosts
2	6.68			
4	8.76	8.75		
6			9.15	
8	11.61	9.65		10.32
12			9.80	
16	18.18	13.79		13.98
24	23.70		14.23	
32		19.34		18.76
48		29.31	23.18	
64				30.93
72			34.41	
96				47.22

Tabela 5.6: Tempos para o modelo clássico no DCE # 3

A Figura 5.5 ilustra graficamente o resultado de diversas simulações do programa, onde variam o número de *hosts* e o número de contingências a serem analisadas. As curvas referentes a um maior número de *hosts* se encontram abaixo das outras curvas, tendo portanto um valor menor para o tempo total de execução. Isto demonstra a grande vantagem do uso de computação distribuída para análise de segurança dinâmica.

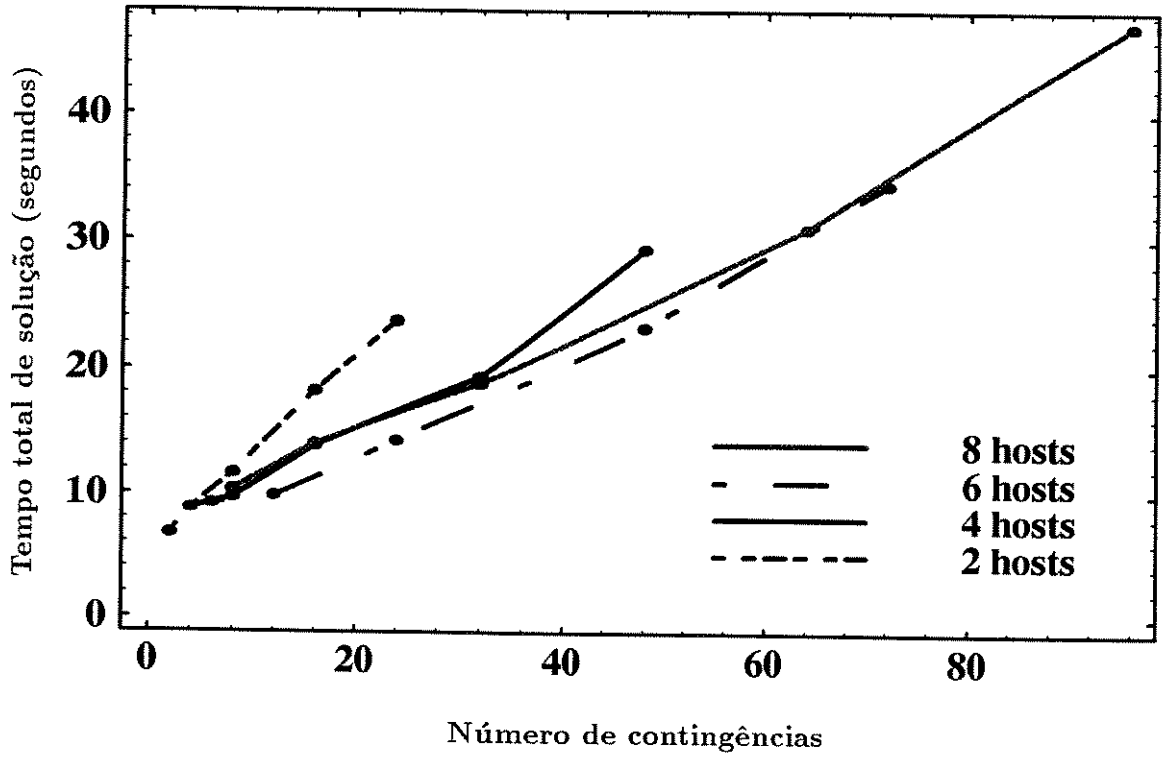

Figura 5.5: Performance no DCE # 3 usando o modelo clássico

Ao analisar os resultados da Tabela 5.7, para 8 *hosts* e 96 contingências no **DCE # 3**, vê-se que o tempo total foi de 556.58 segundos, enquanto que para oito contingências foi de 129.44 segundos. Mantendo-se o número de *hosts* igual a oito e aumentando o número de casos em doze vezes, o esperado seria um tempo igual a 1553.28 segundos (12*129.44), pois cada processo estaria disputando a **CPU** da máquina, no entanto obteve-se um ganho de 279.08 %. Isto se deve à paralelização dos cálculos de ponto flutuante e à forma de distribuição dos dados iniciais através de apenas uma sessão de troca de mensagens entre os processos-filho e o processo-pai, aliado a capacidade de gerenciamento de processos do sistema operacional **UNIX**, que mostrou ser vantajoso utilizar mais de um processo concorrente em um mesmo processador.

Casos Simulados	Tempo decorrido (segundos)			
	2 Hosts	4 Hosts	6 Hosts	8 Hosts
2	129.44			
4	178.96	140.18		
6			141.59	
8	253.13	179.39		146.25
12			180.92	
16	391.02	244.43		181.77
24	531.87		245.59	
32		369.76		257.27
48		535.56	407.83	
64				411.84
72			542.65	
96				556.58

Tabela 5.7: Tempos para o modelo detalhado no DCE # 3

A Figura 5.6 mostra graficamente como se comporta o programa a medida que se variam o número de *hosts* e o número de contingências a serem analisadas. Vê-se que a medida que o número de *hosts* aumenta, as curvas têm um valor menor para o tempo total de execução.

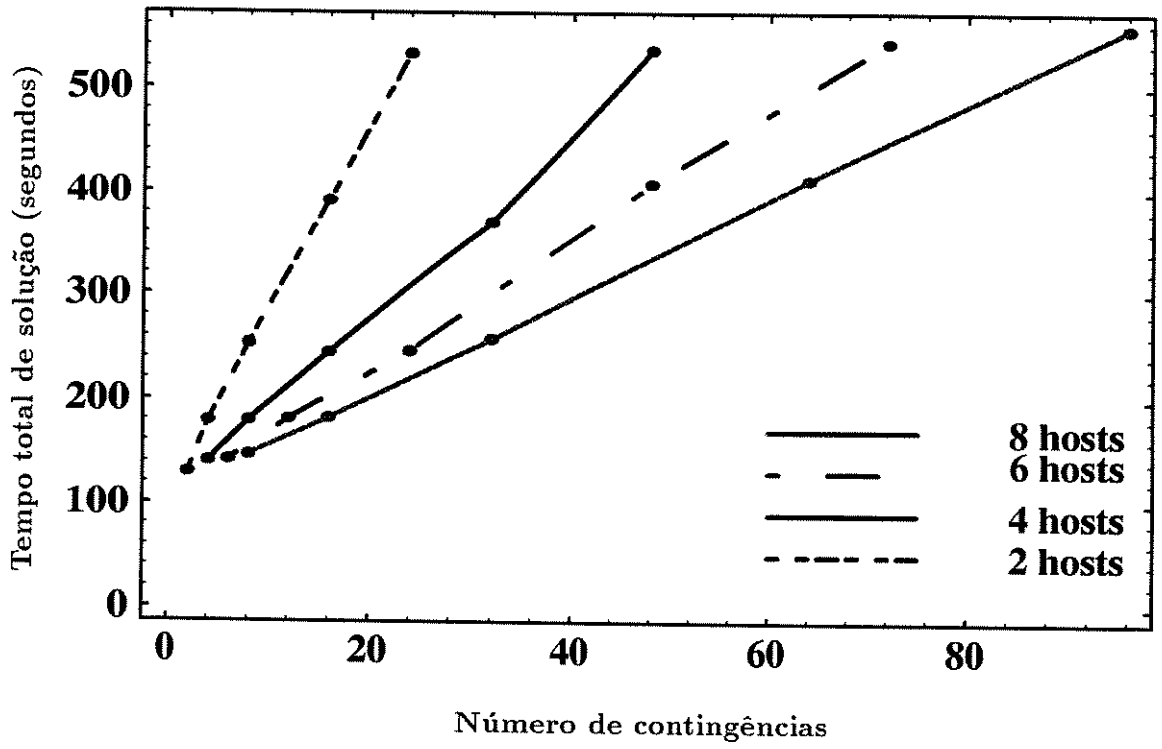

Figura 5.6: Performance no DCE # 3 usando o modelo detalhado

5.2.4 Resultados para o DCE # 4: Risc6000 + SP1

Este ambiente foi aquele que apresentou os melhores resultados no que se refere ao tempo total de execução, o que é plenamente explicável dadas as suas características de *hardware*: possui o maior número de máquinas dentre todos os ambientes distribuídos utilizados, além dos *hosts* e da rede serem mais rápidos.

