HyperGrid: Arquitetura de Grade Baseada em Hipercubo Virtual

Elias Procópio Duarte Jr.
Universidade Federal do Paraná
Depto. Informática
P.O. Box 19018 CEP 81531-990
Curitiba PR Brazil
{**elias,bona**}**@inf.ufpr.br**

Luis Carlos Erpen De Bona
Keiko V. O. Fonseca
Centro Federal de Educação Tecnológica do Paraná
Coord. Pós-Graduação E. E. Informática Industrial
Av. 7 de setembro, 3165
Curitiba PR Brazil
keiko@cpgei.cefetpr.br

Resumo. *Os ambientes de grade computacional oferecem acesso a uma grande quantidade de recursos computacionais para a execução de aplicações paralelas e distribuídas. Este trabalho apresenta a arquitetura do HyperGrid, uma plataforma que tem como objetivo permitir o uso do ambientes de grade para executar aplicações paralelas baseadas no paradigma de troca de mensagens escritas usando o MPI (Message Passing Interface). O Hypergrid é baseado em um hipercubo virtual, que é uma rede virtual sobre a Internet. O hipercubo oferece os recursos necessários escondendo a heterogeneidade, as falhas e as reconfigurações do sistema. O HyperGrid é baseado no algoritmo DiVHA (Distributed Virtual Hypercube Algorithm) que é utilizado para manter o hipercubo e monitorar os recursos do sistema.*

Abstract. *Computational grids offer access to a large number of computational resources for the execution of parallel and distributed aplications. This work presents the architecture of the HyperGrid, a platform for the execution of distributed and parallel applications based on the message passing paradigm and written with MPI (Message Passing Interface). The HyperGrid is based on a virtual hypercube, that is a virtual network over the Internet. The Virtual Distributed Hypercube Algorithm (DiVHA) is used to maintain the hypercube and to monitor the system resources. The hypercube provides the necessary resource location transparency and also hides resource heterogenity, providing a fault-tolerant environment that is capable of self reconfiguration when faults occur.*

Palavras Chave: Computação em Grade, Tolerância a Falhas, MPI, Diagnóstico, Hipercubo

1. Introdução

A popularização das redes de computadores, em especial a Internet, permitiu interconectar um grande número de recursos computacionais das mais diversas tecnologias de hardware e software. Oferecer acesso de forma confiável, segura e eficiente a estes recursos é o objetivo dos *grids* ou grades computacionais [1].

Supercomputadores distribuídos podem ser criados usando estes recursos, possibilitando a resolução de problemas numericamente grandes relacionados geralmente com física (astrofísica, dinâmica dos fluídos e mecânica quântica), química (fluxo de reações e físico química), engenharia (testes de impacto e simulações de aeronaves) ou biologia (análise de genoma), entre outros. Um exemplo deste tipo de aplicação é o projeto SETI@home [2] que permite que usuários da Internet emprestem seus computadores para processar sinais de rádio vindos do espaço. Atualmente o SETI@Home está processando a 61.86 Teraflops/s disponibilizados por mais de 5 milhões de computadores pessoais distribuídos em 226 países. O projeto SETI@Home é um caso de sucesso, entretanto ele é desenvolvido para uma aplicação específica.

Boa parte dos problemas científicos é paralalelizável podendo ser resolvidos usando o modelo de troca de mensagem. O modelo de troca de mensagem provê uma abstração de alto nível para a comunicação sobre TCP (*Transmission Comunication Protocol*)/IP (*Internet Protocol*), preservando um alto grau de controle para o programador de como e quando a comunicação deve ocorrer. O MPI (*Message Passing Interface*) [3] é atualmente um dos padrões para programação paralela mais importantes e utilizados. Assim a integração do MPI ao ambiente de grade de forma eficiente viabiliza sua utilização para resolver uma classe bastante grande e importante de problemas. Entretanto, a integração destas aplicações com a computação em grades é um grande desafio.

