TOLERÂNCIA A FALHAS EM CLUSTERS GEOGRAFICAMENTE DISTRIBUÍDOS BASEADO EM TROCA DE MENSAGENS

Marcelo Soares Souza[1], Anderson Fernando Vieira da Boa Morte[2],

Marco Yudi de Carvalho Hirata[3], Josemar Rodrigues de Souza[4]

Resumo: *Este artigo apresenta uma proposta de implementação de tolerância a falhas em clusters geograficamente distribuídos baseado em troca de mensagens. Propõem-se um método tolerante a um tipo de falha – a desconexão de cluster remoto em uma CoHNOW (Collection of Heterogeneous Network of Workstations) utilizando-se apenas de características do padrão MPI 1.1 (Message Passing Interface). Esta proposta não se baseia em outros projetos de pesquisa que implementam técnicas clássicas de recuperação de falhas, tais como checkpoint, log e/ou através de modificações na semântica do padrão. A solução apresentada finalizou a execução mesmo ocorrendo uma falha através da desconexão física da interconexão entre os clusters geograficamente distribuidos sem incorrer em excessivo custo ao cômputo geral.*

Palavras-Chave: Tolerância a falhas; Computação paralela; CoHNoW.

1. INTRODUÇÃO

O advento da computação de alto desempenho (HPC – *High performance Computing*) através de *clusters* de computadores baseado em estações de trabalho e redes convencionais ou *COTS (Commodity Off The Self)*, inspirado no Projeto *Beowulf*, possibilitou que instituições e organizações com poucos recursos financeiros viabilizassem a construção e o uso de supercomputadores.

Nas chamadas arquiteturas paralelas o principal objetivo é o aumento da capacidade de processamento, utilizando o potencial oferecido por um grande número de recursos computacionais (Sterling, 1999).

Assim projetos que demandam de grande poder computacional, mas que não dispõem de recursos para adquirir supercomputadores tradicionais, para o processamento de crescentes massas de dados em um menor espaço de tempo, podem ser viabilizados.

Tais projetos frequentemente se dão através de parcerias entre diferentes instituições e não raro estas encontram-se geograficamente separadas. Redes privadas ou públicas, como a Internet, servem como meio de interconexão. Abre-se assim a possibilidade de se obter o aumento do poder computacional existente, através da divisão da carga de trabalho entre os recursos computacionais envolvidos, permitindo então que essas parcerias sejam consolidadas.

Neste cenário, de colaboração entre *clusters* geograficamente distribuídos, através de uma rede pública, onde não há garantias de qualidade ou permanencia do serviço de interconexão, faz-se necessário viabilizar meios de se prover tolerância a falhas.

Tradicionalmente, os c*lusters* baseados em troca de mensagens foram construídos tendo

[1]Bacharelando em Informática. Universidade Católica do Salvador. marcelo@cebacad.net.

[2]B.Sc. em Análise de Sistemas. Universidade do Estado da Bahia. anderson@cebacad.net.

[3]B.Sc. em Informática. Universidade Católica do Salvador. yudi@cebacad.net.

[4]Prof. D.Sc. em Computação Paralela. Universidade Autônoma de Barcelona. josemar@aomail.uab.es.

como principal preocupação o provimento de um ambiente eficiente e portável, relegando a segundo plano a questão da tolerância a falhas (Yudi, 2005).

Segundo Weber (2003), o conceito de tolerância a falhas foi originalmente demonstrado por Avizienis em 1967 e apesar de utilizar técnicas já utilizadas desde a construção dos primeiros computadores, a tolerância a falhas ainda não é uma preocupação dos projetistas de sistemas computacionais, ficando sua aplicação quase sempre restrita a sistemas críticos. (Weber, 2003)

Propõem-se neste artigo demonstrar um método de tornar uma aplicação paralela, baseada em troca de mensagens, tolerante a um tipo de falha – a desconexão de um *cluster* remoto em uma CoHNOW (*Collection of Heterogeneous Network of Workstations)* formada por dois *clusters* geograficamente distribuídos e tendo a rede pública Internet como rede de interconexão entre estes.

2. COMPUTADORES PARALELOS VIRTUAIS

Os computadores paralelos virtuais tradicionalmente foram concebidos para prover computação de alta performance, utilizando-se de recursos computacionais construidos com componentes *Off The Self* de baixo custo e facilmente encontrados no mercado e com sistemas operacionais livres e flexíveis.