A Tabela 5.8 mostra os resultados obtidos no **DCE # 4** com 16 *hosts* e um limite de 12 processos por *host*, o que permite analisar cerca de 192 contingências em aproximadamente 5 minutos. A Figura 5.7 ilustra esses resultados, permitindo também comparar a diferença nos tempos de execução quando são usados os modelos clássico e detalhado. Isto já era esperado devido ao maior número de operações de ponto flutuante envolvidas quando se utiliza o modelo detalhado. Estes resultados constam no artigo apresentado no congresso *Sixth Annual Conference of the Power & Energy Society of the IEE Japan* [19].

Número de Casos	Tempo total (segundos)	
	Modelo Clássico	Modelo Detalhado
16	4.59	100.76
32	12.04	101.02
64	16.49	143.96
96	18.16	187.10
128	20.95	261.20
192	27.42	301.21

Tabela 5.8: Tempo total de simulação (s) para o DCE # 4 usando 16 hosts

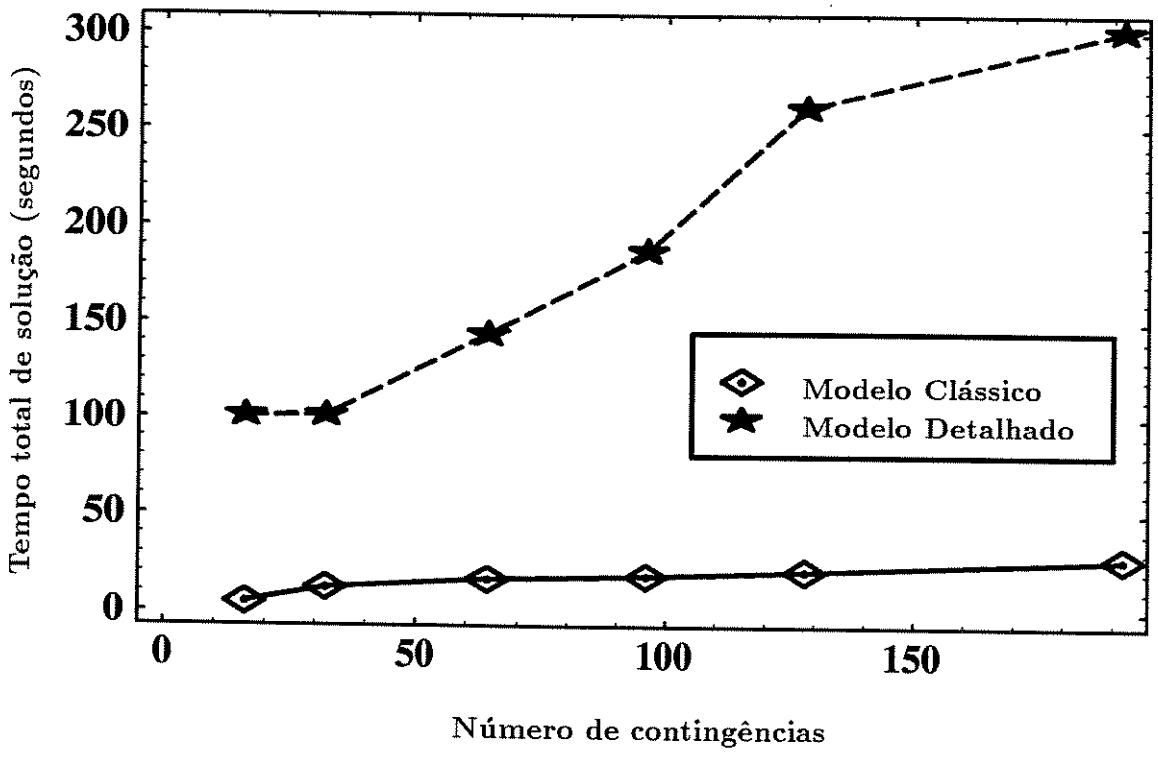

Figura 5.7: Performance no DCE # 4 com 16 hosts

Pelos gráficos apresentados nas Figuras 5.8 e 5.9 pode-se verificar o comportamento de cada **DCE** no que se refere a tempos de leitura do arquivo de dados, envio de dados para os processos-filho, computação em si e escrita dos arquivos de saída. Dependendo do ambiente de computação, o tempo gasto para cálculos é em geral bem maior do que os outros tempos envolvidos na execução do programa. No **DCE** # 1, conforme pode ser visto pela Figura 5.9 quando utilizados tanto o modelo clássico quanto o detalhado, os tempos de computação foram tão elevados que suplantaram os outros tempos.

Pelas Tabelas 5.9 e 5.10 pode-se verificar que os tempos nas estações da **Sun** foram bem maiores do que nas **IBM**, pois, devido a existência de um processador mais rápido, o tempo de cálculo em ponto flutuante é menor e as demais parcelas se tornaram significativas. Isto permite concluir que no futuro, com o aumento da velocidade dos microprocessadores, o "gargalo" da computação distribuída irá se concentrar no meio físico e no protocolo pelo qual é feita a comunicação entre as diversas máquinas.

Tanto no **DCE** # 2 quanto no **DCE** # 3, quando foi usado o modelo clássico, obteve-se um tempo de recepção (64 %) muito grande que suplantou o tempo de cálculo (25 %). Isto denota uma característica de granularidade fina, ou seja, os cálculos demandam um tempo pequeno em relação ao gasto na comunicação envolvida na execução distribuída do programa. Quando se utilizou o modelo detalhado esse problema desapareceu, obtendo-se uma melhor paralelização com o crescimento da granularidade.

Modelo Clássico				
Operação Executada	Tempo decorrido (segundos)			
	DCE 1	DCE 2	DCE 3	DCE 4
recepção de dados	7.32	34.08	33.43	8.00
abertura de arquivos	18.92	2.24	1.63	1.30
cálculos iterativos	87.05	13.10	13.22	4.44
escrita de resultados	13.51	3.52	3.41	1.132

Tabela 5.9: Discriminação dos tempos envolvidos na solução em diversos ambientes

Modelo Detalhado				
Operação Executada	Tempo decorrido (segundos)			
	DCE 1	DCE 2	DCE 3	DCE 4
recepção de dados	21.61	29.23	40.25	38.258
abertura de arquivos	3.82	2.04	2.47	2.040
cálculos iterativos	3332.89	513.86	489.12	166.643
escrita de resultados	15.36	3.98	3.98	1.531

Tabela 5.10: Discriminação dos tempos envolvidos na solução em diversos ambientes