Para integrar o MPI ao ambiente de grade precisam ser tratadas questões como escalabilidade, hetoregeneidade e dinamicidade do ambiente. Como critério adicional deve-se observar o alto desempenho e o baixo *overhead*, requisitos antagônicos por natureza. Outro desafio é a questão da tolerância a falhas, um ambiente de grade é particularmente mais suscetível a falhas que as plataformas tradicionais de computação. Considerando que uma aplicação distribuída paralela pode requer horas, dias ou até mesmo meses para concluir seu processamento, são necessários mecanismos que evitem que a falha de um ou mais participantes no processamento faça com que toda computação seja perdida.

Este trabalho descreve a arquitetura de uma plataforma para execução de aplicações paralelas e distribuídas em MPI para o ambiente de grade. A plataforma proposta é chamada *HyperGrid* e é baseada em um hipercubo virtual, uma rede virtual sobre a Internet. O hipercubo virtual oferece os recursos de comunicação e processamento necessários, realizando de forma transparente e automática as tarefas de monitoramento dos nodos, substituição de nodos falhos e provendo mecanismos para realizar de forma eficiente as comunicações exigidas pelas aplicações paralelas. O hipercubo virtual é mantido pelo algoritmo DiVHA (*Distributed Virtual Hypercube Algorithm*), que também é uma contribuição deste trabalho.

O restante deste trabalho está organizado da seguinte maneira. A seção 2 descreve a plataforma HyperGrid. A seção 3 apresenta uma arquitetura para a plataforma proposta. A seção 4 introduz o diagnóstico distribuído e descreve o algoritmo DiVHA. A seção 5 finalmente conclui este trabalho.

2. O *HyperGrid*

O *Hypergrid*, proposto neste trabalho, oferece uma nova abordagem para processamento paralelo distribuído em ambiente de grade. A abordagem proposta é baseada em um hipercubo virtual, uma camada de software sobre o ambiente de grade.

O hipercubo virtual é uma rede virtual formada por nodos alocados em computadores na Internet integrados por um ambiente de grade. Estes nodos realizam tarefas de processamento distribuído e trocam mensagens através de enlaces virtuais. Os enlaces virtuais são conexões TCP/IP persistentes entre os nodos. Quando todos os nodos que compõem o sistema estão sem falhas, os enlaces virtuais e os nodos formam um hipercubo. O hipercubo virtual é tolerante a falhas, sendo capaz de realocar automaticamente os nodos das máquinas que se tornam indisponíveis. A figura 2 mostra um hipercubo de 8 nodos, formado por computadores interligados através da Internet. As linhas pontilhadas representam os enlaces virtuais.

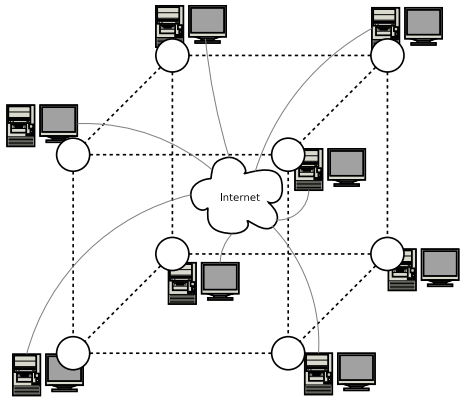

Figura 1: Hipercubo virtual de 8 nodos construído sobre a Internet.

A topologia do hipercubo é muito útil em processamento paralelo. Primeiramente, porque para alguns problemas de processamento de sinais, reconhecimento de padrões, processamento de imagens, entre outros, existem algoritmos eficientes baseados no hipercubo. Segundo, porque outras topologias utilizadas para a execução de algoritmos paralelos como anel, grades n-dimensionais e árvores, estão contidas no hipercubo [4]. Além disto o hipercubo oferece características topológicas importantes como: simetria, diâmetro logarítmico e boas propriedades para algoritmos de tolerância a falhas [5, 6].