As máquinas paralelas de baixo custo fazem uso de redes de computadores comerciais, locais e/ou remotas para distribuir suas transações. Atráves de um *software middleware* é formado um conjunto heterogêneo ou homogêneo de computadores (sériais, paralelos ou vetoriais) que é visto como uma única máquina (Souza, 2000).

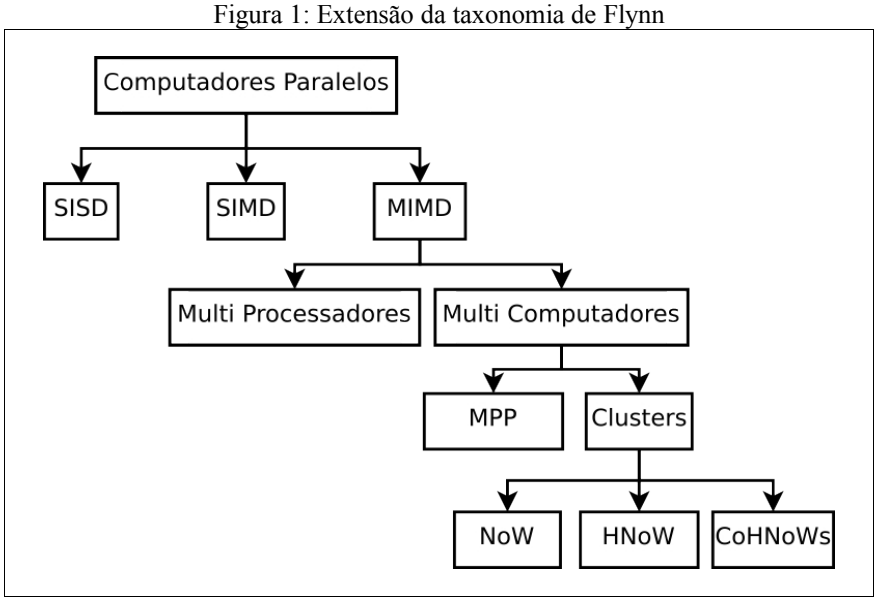
Com a avanço e popularização das tecnologias de rede local (LAN), recurso fundamental na construção de *clusters* eficientes, os computadores paralelos virtuais tornaram-se uma solução atraente e amplamente estudada, e o interesse para sua utilização sobre os tradicionais computadores paralelos, de alto custo e arquitetura fechada, cresce a cada dia.

Conforme aponta por Akl (1989), os computadores que seguem a arquitetura seqüenciais, proposta por Von Neumann esbarram em alguns limites físicos na tentativa de criação de máquinas mais velozes (Akl, 1989). Este fato é de grande importância para explicar o crescente interesse em computadores paralelos virtuais, através de constatações dos evidentes sinais da chegada do limite desta arquitetura.

Os *clusters* são de fato uma rede de computadores, porém possuindo a particularidade de funcionar como uma máquina paralela, daí o fato desses serem chamados de máquinas paralelas virtuais. Cada máquina componente de um cluster é chamada de nodo.

A *HNoW* é um tipo de *cluster,* e *portanto* se enquadra dentro do contexto de computação paralela (figura 1). Esta é uma extensão da taxonomia de Flynn, proposta por Tanenbaum (1990).

Figura 1: Extensão da taxonomia de Flynn



2.1 *Clusters* geograficamente distribuídos

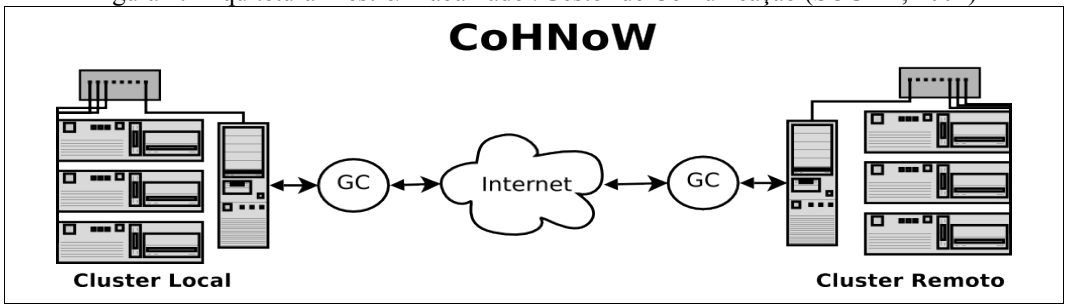
Uma das principais características dos computadores paralelos baseados em estações de trabalho, ou HNoW, é a facilidade de se obter ganhos de performance apenas incluindo, e balanceando, novos recursos computacionais a este, demonstrando a alta escalabilidade desta solução.