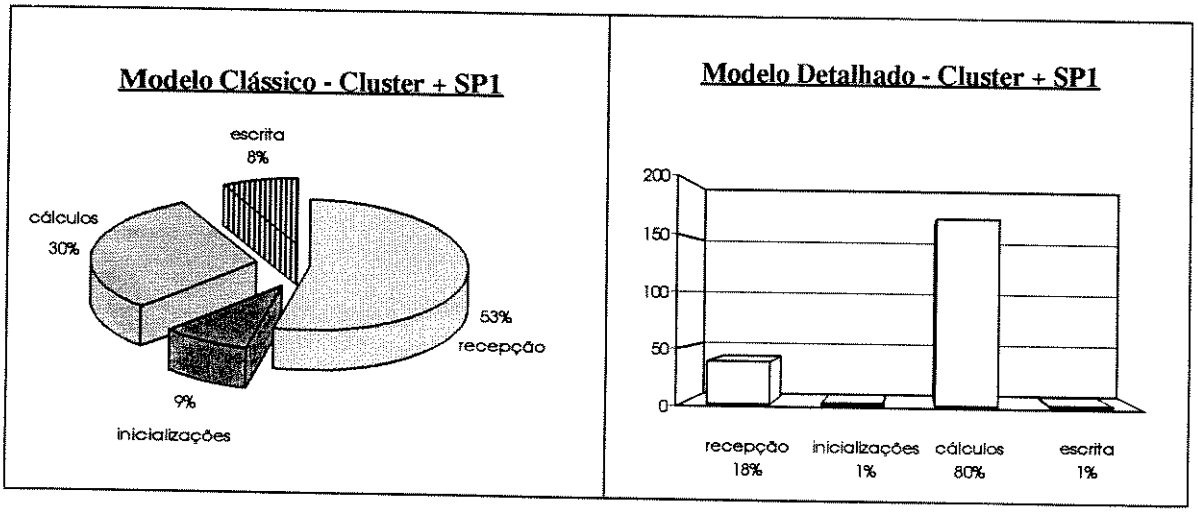

Figura 5.8: Histograma de tempos usando o modelo detalhado no DCE # 4

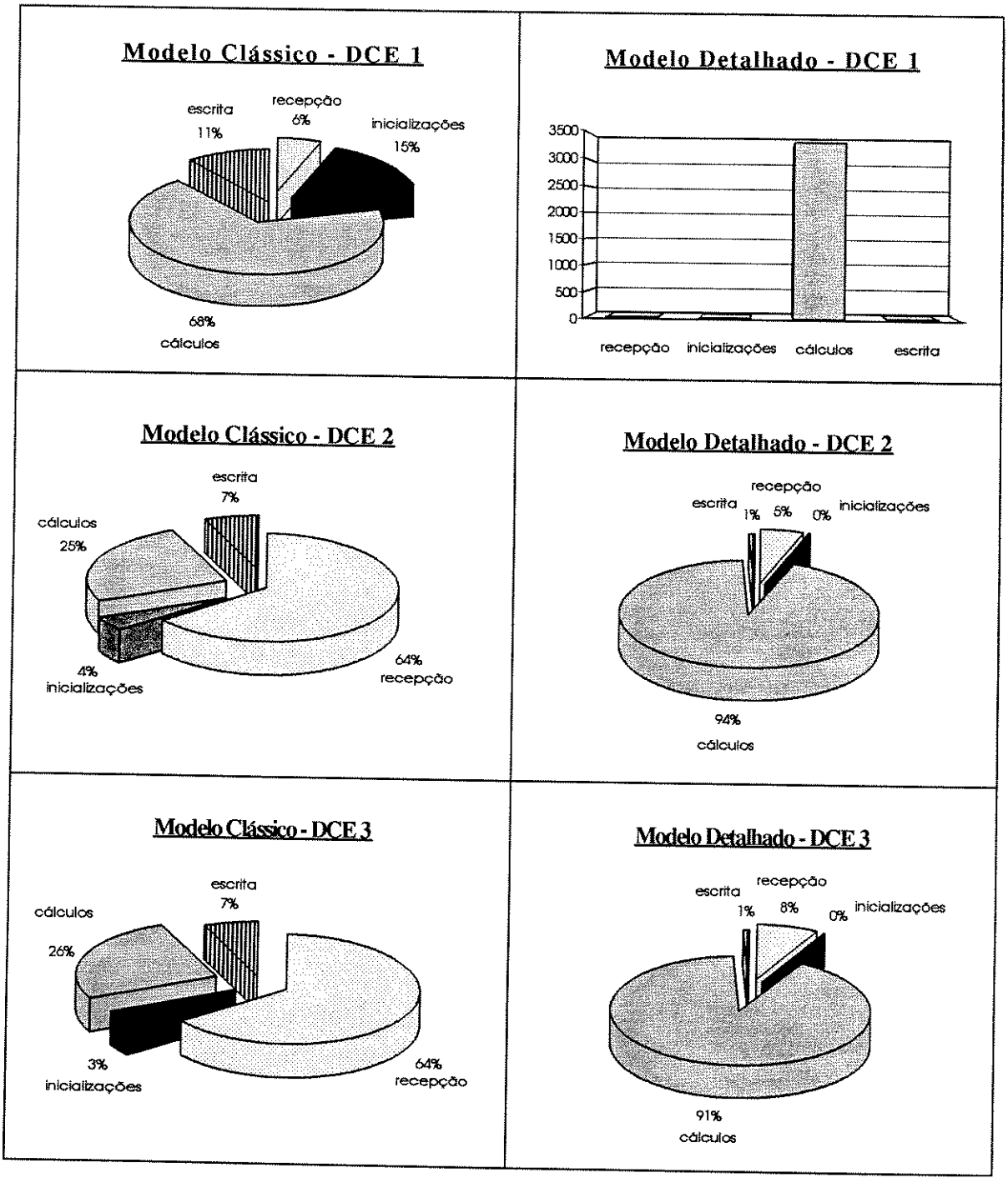

Figura 5.9: Gráfico de tempos para diversos ambientes

5.2.5 Resultados para o DCE # 6

O agrupamento de todas as estações disponíveis em uma única máquina virtual, conforme mostrado na Figura 3.8, não apresentou bons resultados. A razoável distância física que separa algumas máquinas (aproximadamente 600 metros), o compartilhamento com outros usuários e o intenso tráfego de dados na rede de interconexão perturbou o gerenciamento dos processos feito pelo **PVM**. Este, não conseguindo manter um canal de comunicação entre os diversos *daemons*, efetua a chamada da rotina `hostfailentry()`, a qual procura numa lista (*waitlist*) contendo todas as operações relacionadas com cada *daemon*. Como o *daemon* perdeu conexão, `hostfailentry()` encerra as operações. Em outras palavras, eram inicializados em torno de 300 processos, porém com a perda de comunicação entre os processos *daemons* e o processo central de controle do **PVM** , ocorria o desligamento de todos os processos-filho relacionados àquele *daemon* e o processo-pai já não mais aguardava na barreira pelo término da execução de todos os filhos, ficando incompleta a avaliação de todas as contingências da lista. Este é um problema difícil de ser contornado pois é originado por problemas físicos de conexão. Talvez o uso de redes FDDI ou *Fast-Ethernet* fosse uma alternativa.

Outra dificuldade encontrada foi conciliar as diferenças de desempenho das máquinas. Para tanto, a princípio tentou-se uma distribuição de um maior número de processos (15) nas máquinas **IBM** e poucos (3) nas **Sun**. Isto foi inadequado pois mesmo quando colocou-se apenas um único processo por *host* nas **Sun** contra quinze nas **IBM**, estas terminavam sua execução primeiro e ficavam paradas esperando o término das outras máquinas.

Estes fatores parecem indicar uma restrição ao uso de ambientes muito heterogêneos e fisicamente dispersos para este tipo de aplicação.

5.3 Resultados da Fase 2

Nos testes realizados na Fase 2 da implementação, os processos-filho realizam a avaliação de uma contingência e transmitem os resultados para o processo-pai, que se encarrega de avaliar o estado global do sistema para uma dada lista de contingências e escreve um arquivo único contendo os resultados finais. Utilizou-se o modelo detalhado do Sistema Sul-Sudeste brasileiro (320 barras) em todos os testes realizados, pois o tempo de simulação de cada contingência era apenas um pouco maior do que quando se usava o modelo clássico, porém com um nível de representação bem mais próximo da realidade.

Devido às mudanças ocorridas no CENAPAD não foi possível efetuar novos testes com esta versão do programa pois o acesso interativo às máquinas do *cluster* e do **SP1** foi desabilitado. Dessa forma, os testes apresentados se restringiram aos ambientes do laboratórios do Departamento de Sistemas de Energia Elétrica (**DSEE**) da UNICAMP.

5.3.1 Resultados para o DCE # 1: Sparc2 com Ethernet

Analisando-se a Tabela 5.11 nota-se que o tempo gasto por processo, ou seja, a razão entre o tempo total de simulação e número de casos (contingências) analisados, decresce à medida que aumenta o número de processos concorrentes em uma mesma **CPU**, mas vai saturando, como era de se esperar. Isto pode ser visto através da Figura 5.10.

Na Tabela 5.11, as duas últimas colunas da direita, que contêm informações sobre o *speedup*, mostram um ganho crescente, mesmo estando incluídos os tempos de comunicação, à medida que cresce o número de casos a serem analisados. Nessa tabela, os *speedups* são obtidos comparando-se o tempo total de simulação usando vários processos com o tempo gasto para executar somente o processo-pai (penúltima coluna) e com o tempo gasto para executar um programa serial (última coluna). Deve ser ressaltado que o número de *hosts* foi mantido fixo em 10 máquinas. A Figura 5.11 ilustra o comportamento do *speedup* graficamente.

Número de Casos	Tempo Total	Tempo por processo	Speedup proc. paralelo	Speedup proc. serial
40	207.08	5.18	14.78	15.56
60	247.82	4.13	17.69	18.62
80	318.68	3.99	22.75	23.94
100	373.28	3.73	26.64	28.05
120	443.72	3.70	31.67	33.34