Os nodos no hipercubo se comunicam e se monitoram através dos enlaces virtuais. A monitoração ocorre através de mensagens de testes enviadas pelos enlaces virtuais, que têm como objetivo detectar

nodos que tornaram-se indisponíveis e deixaram de realizar suas tarefas de processamento. Neste caso, o hipercubo virtual deve realocar as tarefas deste nodo que se tornou indisponível para uma outra máquina, garantido o funcionamento do sistema como um todo.

O hipercubo é capaz de rotear mensagens através de seus enlaces virtuais. Quando um nodo recebe uma mensagem ele verifica o destino da mensagem, se necessário o nodo encaminha a mensagem para a rota adequada.

No hipercubo os nodos também são virtuais. Inicialmente um nodo em específico pode ser alocado para um determinado computador do sistema. Mas, em caso de falha, este nodo pode ser realocado dinamicamente para outro computador e a tarefa que estava sendo executada é recuperada e reiniciada pelo sistema automaticamente.

A virtualização dos nodos também permite que um ou mais nodos sejam alocados para uma mesma máquina. Isto pode ocorrer em diversas situações. Uma possibilidade é que uma máquina falhou e não existe uma máquina ociosa para a qual o nodo falho possa ser alocado. Ou ainda, que uma máquina possua uma grande capacidade de processamento e possa atender os requisitos de processamento de vários nodos. Este recurso ainda, como trabalho futuro, pode permitir a aglutinação de sítios de supercomputação que possuam um único endereço IP na rede. A seguir é apresentada a arquitetura do HyperGrid.

3. Arquitetura do *HyperGrid*

Visando oferecer uma ferramenta prática para a computação distribuída e paralela em ambiente de grade a arquitetura do *HyperGrid* é descrita nesta seção.

A figura 3 mostra os principais componentes do sistema: os recursos computacionais; a infraestrutura de grade; o hipercubo virtual; uma implementação do padrão MPI; e as aplicações paralelas a serem executadas. Estes componentes são apresentados a seguir.

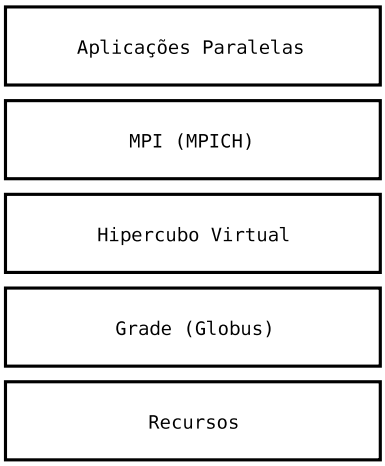

Figura 2: Componentes do *HyperGrid*.

No nível mais baixo da arquitetura estão os recursos do sistema, disponibilizados por um conjunto de computadores interligados por uma rede de comunicação. Um ambiente de grade é utilizado para proporcionar uma interface única e simplificada para estes recursos.

O Globus [7] é a plataforma de grade adotada. Podemos dividir a utilização do Globus em objetivos distintos: inicialização do hipercubo virtual, alocando os nodos virtuais aos computadores disponíveis; disponibilização de informações locais dos sistemas, como capacidade de processamento disponível; transferência dos programas a serem executados; monitoração, integrada ao hipercubo, do processamento executado e das máquinas do sistema; além da identificação, autenticação e autorização dos usuários. Um dos desafios do trabalho, é definir o conjunto mínimo de funcionalidades utilizadas para evitar sobrecargas desnecessárias.

O hipercubo, principal componente do sistema, acessa os recursos através do ambiente de grade, e oferece para a camada superior serviços de comunicação e processamento, de forma confiável, transparente

e homogênea. O hipercubo esconde os mecanismos de acesso aos recursos, bem como esconde as falhas e reconfigurações dos recursos que compõem o sistema.

Os serviços do hipercubo podem ser úteis para uma grande gama de aplicações. Ainda não é possível determinar a granularidade do paralelismo das aplicações que serão adequadas para o hipercubo. Esta determinação depende da avaliação da sobrecarga imposta pela implementação do hipercubo.