Esta boa escalabilidade, aliada a elementos como a evolução da tecnologia de redes de computadores e a popularização da internet foram fatores determinantes para o surgimento de *clusters* formados por um número cada vez maior de nodos, bem como para o surgimento das *CoHNOWS*.

CoHNoWS são *clusters* onde os recursos computacionais são formados por máquinas individuais ou por *clusters* completos e independentes como é demonstrado na figura 2. Neste caso, cada nodo é considerado uma HNoW por si só. (Buyya, 1999) e (Souza, 2004)

Uma das soluções utilizadas para interligar *clusters* geograficamente distribuídos se dá através da utilização de um Gestor de Comunicação (GC). O GC é um nodo do *cluster*, em cada localidade, responsável pela comunicação entre estes. Esta solução visa a otimização da comunicação entre os *clusters* criando uma interconexão ponto-a-ponto.

Figura 2: Arquitetura Mestre/Trabalhador/Gestor de Comunicação (SOUZA, 2004)



2.2 Tolerância a falhas em c*lusters*

A tolerância a falha tem como principal objetivo obter dependabilidade, termo traduzido literalmente do inglês *dependability* que reune diversos conceitos que servem de medida tais como confiabilidade (*reliability*), disponibilidade (*availability*), segurança (*safety*), mantenabilidade (*maintainability*), comprometimento do desempenho (*performability*), e testabilidade (*testability*). Estas são medidas usadas para quantificar a *dependability* de um sistema (Pradhan, 1996).

Nos últimos anos uma acirrada discursão, entre estudiosos da tolerância a falha em computadores paralelos virtuais, se da em torno do provimento de meios para a tolerância a falha pelo padrão da indústria MPI (*Message Passing Interface*), utilizado no desenvolvimento de aplicações paralelas em *clusters* baseado em troca de mensagem.

Batchu (2003) ressalta que o MPI Fórum, no lançamento da versão MPI 1.1 do padrão, apenas enfatizou o provimento de alta performance, escalabilidade e portabilidade entre diversas arquiteturas distintas.

Se por um lado a incorporação de características como confiabilidade implicaria em custos adicionais - o que comprometeria as metas do Fórum, por outro lado essa ênfase em alta performance levou a um modelo com limitadas capacidades de manipulação de erros e confiabilidade. (Batchu, 2003)

Os criadores do MPI contestam tais afirmações explicando que o padrão prover funcionalidades suficientes para o desenvolvimento de estratégias para a tolerância a falhas. Funcionalidades tais como a garantia de comunicação confiável, presença de tratadores de erros, e flexibilidade na definição de um conjunto de erros, tornando assim o padrão MPI apropriado para a tolerância a falhas (Gropp, 2004).

Will Gropp, um dos criadores do padrão MPI e de uma das principais implementaçoes deste (MPICH), afirma que o padrão possui características que permitem a escrita de programas tolerantes a falhas, neste trabalho buscamos demonstrar tal afirmação.

2.2.1 Fases de aplicação das técnicas de tolerância a falhas

A aplicação das técnicas de tolerância a falhas podem ser dividida em quatro fases: detecção, confinamento, recuperação e tratamento. As fases envolvem uma seqüência entre os fatos, que devem ser executadas após a ocorrência de uma falha (Weber, 2003).

A primeira fase, detecção de um erro, ocorre quando uma falha se manifesta como um erro, para então poder ser detectada por algum mecanismo listado na tabela 1. Antes da manifestação como erro, a falha se encontra latente e ainda não pode ser detectada, podendo permanecer no sistema durante a vida útil do sistema, ou seja, sem nunca comprometer o sistema e levar ao estado errôneo.

Devido a essa latência da falha, após a ocorrência da falha até o erro ser detectado, pode ter comprometido os dados. O confinamento estabelece limites para a propagação do dano causado, mas esta ligada diretamente com as decisões estabelecidas na especificação do projeto. Durante a fase de especificação do projeto, devem ser previstas e implementadas restrições ao fluxo de informações para evitar fluxos acidentais e estabelecer uma interação entre a verificação de detecção de erros.

Os mecanismos de detecção de erros baseiam-se em redundância. O conceito de redundância implica na adição de informação, recursos, ou tempo além do necessário para o funcionamento normal do sistema.