Tabela 5.11: Resultados para o DCE # 1 na Fase 2 com 10 hosts

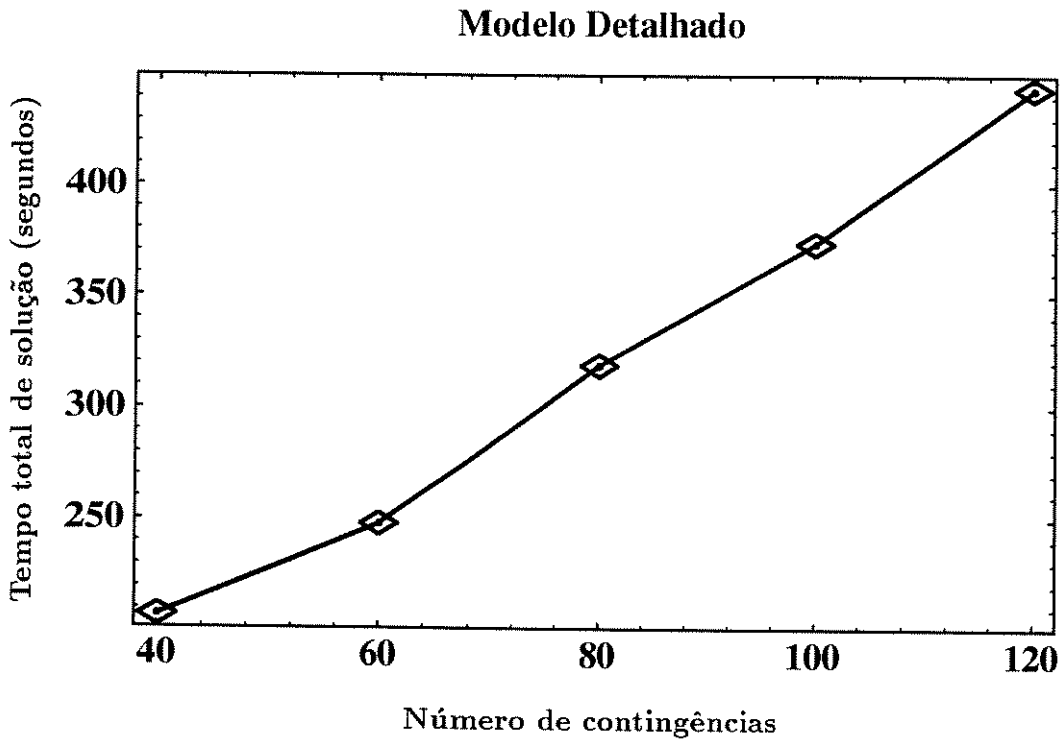

Figura 5.10: Performance no DCE # 1 na Fase 2 com 10 hosts

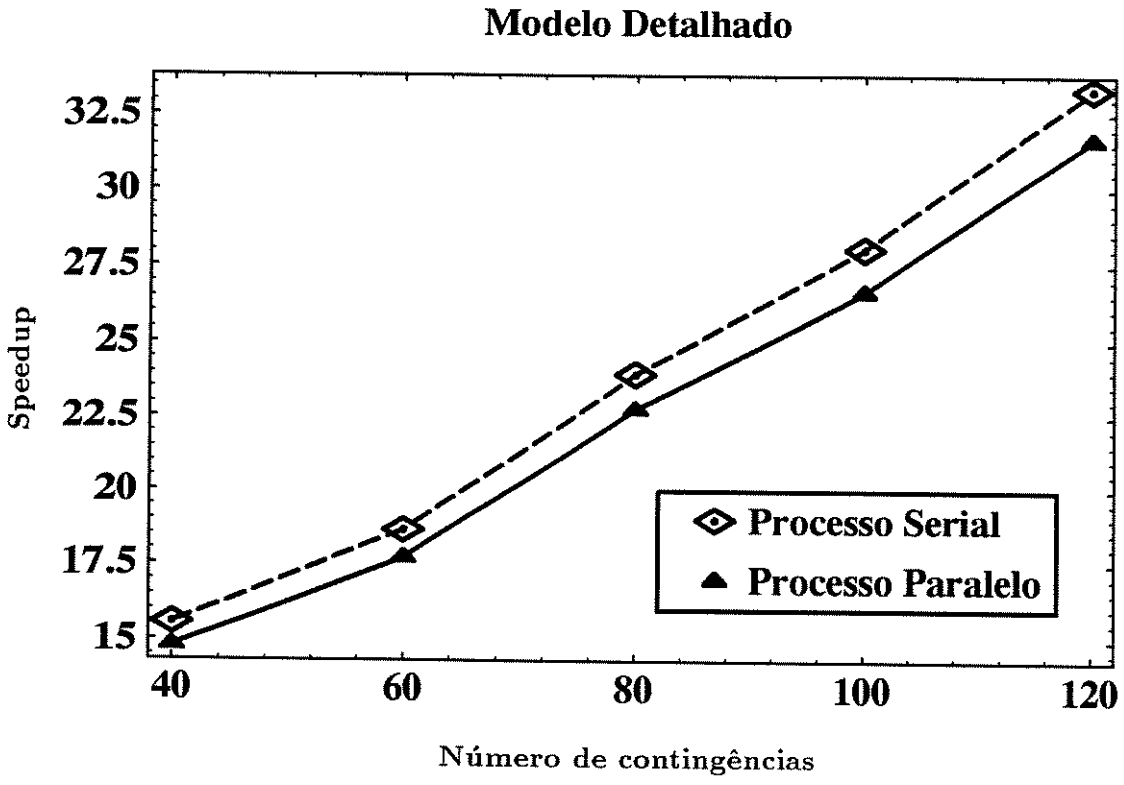

Figura 5.11: *Speedup* no DCE # 1 na Fase 2 com 10 hosts

5.3.2 Resultados para o DCE # 5: Sparc Classic com FDDI

Neste **DCE** o número de *hosts* foi mantido fixo em 9 máquinas e notou-se um comportamento diverso em relação ao caso anterior apresentado. Da Tabela 5.12 pode-se ver uma não-linearidade durante a simulação de 36 casos. Isto pode ser explicado pelo fato desse ambiente estar em fase de implantação, utilizando um sistema operacional novo SunOs 5.3 (Solaris) e uma rede **FDDI** que ainda apresentam problemas de configuração dos equipamentos. Para o restante da tabela a medida que aumenta o número de casos simulados, houve uma diminuição do tempo por processo e um aumento nas duas colunas de *speedup*. A Figura 5.12 ilustra este comportamento graficamente.

Deve-se analisar a importância de usar mais de um índice para medir o desempenho de um programa distribuído, pois como pode ser visto pela primeira linha da Tabela 5.12, quando se deseja analisar apenas 36 contingências, pelo critério de tempo por processo fornecido pela terceira coluna tem-se um valor bem pequeno. Por outro lado, quando se analisa as duas últimas colunas, vê-se que os dados de *speedup* não são tão animadores. Logo, usando mais de um índice, teremos melhores informações e menos chances de incorrer em erros de avaliação sobre qual a melhor configuração para determinado tipo de aplicação. A Figura 5.13 ilustra o comportamento do *speedup* graficamente.

Número de Casos	Tempo Total	Tempo por processo	Speedup proc. paralelo	Speedup proc. serial
36	67.76	1.89	5.49	5.80
54	232.12	4.30	18.80	19.86
72	275.81	3.83	22.33	23.60
90	282.92	3.14	22.91	24.20