No *HyperGrid* optou-se por integrar o hipercubo ao MPI (*Message Passing Interface*) [8], devido à sua popularidade e aceitação como interface padrão para programação paralela. A implementação do MPI escolhida foi o MPICH [9], por ser uma das mais testadas e ser implementado em camadas, facilitando a integração com o hipercubo. Esta integração é feita através de um driver para a camada ADI (*Abstract Device Interface*) [10] do MPICH. Este *driver* ADI é simplificado, já que o hipercubo provê uma abstração para os recursos do sistema.

4. Diagnóstico Distribuído

O algoritmo DiVHA é baseado no algoritmo de diagnóstico distribuído *Hi-ADSD with Timestamps* [11, 12]. Esta sessão apresenta o diagnóstico distribuído e ambos algoritmos de diagnóstico.

4.1. Diagnóstico Hierárquico, Adaptativo e Distribuído

Considere um sistema com N nodos. Assuma que o sistema é totalmente conectado, que os nodos podem estar falhos ou sem-falhas e que os enlaces de comunicação não falham. O objetivo do diagnóstico distribuído em nível de sistema é permitir que todos os nodos sem-falhas determinem o estado, falho ou sem-falhas, de todos os nodos do sistema. Os nodos em um sistema de diagnóstico são capazes de testar outros nodos e determinar seu estado corretamente. O modelo de falhas considerado é crash e o sistema é síncrono.

Os testes realizados pelos nodos do sistema podem ser determinados de acordo com o resultados dos testes realizados anteriormente, estes algoritmos são ditos algoritmos adaptativos. Os algoritmos adaptativos possuem a noção de *rodadas de testes* (*testing rounds*), que é o período de tempo necessário para que todos os nodos executem os testes que lhe foram atribuídos. A *latência* é definida como o número de rodadas de testes necessárias para que todos os nodos do sistema realizem o diagnóstico de um evento ocorrido.

Além de distribuídos e adaptativos os algoritmos de diagnóstico também podem ser hierárquicos. O algoritmo Hi-ADSD (*Hierarchical Adaptive Distributed System-Level Diagnosis*) alcança a hierarquização utilizando o conceito de *clusters* que são conjuntos de nodos testados a cada rodada, o tamanho aumenta progressivamente a cada rodada sempre sendo uma potência de dois. A figura 3 mostra um sistema de oito nodos organizado em *clusters*.

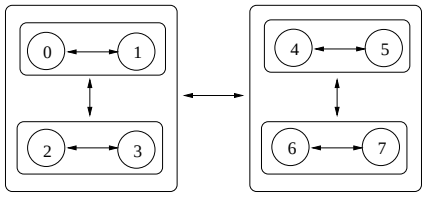

Figura 3: Sistema de 8 nodos agrupados em clusters no algoritmo *Hi-ADSD*.

O algoritmo *Hi-ADSD with Timestamps* é um algoritmo de diagnóstico distribuído, adaptativo e hierárquico. Eventos são definidos como a mudança de estado de um nodo detectada pelo algoritmo de diagnóstico. O algoritmo *Hi-ADSD with Timestamps* é um algoritmo que prevê falhas dinâmicas, onde eventos podem ocorrer antes do diagnóstico do evento anterior ter sido completado por todos os nodos. O *timestamp* é uma informação guardada sobre o estado de cada nodo do sistema, a informação é um contador incrementado a cada mudança de estado. A utilidade do *timestamp* é permitir datar informações, distinguindo as informações mais atualizadas das menos atualizadas.

No algoritmo *Hi-ADSD with Timestamps* sempre que um nodo i testa um nodo j como sem-falhas são lidas informações de diagnóstico de todos os nodos que estão a uma distância de diagnóstico de até $log_2 N - 1$ do nodo i passando pelo nodo j, um conjunto que tem sempre $N/2$ nodos.

4.2. O Algoritmo DiVHA

O hipercubo virtual é mantido pelo algoritmo DiVHA (*Distributed Virtual Hypercube Algorithm*) proposto neste trabalho. Os enlaces virtuais do hipercubo são determinados a partir dos resultados de testes executados pelo algoritmo; enlaces virtuais correspondem a teste entre nodos sem-falhas.