Tabela 1 - fases de aplicação das técnicas de tolerância a falhas

Fases	Mecanismos (Souza, 2004)
Detecção de erros	Replicação Testes de limite de tempo Cão de guarda Codificação: paridade, código de detecção de erros, e etc. Teste de razoalidade, de limites e de compatibilidade Testes estruturais e de consistência Diagnóstico
Confinamento e avaliação	Ações atômicas Operações primitivas auto encapsuladas Isolamento de processos Hierarquia de processos Controle de recursos
Recuperação de erros	Técnicas de recuperação por retorno Técnicas de recuperação por avanço
Tratamento da falha	Diagnostico Reparo

A recuperação de erros ocorre após a detecção, e envolve mais uma vez a especificação do projeto, onde deve ser definida a forma de recuperação dos dados perdidos ou não processados. Isso envolve o custo de refazer a tarefa perdida após a detecção do erro, ou tentar implementar uma técnica que torne a tarefa de maior granularidade e apresente pontos de verificação, facilitando assim a restauração da tarefa.

A última fase, tratamento de falhas consiste em localizar a origem do problema (erro), reparar a falha e recuperar o restante do sistema. Após a localização da origem do erro, o componente com problema deve ser isolado para que este não gere mais erros para o restante do sistema, como procuramos utilizar técnicas dinâmicas, o processo de isolamento passa a ser de forma lógica, sem isolamento físico, o elemento controlador (*Master*) passa a desconsiderar qualquer informação gerada pela fonte problemas (Weber, 2003).

2.2.2 Técnicas para a recuperação de falhas

Dois grupos de técnicas tradicionalmente podem ser utilizadas para a recuperação de falhas ocorridas, *Checkpoint* e *Log*. *Checkpoints* são instantes temporais durante a execução de dado processo onde é realizado o armazenamento, em local confiável, de informações necessárias sobre o processamento para que a qualquer instante o sistema possa retornar a esse ponto e ser reiniciado na ocorrência de uma falha. Tal repositório pode ser um disco ou memória protegida por códigos de correção de erros, ou memória duplicada e/ou registradores. (Pradhan, 1996)

O *checkpointing* é o esquema mais utilizado e envolve o armazenamento de toda a informação necessária, sobre o estado do processo, em pontos discretos para garantir que o programa possa ser reiniciado a partir destes pontos caso haja alguma falha. (Pradhan 1996)

A técnica de *log* salva o histórico dos eventos associados aos estados dos processos, para que se possa executá-los a partir do *logs* localizados entre o último *checkpoint* dos processos falhos e o momento da falha. Isto assumido que todo evento não-determinístico da aplicação pode ser registrado.

2.3 Implementação de tolerância a falhas no MPI

Algumas implementações e técnicas foram utilizadas para prover algumas das técnicas, acima descritas, para a tolerância a falha nas implementações baseadas no padrão MPI, descrevemos algumas destas.

CoCheck utiliza a técnica de *checkpoint,* implementada através de um coordenador central

incorrendo em um grande *overhead* devido aos *checkpoints*. Starfish implementa uma API para o desenvolvimento do controle dos *checkpoints* e da recuperação das falhas (Bosilca, 2002) mas sofre dos mesmos problemas de excessivo *overhead* do CoCheck.

Egida baseia-se na técnica de *log* de mensagens com recuperação transparente, mas também incorre em um excessivo overhead (Bosilca, 2002).

O FT-MPI, desenvolvido por Batchu (2003), traz mudanças a semântica do padrão MPI e é amplamente criticado por Gropp (2004). Este gerencia as falhas através da manipulação da estrutura padrão do MPI *communicator*.

Um processo só pode comunicar-se diretamente com um outro, caso ambos pertençam a um mesmo communicator. Quando uma falha ocorre, todos os processos no *communicator* são informados sobre a sua ocorrência (falha da mensagem ou do nodo). Esta informação é transmitida para a aplicação através de um código de retorno.

Tem como principal vantagem a performance, pois não implementa *checkpoints* ou *log* de mensagens (Bosilca, 2002).

MPICH-V é uma solução escalável e que prover *checkpoint* completo e *log* de mensagens (Gropp, 2004). Tem como incoveniente a necessidade de um subsistema dedicado e confiável para o armazenamento dos *checkpoints* e *log* de mensagens.

Entre os problemas encontradas nestas soluções estão a existência de alto *overhead*, não são específicas a arquitetura de *cluster* geograficamente distribuídos*,* ou ainda modificam o padrão MPI.