Tabela 5.12: Tempos para o DCE # 5 na Fase 2 com 9 hosts

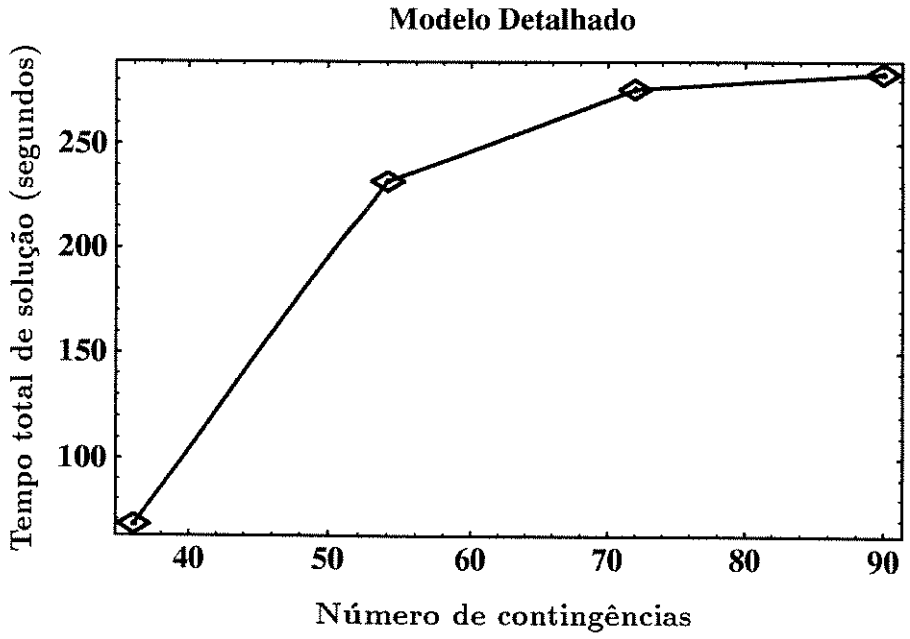

Figura 5.12: Performance no DCE # 5 na Fase 2 com 9 hosts

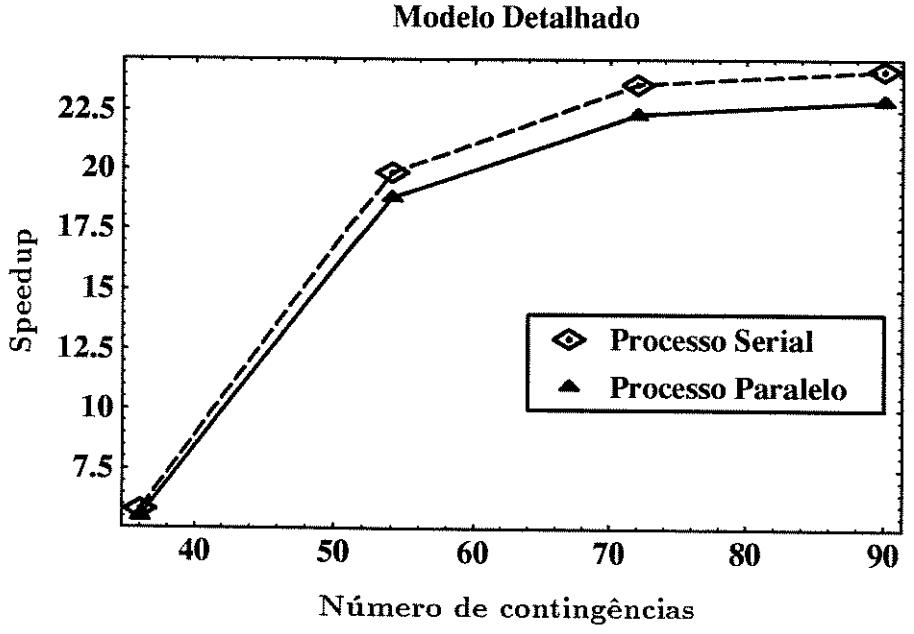

Figura 5.13: *Speedup* no DCE # 5 na Fase 2 com 9 hosts

5.4 Sumário de Resultados

A seguir, apresenta-se um sumário contendo os tempos de simulação (em minutos) para o modelo detalhado (ss320exc) nos diversos **DCE** utilizados. Foram mostrados apenas o maior número de casos (contingências) simulados para cada ambiente de forma a dar uma idéia do horizonte de simulação possível.

	DCE #	Maior $n^{\underline{o}}$ de casos	Tempo total de solução
Fase 1	1	120	64.0 minutos
	2	96	13.8 minutos
	3	96	9.3 minutos
	4	192	5.0 minutos
Fase 2	1	120	7.4 minutos
	5	90	4.7 minutos

Tabela 5.13: Sumário de resultados em diferentes ambientes

Capítulo 6

Conclusões

Os resultados obtidos nesta dissertação, quando da implementação em vários ambientes distribuídos, usando PVM, de uma metodologia para avaliar de modo *on-line* a segurança transitória, permitem tirar as seguintes conclusões:

A utilização de ambientes de computação distribuída é viável para realizar a análise *on-line* da estabilidade transitória de sistemas de potência em paralelo, considerando a tecnologia atual disponível de estações de trabalho e rede de computadores. Em outras palavras, os ambientes distribuídos, utilizados como máquinas paralelas virtuais, podem competir em igualdade de condições com computadores paralelos e vetoriais, envolvendo menores custos.

Para um sistema real de porte médio (320 barras, 46 geradores) mostrou-se que 100 a 200 contingências podem ser analisadas em cerca de 5 minutos, usando modelos detalhados para os geradores. Um horizonte de cinco minutos como ciclo de trabalho em análises *on-line* dinâmicas é bastante razoável. Evidentemente, para sistemas maiores, ou com modelagem mais complexa, os tempos totais de análise irão aumentar, o que exigirá um compromisso entre tempo de ciclo, número e capacidade de *hosts* no ambiente, número de contingências a analisar, tempo de simulação, e outros. Deve ser ressaltado que este trabalho abordou apenas parte do problema da análise *on-line* de segurança dinâmica. A solução completa desse problema, incluindo previsão de carga e redespacho de geração, demanda obviamente muito mais esforço computacional, bem como o desenvolvimento de métodos mais eficientes e adequados para processamento em paralelo.

O conhecimento de técnicas e conceitos de programação paralela, bem como o

domínio sobre o UNIX e seu sistema de gerenciamento de recursos nas máquinas, leva a uma otimização dos programas, resultando numa diminuição no tempo total de execução sem alterar a precisão dos resultados.

O software PVM mostrou ser um bom gerenciador de ambientes distribuídos razoavelmente homogêneos e baseados em UNIX. Demonstrou ser de uso amigável e eficiente no desempenho. A portabilidade entre diferentes arquiteturas também revelou-se um ponto positivo. Só não funcionou bem em ambientes muito heterogêneos e dispersos geograficamente (como o DCE # 6), para a aplicação desejada. Porém, essas características dificilmente serão encontradas em centros de controle distribuídos de sistemas de energia elétrica.

Trabalhos Futuros

Elaborar uma rotina que verifique o estado da rede, usando parâmetros como a carga dos computadores, o número de usuários da rede, a velocidade de cada processador, a memória disponível entre outros, de forma a selecionar as máquinas mais adequadas para efetuar o processamento e, só então, configurar os processos-filhos para que estes efetuem sua execução. Com isso espera-se balancear a carga, reduzindo possíveis atrasos na execução causados por máquinas que estejam sobrecarregadas.

Implementar uma tolerância a falha no programa desta dissertação, permitindo solucionar problemas como a perda de um computador que integra a máquina virtual, perda da conexão entre um computador e a rede, atraso excessivo em retornar resultados ocasionado pelo compartilhamento da **CPU** do computador por processos de outros usuários. Para tanto, podem ser utilizadas as rotinas de `pvmfnrecv()` para temporizar a recepção esperando durante um intervalo de tempo máximo pelos resultados e tomando uma atitude de redirecionar os cálculos para outro computador que esteja disponível.

Podem ser adotadas duas filosofias para abordar este problema:

- eliminar o computador que estiver provocando o atraso através da rotina `call pvmfdelhost()` e adicionar recursos computacionais que estejam na espera à máquina virtual, através da rotina `call pvmfaddhost()` ;
- detectar o processo que está causando atraso no resultado final e eliminá-lo através

da rotina `call pvmfkill()` e repassar os cálculos para uma máquina que esteja menos sobrecarregada neste instante.

Utilizar sistemas maiores (em número de barras, linhas e geradores) para testar a metodologia distribuida de resolução do problema de estabilidade transitória, bem como submeter diferentes tipos de contingências (mútiplas) ao sistema de forma a comprovar a exatidão das análises se segurança e dos cálculos das margens de estabilidade e ou instabilidade.

Testar outras formas de comunicação entre os processadores pois, dependendo da arquitetura das máquinas, pode ser utilizado um protocolo dedicado que venha a apresentar melhores resultados. Como exemplo, poderíamos citar o **PVMe** que vem a ser uma versão do **PVM** aperfeiçoada pela **IBM** que se utiliza do **HPS** (*High Performance Switch*) fazer a comunicação entre os nós do **SP1**.

Apêndice A

Glossário de termos e abreviações

Segue abaixo, uma relação dos termos e abreviações usados nesta dissertação:

broadcast enviar uma mensagem para todos os possíveis receptores;

bus meio único de comunicação compartilhado por um ou mais componentes. Um exemplo seria uma rede compartilhada por processos em vários computadores distribuídos;

buffer área de armazenamento temporária criada na memória. Muitos métodos de roteamento de mensagens entre processos se utiliza de buffers na fonte, destino ou em processos intermediários;

contingência defeito ocorrido no sistema de energia elétrica, normalmente significando a saída de uma linha de transmissão ou um barramento;

DCE *Ditributed Computing Enviroment* - ambiente de computação distribuída que agrega computadores interligados de maneira a se permitir a troca de dados e mensagens;

DSA *Dynamic Security Assessment* - procedimento de avaliação dinâmica de segurança utilizado em centros de controle de empresas de energia elétrica;

DSEE Departamento de Sistemas de Energia Elétrica - departamento que integra a Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas;

eficiência taxa que mede a utilização do *hardware*, sendo igual a divisão entre o speedup atingido em **P** processos e o número de processos **P**;

Ethernet popular tecnologia **LAN** inventada pela Xerox. **Ethernet** é um *bus* com 10-Mbit/SCSMA/CD (**Carrier Sense Multiple Access with Colision Detection**). Computadores numa rede **Ethernet** enviam pacotes de dados diretamente para

outros computadores. Eles esperam até a rede ficar livre antes de transmitirem e retransmitirem quando várias estações simultaneamente desejam enviar mensagens;

FDDI *Fiber Distributed Data Interface* - padrão, especificado pelo Instituto Nacional Americano de Padrões, para *links* de fibra ótica acima de 100 Mbps.