O algoritmo DiVHA é inspirado no algoritmo de diagnóstico *Hi-ADSD with Timestamps* mas propõe uma estratégia nova para determinar os testes realizados pelo sistema.

O algoritmo DiVHA propõe uma nova estratégia para o algoritmo de diagnóstico *Hi-ADSD with Timestamps* considerando as informações de diagnóstico sobre o estado do sistema e determina os testes a serem realizados considerando o sistema como um todo. Esta estratégia tem como objetivo minimizar o número de testes necessários e tornar o algoritmo determinístico. Assim, o algoritmo DiVHA também é um resultado na área de diagnóstico distribuído.

O sistema no algoritmo DiVHA pode ser modelado utilizando grafos. O grafo $H(S)$ é um hipercubo completo, que representa o sistema quando todos os nodos são não falhos, os vértices representam os nodos e uma aresta direcionada entre o nodo i e j indica que o nodo i testa o nodo j. A *distância mínima* entre o nodo i e o nodo p é definida pelo tamanho do menor caminho entre o nodo i e o nodo p em $H(S)$.

O grafo $T(S)$ é o grafo $H(S)$ onde são removidas as arestas cuja origem são nodos falhos. O grafo $T_H(S)$ é construído pelo algoritmo DiVHA a partir de $T(S)$ e determina todos os testes que devem ser realizados pelos nodos do sistema. A cada evento, falha ou recuperação de um nodo, cada nodo do sistema calcula o mesmo $T_H(S)$, sendo assim capaz de determinar seus testes considerando os testes realizados pelos demais nodos.

O grafo $T_H(S)$ é construído adicionando-se arestas ao grafo $T(S)$, com os objetivos de garantir que cada nodo falho seja testado pelo menos por um nodo sem-falha e de garantir a manutenção da distância mínima entre todos os nodos sem-falha do sistema. A manutenção da distância mínima garante a latência de diagnóstico e facilita o roteamento de mensagens no hipercubo virtual.

A construção de $T_H(S)$ pode ser dividida em duas fases. Na primeira fase é determinado o conjunto dos nodos falhos que não recebem nenhuma aresta, ou seja, os nodos falhos que deixaram de ser testados. Para cada nodo deste conjunto é atribuído um testador. O testador de um nodo falho é o nodo sem-falha com a menor distância mínima do nodo falho. Havendo mais de um nodo com a mesma distância mínima para um nodo falho, escolhe-se o testador que pertence ao menor *cluster* em relação ao nodo falho. A definição de *cluster* para este desempate é a do algoritmo *Hi-ADSD* original.

As figuras 4.a e 4.b mostram as atribuições de testadores para nodos falhos em um sistema de 16 nodos. Na figura 4.a podemos tomar o nodo 7 como exemplo. O nodo 1 é escolhido como testador porque está a distância 2 de 7, enquanto 0 está a distância 3. Da figura 4.b podemos retirar outro exemplo, os nodos 8 e 4 estão a distância 2 de 13. O critério de desempate por *cluster* escolhe como testador o nodo 8.

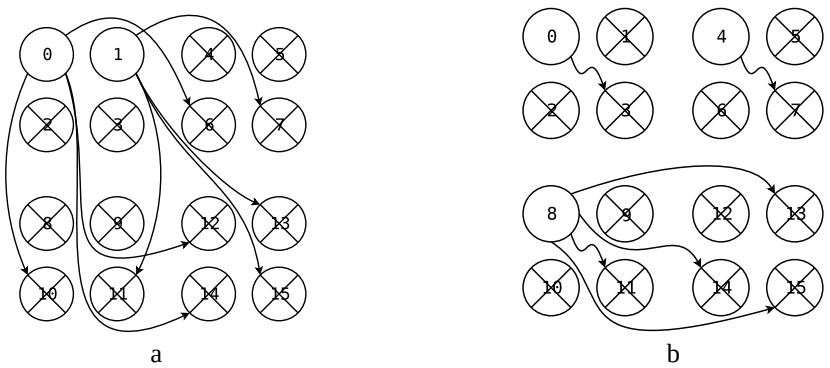

Figura 4: Exemplos de atribuição de testadores para nodos falhos.