3. SOLUÇÃO PROPOSTA

A solução aqui proposta e desenvolvida, implementa mecanismos de tolerância à desconexão do *cluster* remoto, adaptando o algoritmo de cálculo de matriz Fox (Pacheco, 1996), utilizando-se do paradigma Mestre/Trabalhador com o uso de um Gestor de Comunicação ponto-a-ponto, apenas com as características definidos no padrão MPI 1.1.

A estratégia adotada fundamenta-se em alguns dos conceitos e técnicas de tolerância a falha apresentados em tópicos anteriormentes, mas não é diretamente derivada de nenhuma das técnicas (*log* e *checkpoints*).

3.1 Pré-processamento

Durante a inicialização do processo o mestre do processamento no *cluster* local envia uma mensagem a cada um dos trabalhadores locais e remotos, através do gestor de comunicação. Todos os pacotes destinados ao GC remoto são mantido numa lista encadeada no mestre local. Esta lista prover o elemento de redundância necessário para a detecção de falhas, conforme tópico 2.2.1.

3.2 Distribuição

Esta fase ocorre desde o inicio do recebimentos, pelo mestre local, dos primeiros pacotes processados pelos trabalhadores até o último. A cada pacote recebido do GC este é retirado da lista encadeada criada na fase de pré-processamento.

3.2.1 Detecção e correção da falha

A detecção de uma falha ocorrida se da através da verificação do estado do *cluster*, constatando-se que nodos do cluster local estejam ociosos durante um determinado tempo, e que todos os pacotes restantes do processamento estão em nodos remotos há a caracterização de ocorrência de uma falha na comunicação entre os *clusters*.

Automaticamente é acionada a correção desta falha implementando a técnica de confinamento, onde todos os pacotes do *cluster* remoto serão ignorados a partir do momento da caracterização da falha. Importante resaltar que a recuperação não ocorre de forma retroativa, todos

os pacotes processados do cluster remoto anteriores a falha são considerados válidos e entrarão no computo final.

Em sua etapa final de correção da falha, há o tratamento ou reconfiguração do sistema através da redistribuição dos pacotes da lista encadeada aos trabalhadores locais.

3.3 Pós-processamento

Esta fase se dá quando os últimos pacotes que estão nos trabalhadores (*workers)* e no GC são recebidos.

3.4 Finalização

Finalizado o processamento de todos os pacotes, o mestre sinaliza através de uma *flag* para que todos os trabalhadores possam finalizar a sua execução. Verifica-se se a quantidade de pacotes inicialmente enviados a todos os trabalhadores, locais e remotos, foram recebidos.

4. METODOLOGIA UTILIZADA NOS EXPERIMENTOS

O experimento conduzido se deu entre dois *clusters* geograficamente distribuídos, um destes encontra-se na Universidade Católica do Salvador (UCSal) e outro na Universidade Autônoma de Barcelona (UAB). O ambiente utilizado consistiu de um ambiente heterogêneo descrito nas tabelas 2 e 3 utilizando-se do Sistema Operacional GNU/Linux, kernel 2.4.20, mpich 1.2.52 como implementação do padrão MPI 1.1 e compilador gcc 3.2.3. A rede de conexão local, em cada *clusters*, foi baseada em padrão Ethernet utilizando *HUB* de 10 mb/s e a interconexão se deu através da Internet por um link dedicado de 512kbps.

Tabela 2: Especificação do *cluster* local UCSal

hostname	Função	Processador	Memória
infoquir1	trabalhador	Intel Pentium 166 Mhz	32 MB
infoquir2	Gestor de Comunicação	Intel Pentium 166 Mhz	32 MB
infoquir3	trabalhador	Intel Pentium III 500 Mhz	64 MB
infoquir5	mestre	Intel Pentium 133 Mhz	32 MB
infoquir6	trabalhador	Intel Pentium 133 Mhz	32 MB
infoquir7	trabalhador	Intel Pentium 133 Mhz	32 MB
infoquir8	trabalhador	Intel Pentium 133 Mhz	32 MB

Tabela 3: Especificação do *cluster* remoto UAB

hostname	Função	Processador	Memória
Aoquir2	Gestor de comunicação	Intel Pentium 200 Mhz	32 MB
Aoquir1	trabalhador	Intel Celeron 533 Mhz	64 MB
Aoquir3	mestre	Intel Pentium 200 Mhz	32 MB
Aoquir4	trabalhador	Intel Pentium 200 Mhz	64 MB
Aoquir5	trabalhador	Intel Pentium 200 Mhz	32 MB
Aoquir6	trabalhador	Intel Pentium 200 Mhz	32 MB

Os testes foram realizados com o algoritmo original e o algoritmo modificado com a implementação proposta. Utilizou-se de matrizes de ordem 1000, 2000 e 3000 com a granulação de blocos da ordem de 100.