FLOPS *Floating-Point Operations per Second*, uma medida de performance no que se refere ao acesso de memória, sendo igual a taxa pela qual a máquina consegue executar cálculos de ponto flutuante com precisão simples;

Gateway usualmente designa computadores responsáveis pela transferência de informações entre duas redes distintas de computadores;

granularidade tamanho das operações efetuadas por um processo entre eventos de comunicação. Um grão-fino é aquele que executa poucas operações aritméticas durante o tempo em que processa uma mensagem e a próxima, enquanto que um grão-grosso são efetuadas milhões de operações;

grupo dinâmico conjunto de tarefas referenciadas a um nome simbólico comum, com propósitos de endereçamento. Funções de grupos dinâmicos foram criadas, pelos projetistas do **PVM**, para serem gerais, transparentes ao usuário, porém com algum custo no que se refere a eficiência. Quando a primeira função de grupo é invocada, cria-se um processo novo, diferente do *daemon* **pvmd**, para gerenciar este grupo. Permite que qualquer tarefa possa se unir ou deixar qualquer grupo, a qualquer instante de tempo sem ter de informar as restantes. Tarefas podem enviar mensagens mesmo para grupos dos quais não façam parte;

host um computador que integra uma rede computadores na Internet;

hostfile arquivo que define o nome e o número de máquinas a serem configuradas no ambiente de simulação pelo **PVM** ;

HPS *High Performance Switch*, pacote que possibilita a comunicação com alta performance entre os processadores interconectados num **SP** da **IBM**;

IP *Internet Protocol*, protocolo padrão da *Internet* que possibilita o envio de blocos de dados entre hosts interconectados em rede;

LAN *Local Area Network* - é uma implementação física a qual permite o compartilhamento por vários dispositivos tais como estações de trabalho, computadores de cálculos intensivos e dispositivos de comunicação de dados, os quais são interconectados em uma área local (tipicamente dentro de uma mesma área geográfica);

latência tempo gasto para atender uma solicitação de comunicação ou enviar uma mensagem, sendo independente do tamanho ou do tipo da operação. A latência de um sistema por trocas de mensagem é o mínimo tempo gasto para enviar qualquer mensagem;

LSEE Laboratórios de Sistemas de Energia Elétrica - ambientes de computação distribuída que agrega estações de trabalho **Sun** pertencente ao DSEE;

máquina virtual qualquer agrupamento de computadores, homogêneos (mesmo fabricante, mesmo poder computacional, mesmo modelo) ou não, que estejam interconectados fisicamente, operando sobre o controle do *software* **PVM** que permite a comunicação destas máquinas e a execução de programas.

Mbps unidade de medida da taxa de transmissão de dados, baseada no número de vezes por segundo que um sinal troca de estado numa linha de transmissão. Normalmente uma linha de transmissão utiliza apenas 2 sinais para indicar a transferência de dados, fazendo com que a taxa de transmissão seja igual ao número de bits pro segundo que podem ser transferidos;

MIMD *Multiple-Instruction Multiple-Data*, uma categoria na taxonomia de Flynn na qual VÁRIAS cadeias de instruções são concorrentemente relacionadas com múltiplos conjuntos de dados. É uma arquitetura na qual processos HETEROGÊNEOS podem ser executados em instantes de tempo diferentes;

multicast enviar uma mensagem para vários, porém não todos, entre os possíveis receptores;

nó computador básico que representa um bloco de um multicomputador. Usualmente utilizado para descrever um processo composto de memória e mecanismo para comunicação com outros processos no sistema;

processo um espaço de endereçamento, mecanismos de comunicação e um ou mais laços de controle de programa;

PVM Parallel Virtual Machine - Pacote desenvolvido pelo Laboratório Nacional de Oak Ridge na Universidade do Tenessee;

PVMe Parallel Virtual Machine Enhanced - Pacote aperfeiçoado pela **IBM**a partir do **PVM** ;

RISC reduced instruction set computer - tecnologia de microprocessadores que utiliza um pequeno número de instruções para controlar um computador;

scalable aquele que possui a capacidade de aumentar em tamanho. Mais importante, tendo a capacidade de aumentar a performance proporcionalmente à um aumento em tamanho;

socket canal de comunicação entre processos Unix;

SP *Scalable POWERparallel System* ambiente de processamento paralelo da **IBM**;

Sparc *Scalable Processor ARChitecture*: um exemplo de arquitetura que se utiliza de processadores RISC e não um *chip*;

spawn criação de um novo processo ou tarefa **PVM**, podendo possuir um código de execução diferente do processo que a criou;

SIMD *Single-Instruction Multiple-Data*, uma categoria na taxonomia de Flynn na qual UMA cadeia de instruções são concorrentemente relacionadas com múltiplos conjuntos de dados. É uma arquitetura na qual processos HOMOGÊNEOS executam de maneira síncrona as mesmas instruções, usando sua própria base de dados;

SPMD *Single-Program Multiple-Data*, uma categoria às vezes adicionada a taxonomia de Flynn para descrever programas compostos de várias cópias de um mesmo tipo de processo, cada um executando o mesmo código independentemente. **SPMD** pode visto como uma extensão de **SIMD** ou uma restrição de **MIMD**;

task menor componente de um programa endereçável no **PVM**. Geralmente é um processo executado em uma máquina;

TID *Task Identifier* - número identificador para uma tarefa inicializada pelo **PVM** ;

TCP *Transmission Control Protocol* - um confiável protocolo host-host para redes de computadores interconectadas por **IP**;

TEF *Transient Energy Function* - função transitória de energia que fornece o comportamento no tempo da energia total de um sistema elétrico, utilizada em métodos híbridos para cálculo de estabilidade transitória;

XDR *eXternal Data Representation Standard* - uma codificação de dados padrão utilizada na Internet. **PVM** converte dados para o formato XDR de forma a permitir a comunicação entre hosts que utilizam formato de dados diferentes;

XPVM interface gráfica para o pacote do **PVM** ;

Apêndice B

Rotinas de destaque do PVM

Segue abaixo, uma relação das principais rotinas do **PVM** usadas nesta dissertação:

1. pvmfbarrier()
2. pvmfbcast()
3. pvmfconfig()
4. pvmfexit()
5. pvmfinitsend()
6. pvmfjoingroup()
7. pvmflvgroup()
8. pvmfmytid()
9. pvmfpack()
10. pvmfparent()
11. pvmfrecv()
12. pvmfsend()
13. pvmfspawn()
14. pvmfunpack()

B.1 Descrição das principais rotinas do PVM

Segue abaixo uma breve descrição das rotinas da biblioteca do **PVM 3.3** extraída da referência [9] (residentes em `/home/local/pvm3/lib/SUN4/libfpvm3.a`), com a finalidade de introduzir o usuário interessado na programação de aplicações em ambientes de processamento distribuído. Esta seção é útil para uma rápida leitura visando dar as primeiras noções sobre as interfaces com o usuário propiciadas pelo **PVM** . Entretanto, frisa-se, mais uma vez, que são indispensáveis consultas à referência [21] sobre detalhes específicos das rotinas para FORTRAN e C. Assim, são incluídas descrições de rotinas usadas com mais freqüência, compreendendo as seguintes funções:

- controle de processos;
- informação;
- configuração dinâmica da máquina virtual;
- sinalização;
- ajustando e mostrando opções;
- transferência de mensagens;
- uso de grupos de dinâmicos.

B.1.1 Controle de processos

`call pvmfmytid( mytid )`

Em sua primeira chamada esta rotina envolve o processo corrente dentro do **PVM** e gera um único `tid` (= `mytid`) se o processo não foi configurado através da chamada de `pvmfspawn()`. É geralmente a primeira rotina **PVM** chamada nas aplicações.

`call pvmfexit( info )`

Esta rotina passa a informação ao `pvmd` local que o processo está deixando o **PVM** . Ela não finaliza a execução de um processo, deixando que esse continue a executar suas tarefas como qualquer outro processo serial. Por segurança, esta rotina deve ser chamada ao final de cada processo do **PVM** .

`call pvmfspawn( task,flag,where,ntask,tids,numt )`

Suponha a existência prévia de um arquivo executável cujo nome é dado pelo *string* `task`. A chamada da rotina `pvmfspawn()` inicializa (isto é, configura) na máquina virtual `ntask` cópias deste executável (ou seja, cópias desta tarefa). Os argumentos `flag` e `where` são opções oferecidas por `pvmfspawn()` para controle da configuração de processos. `numt` é um valor retornado por `pvmfspawn()` que corresponde ao número de tarefas configuradas com sucesso ou um código de erro se nenhuma tarefa pôde ser inicializada. Se tarefas foram inicalizadas com sucesso, então `pvmfspawn()` retorna um vetor com os `tids` das tarefas efetivamente configuradas.

`call pvmfkill( tid,info )`

Esta rotina "mata" alguma outra tarefa **PVM** identificada pelo `tid`. `pvmfkill()` não foi projetada para "matar" a própria tarefa que a está chamando, a qual deve ser completada [1].