A segunda fase de construção de $T_H(S)$, visa garantir a manutenção dos caminhos mínimos entre os nodos sem-falha. Nesta fase são inseridas arestas para reconstruir os caminhos que foram eliminados pela falha dos nodos.

O algoritmo verifica progressivamente o atendimento das distâncias mínimas entre os nodos. Inicialmente são verificados os nodos conectados por distância mínima 2, aumentado esta distância até alcançar $log_2 N$. Quando a verificação da distância mínima entre dois nodos falha, é inserida uma aresta ligando diretamente estes nodos.

5. Conclusão

Este trabalho apresentou a arquitetura do *HyperGrid* uma plataforma que tem como objetivo permitir o uso do ambientes de grade para executar aplicações paralelas distribuídas baseadas no paradigma de troca de mensagens.

O *HyperGrid* é baseado em uma abordagem que faz uso de um hipercubo virtual, que é uma rede virtual criada sobre a Internet. O hipercubo oferece os recursos necessários para a execução de aplicações paralelas com um alto grau de abstração, escondendo a hetoregeneidade dos computadores que disponibilizam os recursos, assim como as falhas e reconfigurações do sistema. O algoritmo DiVHA (*Distributed Virtual Hipercube Algorithm*) responsável por manter o hipercubo virtual também é contribuição deste trabalho.

Também foi apresenta a arquitetura para implementação do *HyperGrid*. A arquitetura é baseada na utilização da implementação da biblioteca de troca de mensagens MPICH, um das mais populares atualmente, e nos serviços do ambiente de grade Globus.

Referências

[1] Ian Foster and Carl Kesselman (eds), *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, July 1998. ISBN 1-55860-475-8.

[2] SETI@Home, http://setiathome.berkeley.edu/

[3] Message Passing Interface Forum, "MPI: A Message-Passing Interface standard," *International Journal of Supercomputer Applications*, Vol. 8, No. 3/4, pp. 165-414, 1994.

[4] G. E. Blelloch, "Programming Parallel Algorithms," *Communications of the ACM*, Vol. 39, No. 3, March 1996.

[5] Sing-Ban Tien, C. S. Raghavendra, "Algorithms and Bounds for Shortest Paths and Diameter in Faulty Hypercubes," *IEEE Transactions on Parallel and Distributed Systems* , pp. 713-718, 1993.

[6] J. M. Krull, J. Wu, and A. M. Molina, "Evaluation of a Fault-Tolerant Distributed Broadcast Algorithm in Hypercube Multicomputers," *in Proceedings of the 20th ACM Annual Computer Science Conference*, pp. 11-18, 1992.

[7] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of Supercomputer Applications*, Vol. 11, No.3, pp. 115-128, 1997.

[8] Message Passing Interface Forum, "MPI-2: Extensions to the Message Passing Interface," 1997.

[9] W. Gropp, E. Lusk, N. Doss, A. Skejellum, "MPICH: A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard," *Parallel Computing*, Vol. 22, N. 6, pp. 789-828, 1996.

[10] W. Cropp, E. Lusk: "MPICH ADI Implementation Reference Manual", *Argonne National Laboratory*, 1995.

[11] E.P. Duarte Jr., L.C.P. Albini, and A. Brawerman, "An Algorithm for Distributed Diagnosis of Dynamic Fault and Repair Events," *In Proceedings of the 7th IEEE International Conference on Parallel and Distributed Systems, IEEE/ICPADS'00*, pp. 299-306, Iwate, Japan, 2000.

[12] E. P. Duarte Jr., L. C. E. Bona, "A Dependable SNMP-based Tool for Distributed Network Management," *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'2002), Dependable Computing and Communications (DCC) Symposium*, Washington D. C., USA, 2002.