A inserção da falha se deu através de desconexão física do gestor de comunicação da conexão com a Internet. Esperou-se o término da execução do algoritmo para a validação da solução, sem que este ficasse bloqueado e da verificação final do número de pacotes processados, a solução mostrou-se eficaz na tolerância a falha proposta.

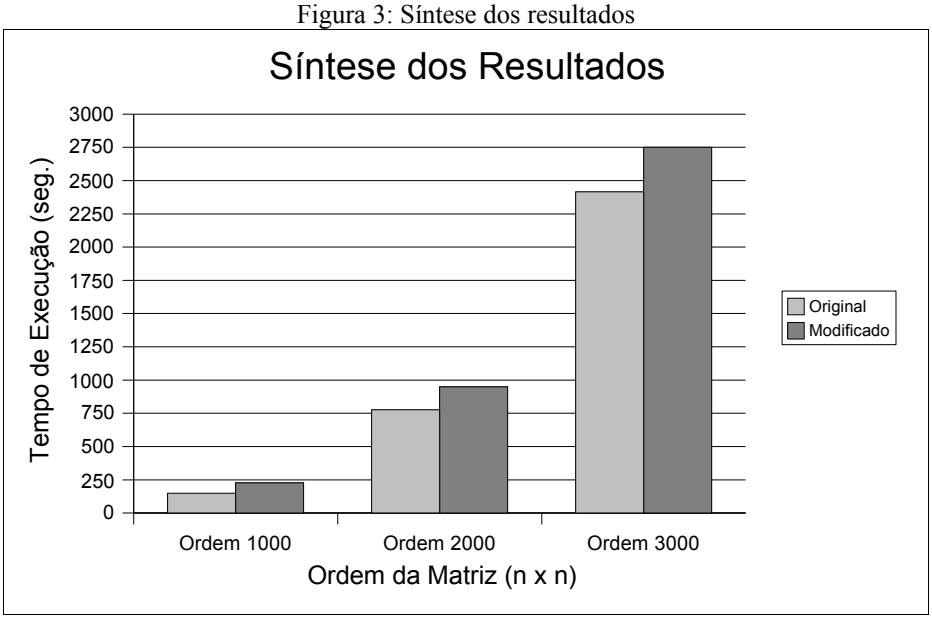
Os experimentos foram realizados durante o mês de novembro de 2004, ressaltamos que caso estes experimentos fossem realizados em outras datas e horários, o tempo de execução do algoritmo poderia ser influenciado pelo tráfego de dados desta interconexão.

Os resultados levantados e apresentados se deram pela análise dos dados obtidos por meio da execução do algoritmo original e modificado, em diferentes horários e dias. Os dados recolhidos servem de amostra, devendo-se levar em consideração o fato da velocidade e latência de conexão serem variáveis e influenciados pelo tráfego de dados na Internet no instante testado.

5. RESULTADOS OBTIDOS

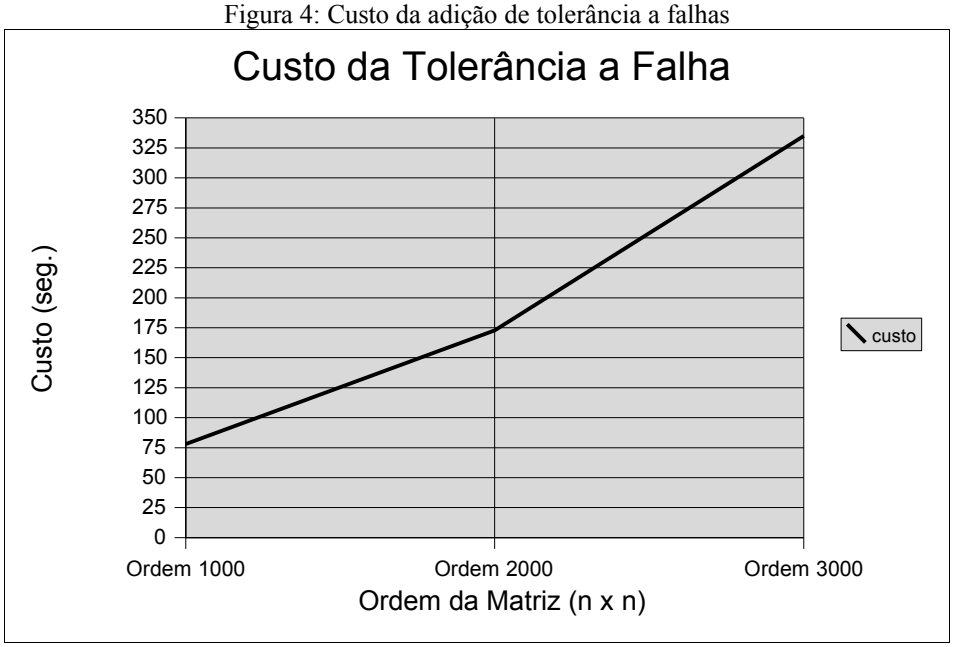
Os resultados obtidos são sintetizados na figura 3, onde o eixo da abscissa representa a ordem da matrizes, enquanto a ordenada representa o tempo de execução em segundos dos algoritmos original e modificado.