B.1.2 Informação

`call pvmfparent( mtid )`

A rotina `pvmfparent()` retorna o `tid` (= `mtid` = *master's tid*) do processo que desovou a tarefa corrente. Se o processo no qual `pvmfparent()` está sendo chamada não foi criado com `pvmfspawn()`, então o `tid` é ajustado para **PvmNoParent**.

`call pvmfpstat( tid,status )`

Esta rotina fornece o `status` de uma tarefa **PVM** identificada pelo `tid`. A correspondência entre o valor inteiro retornado (`status`) e o seu significado está mostrada na Tabela B.1.

`call pvmfconfig( nhost,narch,dtid,name,arch,speed,info )`

A rotina `pvmfconfig( )` retorna informações sobre a presente máquina virtual. Cada chamada de `pvmfconfig()`, em FORTRAN, retorna informações por *host*. Assim, se esta rotina for chamada `nhost` vezes, a máquina virtual inteira será representada.

[1] Em C, esta operação é completada chamando-se `pvm_exit()` seguida de `exit()`

CÓDIGO DE ERRO	NÚMERO	SIGNIFICADO
PvmOk	0	*okay*
PvmBadParam	-2	*bad parameter*
PvmMismatch	-3	*barrier count mismatch*
PvmNoData	-5	*read past end of buffer*
PvmNoHost	-6	*no such host*
PvmNoFile	-7	*no such executable*
PvmNoMem	-10	*can't get memory*
PvmBadMsg	-12	*can't decode received msg*
PvmSysErr	-14	*pvmd not responding*
PvmNoBuf	-15	*no current buffer*
PvmNoSuchBuf	-16	*bad message id*
PvmNullGroup	-17	*null group name is illegal*
PvmDupGroup	-18	*already in group*
PvmNoGroup	-19	*no group with that name*
PvmNotInGroup	-20	*not in group*
PvmNoInst	-21	*no such instance in group*
PvmHostFail	-22	*host failed*
PvmNoParent	-23	*no parent task*
PvmNotImpl	-24	*function not implemented*
PvmDSysErr	-25	*pvmd system error*
PvmBadVersion	-26	*pvmd-pvmd protocol mismatch*
PvmOutOfRes	-27	*out of resources*
PvmDupHost	-28	*host already configured*
PvmCantStart	-29	*failed to exec new slave pvmd*
PvmAlready	-30	*slave pvmd already running*
PvmNoTask	-31	*task does not exist*
PvmNoEntry	-32	*no such (group,instance)*
PvmDupEntry	-33	*(group,instance) already exists*

Tabela B.1: Códigos de erros retornados pelas rotinas do *PVM 3*

```
call pvmftasks( which,ntask,tid,ptid,dtid,flag,aout,info )
```

Esta rotina retorna informações sobre tarefas **PVM** que estejam sendo executadas na máquina virtual. Cada chamada de `pvmftasks()`, em FORTRAN, retorna informações por tarefa. Assim, se `which` = 0 e se `pvmftasks()` for chamada `ntask` vezes, todas as tarefas serão representadas.

```
call pvmftidtohost( tid,dtid )
```

Se o usuário precisa saber em qual `host` uma tarefa (de `tid` especificado) está sendo executada, então `pvmftidtohost()` pode fornecer esta informação. `dtid` é o *host* **ID** retornado pela chamada de `pvmftidtohost()`.

B.1.3 Configuração dinâmica da máquina virtual

```
call pvmfaddhost( host,info )
call pvmfdelhost( host,info )
```

Estas rotinas, em FORTRAN, respectivamente "adiciona" ou "subtrai" um membro da máquina virtual. `host` é uma *string* que contém o nome da máquina.

B.1.4 Sinalização

```
call pvmfsendsig( tid,signum,info )
```

Esta rotina envia um sinal (*signal number*) a outro processo **PVM** identificado pelo `tid`.

```
call pvmfnotify( what,msgtag, cnt,tids,info )
```

A rotina `pvmfnotify()` solicita ao **PVM** para notificar ao processo que a está chamando sobre a detecção de certos eventos, tal como a falha de um *host*.

B.1.5 Ajustando ou mostrando opções

```
call pvmfsetopt( what,val,oldval )
call pvmfgetopt( what,val )
```

São rotinas de propósito geral para permitir ao usuário "ajustar" ou "mostrar" opções no sistema **PVM** . No *PVM 3*, `pvmfsetopt()` pode ser usada para ajustar várias opções, incluindo: impressão automática de mensagem de erro, nível de depuração, e método de roteamento de comunicação para todas as chamadas **PVM** subseqüentes.

B.1.6 Transferência de mensagem

O envio de uma mensagem em **PVM** consiste de três etapas:

1. inicializar o *buffer*;
2. a mensagem deve ser "empacotada" dentro do *buffer*;
3. a mensagem como um todo é enviada para outro processo.

A mensagem ao ser recebida pelo processo-destino deve ser "desempacotada", item por item, a partir do *buffer* receptor. Dentre as várias opções de transferência de mensagem, destacam-se as mais usadas:

`call pvmfinitsend( encoding,bufid )`

Limpa o *buffer* emissor e cria um novo para "empacotar" uma nova mensagem.

`call pvmfpack( what,xp,nitem,stride,info )` ⟶ no processo-fonte
`call pvmfunpack( what,xp,nitem,stride,info )` ⟶ no processo-destino

Cada escalar, vetor, matriz ou *string* para ser transmitido precisa ser "empacotado" no processo-fonte e "desempacotado" no respectivo processo-destino. Estas são, respectivamente, as funções de `pvmfpack()` e `pvmfunpack()`.

`call pvmfsend( tid_dest,msgtag,info )`

Esta rotina rotula a mensagem com um número inteiro `msgtag` e a envia imediatamente para o processo cujo `tid` especificado é `tid_dest`.

`call pvmfbcast( group, msgtag, info )`

Esta rotina rotula a mensagem com um inteiro `msgtag` e divulga (ou difunde) a mensagem para "todas as tarefas" que compõem o grupo `group` (exceto para ela própria). O valor de retorno da variável `info` permite ao usuário verificar se a operação foi executada

com êxito, ou não. Valores negativos indicam erros que podem ser consultados na Tabela B.1.

`call pvmfmcast( ntask,tids,msgtag,info )`

Esta rotina rotula a mensagem com um inteiro `msgtag` e divulga (ou difunde) a mensagem para "todas as tarefas especificadas" no vetor `tids` (exceto para ela própria).

`call pvmfrecv( tid_source,msgtag,bufid )`

Esta rotina, ao ser chamada numa aplicação **PVM**, bloqueia o código e espera até que uma mensagem com rótulo `msgtag` tenha chegado, proveniente do processo cujo `tid` especificado é `tid_source`. Já a rotina `pvmfnrecv( tid_source,msgtag,bufid )` verifica se a referida mensagem chegou, sem bloquear o código, podendo ser chamada múltiplas vezes para a mesma mensagem. Portanto, `pvmfnrecv()` fica "checando" se a mensagem esperada chegou, enquanto a aplicação realiza trabalho útil entre uma chamada e outra.

B.1.7 Uso de grupos dinâmicos

`call pvmfbarrier( group, count, info )`

Esta rotina bloqueia o processo que a chamou até que todos os processos integrantes do grupo `group` também a tenham chamado. Desta forma, é possível criar um sincronismo entre os N processos configurados `count` e controlar a execução destes. O retorno da rotina é dado pela variável `info` e os erros podem ser consultados na Tabela B.1.

`call pvmfjoingroup( group, inum )`

Esta rotina coloca o processo que a chamou em um determinado grupo `group`. O retorno da rotina é dado pela variável `inum` que indica o número pelo qual a tarefa foi identificada. Valores menores que "0" (zero) indicam que houve erros que podem ser consultados na Tabela B.1.

`call pvmflvgroup( mtid )`

Esta rotina retira o processo que a chamou de um determinado grupo `group`. O retorno da rotina é dado pela variável `info` que indica o número pelo qual a tarefa foi identificada. Valores menores negativos indicam que houve erros que podem ser consultados na Tabela B.1.

Apêndice C

Programa exemplo

Vamos apresentar a seguir uma listagem contendo o código fonte de um programa exemplo, similar ao implementado nesta tese, que contém todos as rotinas básicas a serem usadas num programa de computação distribuída que utiliza o **PVM** .

De posse desse exemplo é possível implementar um programa usando o **PVM** para rodar em ambientes de computação distribuída. Para tanto basta acrescentar a definições de variáveis no cabeçalho; incluir os vetores e matrizes, que possuem as informações para os processos, na parte referente ao empacotamento e envio de dados; colocar as rotinas de cálculos que deseja efetuar e as rotinas de saída de dados.

```
C Exemplo de programa utilizando grupos dinamicos do PVM
C usando o paradigma SPMD
C
      program fgexample
      include '/home/local/pvm3/include/fpvm3.h'
      implicit none

C --- Definindo os valores iniciais e as variaveis
      integer    ...
      parameter ...
      character*32 ...

C --- Fazer declaracoes de rotinas externas
      external ...

C ---------------- Inicio do Programa ----------------------

c     Conectar processo ao PVM e aderir a um grupo dinamico
      call pvmfmytid( mytid )
      call pvmfjoingroup( 'matrix', myinst )
      if( info .lt. 0 ) then
        call pvmfperror( 'joingroup: ')
        call pvmfexit( info )
        stop

      endif

c     Definir o numero de tarefas a serem configuradas
      call pvmfconfig(nhost,narch,dtid,host,arch,speed,info)
      nproc = 2*nhost
      if( nproc .gt. 32 ) nproc = 32

c     Verificar se sou o processo-pai e entao:
c       *  imprimir cabecalho na tela
c       *  inicializar tarefas em outras maquinas
      if( myinst .eq. 0 ) then
        print*
        print*, 'Existem ',nhost, ' maquinas configuradas'
        print*, 'Inicializando ',nproc, ' tarefas'
        print*
        open(unit=10,file='fgexample.out',status='unknown')

c       Criar mais copias deste programa em outras maquinas
        if(nproc  .gt.  1)  then
          call pvmfspawn( 'gexample', PVMDEFAULT, '*',
     >                    nproc -1, tids, ninst )
          if( ninst .lt. nproc-1 ) then
            print*, 'Problemas no spawn. Checar tids'
            print*, tids
            call pvmflvgroup( 'matrix', info )
            call pvmfexit( info )
          endif
        endif
      endif

c     Esperar ate que todas as tarefas tenham aderido ao
c     grupo antes de iniciar o processamento
      call pvmfbarrier( 'matrix', nproc, info )

c     Verificar se sou o processo-pai e entao
c       * distribuir os dados a todos os membros do grupo
      if( myinst .eq. 0 ) then
        call pvmfinitsend( PVMDEFAULT, bufid )
        call pvmfpack(INTEGER4, nproc, 1, 1, info)
        call pvmfpack(INTEGER4, dimension, 1, 1, info )
        call pvmfbcast( 'matrix', INITTAG ,info )
c     Caso nao seja o processo-pai entao:
c       * esperar a resposta de todos os membros do grupo
      else
        call pvmfrecv( -1, INITTAG, info )
        call pvmfunpack( INTEGER4, nproc, 1, 1 ,info)
        call pvmfunpack( INTEGER4, dimension, 1, 1, info)
      endif

C     Efetuar o processamento da rotina de calculos

        call calculos

c     Verificar se sou o processo-pai e entao
c           * imprimir as respostas do programa
      if( myinst .eq. root ) then
        write(10,1000) dimension
      endif
1000  format(' (Os resultados sao ', I3, ' result)')

c     Calculos terminados. Esperar ate que todos os
c     membros do grupo tenham terminado
c     sua execucao antes de terminar.
      call pvmfbarrier( 'matrix', nproc, info)
      call pvmflvgroup( 'matrix', info)
      call pvmfexit( info )
      stop
      end
```

Figura C.1: Programa Exemplo

Bibliografia

[1] A. A. Fouad. Dynamic security assessment practices in North America. *IEEE Trans. Power App. Syst.*, 3(3), August 1988.

[2] A. A. Fouad and S.E. Stanton. Transient stability of a multi-machine power system. Part I: Investigation of system trajectories. *IEEE Trans. Power App. Syst.*, PAS-100(7), July 1981.

[3] A. A. Fouad and S.E. Stanton. Transient stability of a multi-machine power system. Part II: Critical transient energy. *IEEE Trans. Power App. Syst.*, PAS-100(7), July 1981.

[4] A. A. Fouad, S. E. Stanton et alli. Contingency analysis using the transient energy margin technique. *IEEE Trans. Power App. Syst.*, PAS-101(4), April 1982.

[5] A. A. Fouad, V. Vittal. Power system response to a large disturbance: energy associated with sistem separation. *IEEE Trans. Power App. Syst.*, PAS-102(11), November 1983.

[6] A. S. Tanenbaum. *Operating Systems: Desgin and Implementation.* Prentice-Hall, Englewood Cliffs, N.J. 07632, 1987.

[7] A. S. Tanenbaum. *Structured Computer Organization, Third Edition.* Prentice-Hall, Englewood Cliffs, N.J., 1990.

[8] Alquindar Pedroso. Integração da segurança dinâmica nos sistemas de gerência de energia. In *Anais do Simpósio de Automação de Sistemas Elétricos*, pages 1–10, Setembro 1994.

[9] Antônio César Baleeiro Alves. Estudos especiais I - pvm3: Um software para processamento paralelo distribuído. Publicaçao interna do DSEE, Julho 1994.

[10] Brian Stott, Ongun Alsac and Alcir J. Monticelli. Security analysis and optimization. In *Proceedings of the IEEE*, volume 75, pages 1623–1644, Dec 1987.

[11] C. K. Tang et alli. Transient stability index from conventional time domain simulation. *IEEE Trans. on Power Systems*, 9(3), August 1994.

[12] F. A. Rahimi et alli. Evaluation of the tef method for on-line dynamic security analysis. *IEEE Trans. on Power Systems*, 8(2), May 1993.

[13] F. A. Rahimi, N. J. Balu, et alli. Assessing online transient stability in energy management systems. *IEEE Computer Applications in Power*, July 1991.

[14] G. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. S. Sunderam. *PVM: Parallel Virtual Machine A Users' Guide and Tutorial for Network Parallel Computing.* The MIT Press, Cambridge,Massachusetts, 1994.

[15] G.A. Maria, C. K. Tang and J. Kim. Hybrid transient stability analysis. *IEEE Trans. on Power Systems*, 5(2), May 1990.

[16] H. Ota, N Fukushima, Y. Kokai et alli. Development of transient stability control system (tsc system) based on on-line stability calculation. *IEEE PES Summer Meeting*, 585-0 PWRS, July 1995.

[17] K. Demaree, T. Athay et alli. An on-line dynamic security analysis system implementation. *IEEE PES Winter Meeting*, 220-4 PWRS, 1994.

[18] L. G. S. Fonseca and I. C. Decker. Iterative algorithm for critical energy determination in transient stability of power systems. In *Proceedings of IFAC Symposium on Planning and Operation of Electric Energy Systems*, pages 483–489, July 1985.

[19] Marcelo S. Castro and A. Morelato. On-line transient stability assessment on a virtual parallel machine. In *Proceedings of the Sixth Annual Conference of the Power & Energy Society of the Institute of Electrical Engineers of Japan*, pages 77–82, Nagoya - Japan, August 1995.

[20] N. Balu et alli. On-line power system security analysis. In *Proceedings of the IEEE*, volume 80, pages 262–280, Feb 1992.

[21] Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831. *PVM 3 User's Guide and Reference Manual - Release 3.x.*, May 1994.

[22] S. Geeves et alli. Assessment of fast transient stability methods: Performance comparison. Technical report, CIGRE Task Force Report, July 1994.

[23] Brian Stott. Power system dynamic response calculations. In *Proceedings of the IEEE*, volume 67, pages 223–231, Feb 1979.

[24] Sun Microsystems, Inc, 2550 Garcia Avenue, Mountain View, CA, U.S.A. *Sun FORTRAN User's Guide & Reference Manual*, March 1990.

[25] Sun Microsystems, Inc, 2550 Garcia Avenue, Mountain View, CA, U.S.A. *SunOS Release 4.1 Programmer's Overview Utilities & Libraries*, March 1990.

[26] Sun Microsystems, Inc, 2550 Garcia Avenue, Mountain View, CA, U.S.A. *SunOS Release 4.1 Reference Manual: volume 2*, March 1990.

[27] Sun Microsystems, Inc, 2550 Garcia Avenue, Mountain View, CA, U.S.A. *FORTRAN 2.0.1 Numerical Computation Guide*, October 1992.

[28] Sun Microsystems, Inc, 2550 Garcia Avenue, Mountain View, CA, U.S.A. *SPARCompiler FORTRAN 2.0.1 - Reference Manual*, October 1992.

[29] Sun Microsystems, Inc, 2550 Garcia Avenue, Mountain View, CA, U.S.A. *SPARCompiler FORTRAN 2.0.1 - User's Guide*, October 1992.

[30] W. R. Stevens. *Unix Network Programming.* Prentice-Hall, Englewood Cliffs, N.J., 1990.

[31] Y. Mansour et alli. B.c. hydro's on-line transient stability assessment - model development, analysis, and post-processing. *IEEE PES Winter Meeting*, 240-2 PWRS, 1994.