Figura 3: Síntese dos resultados



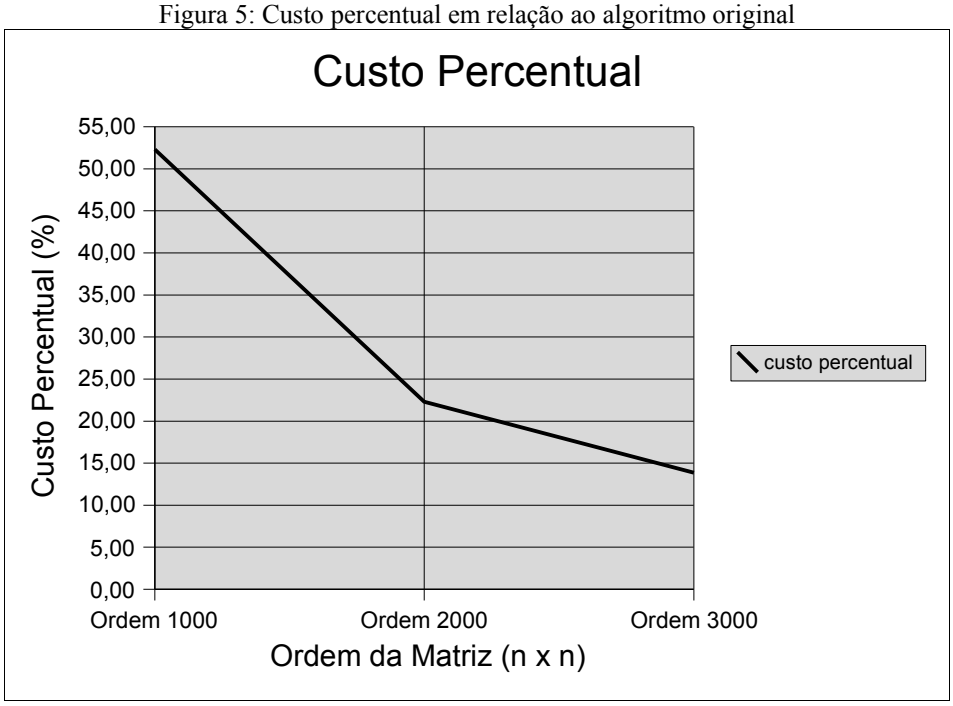
Observa-se que a inserção da característica de tolerância a desconexão representa um custo, que varia conforme a ordem da matriz, os valores deste custo, são expostos na figura 4.

Figura 4: Custo da adição de tolerância a falhas



O custo é diretamente proporcional à ordem da matriz utilizada no experimento, quanto maior a ordem desta, maior é o custo do acréscimo da característica de tolerância a falhas.

Figura 5: Custo percentual em relação ao algoritmo original



Caso o custo seja medido em termos percentuais, em relação ao tempo de execução do algoritmo original, nota-se um comportamento inversamente proporcional em relação à ordem das matrizes. Conforme demonstra a figura 5, quanto maior é a ordem da matriz utilizada, menor será o custo percentual em relação ao algoritmo.

Conclui-se a eficácia do algoritmo em prover a tolerância a desconexão. Mas que a adição de tolerância a falhas representa um custo alto (mais de 50%) em matrizes de ordem baixa comparado a execução do algoritmo original sem tolerância a falhas.

Provou-se um ganho em eficiência em matrizes de ordem acima de 2000 (dois mil), constatou-se uma variação de 22 a 13% até matrizes de ordem 3000 (três mil).

6. CONCLUSÃO

Este trabalho demonstrou eficácia na proposta de inserir a tolerância à desconexão em *clusters* geograficamente dístribuido, através de um gestor de comunicação. Corrobando a afirmação de Gropp (2004), de que é possível implementar características da tolerância a falhas utilizando-se apenas do padrão MPI 1.1.

Outros projetos de pesquisa se propuseram a implementar técnicas de recuperação de falhas no MPI, tais como *checkpoint*, *log* e/ou através de modificações na semântica do padrão. Nossa proposta não se baseou diretamente em nenhuma destas técnicas. O algoritmo proposto terminou sua execução mesmo na ocorrência da falha, que se deu através de desconexão física, da interconexão entre os *clusters* geograficamente distribuidos utilizando-se de uma rede pública (Internet).

Os custos envolvidos na implementação desta solução diminuem a medida que se aumenta a demanda por poder computacional. Os resultados obtidos permitem afirmar que quanto maior a ordem da matriz menor é o custo percentual em se adicionar a característica de tolerância a falhas aqui proposta.

8. REFERÊNCIAS

AKL, Selim G. **The Design and Analysis of Parallel Algorithms**. Englewood Cliffs, New Jersey: Prentice Hall, 1989.

BATCHU, Rajanikanth Reddy. **Incorporating Fault-Tolerant Features into Message-Passing Middleware**. Maio de 2003. 105 f. Dissertação de Mestrado. Department of Computer Science and Engineering, Faculty of Mississippi State University, Mississippi, 2003.

BUYYA, R. **High Performance Cluster Computing: Architectures and Systems**, volume 1. Prentice Hall PTR. 1999.

BOSILCA, George et al. **MPICH-V Toward a Scalable Fault Tolerant MPI for Volatile Nodes**, IEEE, Junho de 2002.

GROPP, William; LUSK, Ewing. **Fault Tolerance in MPI Programs**. International Journal of High Performance Computing Applications, August 2004, vol. 18, no. 3, pp. 363-372

HIRATA, Marco Yudi de C.; Boa Morte, Anderson F. V.; Souza, Marcelo Soares; Souza, Josemar Rodrigues. **Um Framework para o Desenvolvimento de Aplicações Paralelas Tolerante a Falhas baseado em Troca de Mensagem**. I Workshop de Computação Paralela Bahia-Sergipe, Salvador, UNIFacs, 2005 – BaSe-Par 2005.

PRADHAN, Dhiraj K. **Fault Tolerant System Design**. Prentice Hall, New Jersey, 1996.

STERLING, Thomas L. Salmon, John. Becker, Donald J. e Savaresse, Daniel F. ***How to build a Beowulf: a guide to the implementation and aplication of PC clusters***. Massachusetts Institute of Technology. 1999

SOUZA, Josemar Rodrigues de; Furtado, Adhvan; Reboças, André; Rexachs, Dolores; Luque, Emilio; Argollo, Eduardo. I**mproving performance of Long-distance Geographically Distributed Dedicated Clusters**. In: XXXI Seminário Integrado de Software e Hardware (SEMISH), SBC2004 - XXIV Congresso da Sociedade Brasileira de Computação UFBa, Salvador, 31 de Julho a 6 de agosto de 2004b. Brasil. ISBN: 8588442949, pp.22-22, 2004.

SOUZA, Josemar; Souza, Marcelo; Micheli, Milena. **Influência da comunicação no rendimento de uma máquina paralela virtual baseada em Redes ATM** – I Workshop Aplicações Internet2, MetroPOA 2000. PUCRS, Porto Alegre, Rio Grande do Sul, Brasil. Anais 2000.

TANENBAUM, Andrew S. **Organização Estruturada de Computadores**. 3. ed. Rio de Janeiro: LTC – Livros Técnicos e Científicos Editora, 1990. 398p.

WEBER, Taisy Silva. **Um roteiro para exploração dos conceitos básicos de tolerância a falhas**. Apostila do Programa de Pós-Graduação – Instituto de Informática - UFRGS. Porto Alegre, 2003.

_____; Jansch-Pôrto, I.; Weber, R. **Fundamentos de tolerância a falhas**. Vitória: SBC/UFES, 1990. (apostila preparada para o IX JAI – Jornada de Atualização em Informática, no X Congresso da Sociedade Brasileira de Computação).