HyperBone: Uma Rede Overlay Baseada em Hipercubo Virtual sobre a Internet

Luis C. E. Bona [1] **, Elias P. Duarte Jr.**[2] **, Samuel L. V. Mello** [2] **, Keiko V. O. Fonseca** [1]

[1] Universidade Tecnológica Federal do Paraná
CPGEI Av. Sete de Setembro 3165 Curitiba PR
{bona, keiko}@cpgei.cefetpr.br

[2]Universidade Federal do Paraná
Departamento de Informática Caixa Postal 19018 Curitiba PR
{elias, slucas}@inf.ufpr.br

Abstract. *This paper presents HyperBone, an overlay network based on a virtual hypercube that offers services such as monitoring and routing, allowing the execution of distributed applications across the Internet. Hypercubes are scalable by definition, presenting several properties such as symmetry and logarithmic diameter, that are advantageous for distributed and parallel applications. HyperBone nodes run the Distributed Virtual Hypercube Algorithm (DiVHA) in order to maintain the topology. DiVHA keeps the hypercube properties even when the number of nodes is not a power of two, or under a dynamic fault situation, in which nodes fail and recover continuously, leaving and joining the system. HyperBone was implemented and experimental results are presented, obtained from the execution of a set of MPI parallel applications run on a virtual hypercube spread across the world built with PlanetLab nodes.*

Resumo. *Este trabalho apresenta o HyperBone, uma rede overlay baseada em hipercubo virtual que oferece serviços de monitoração e roteamento, permitindo a execução de aplicações distribuídas na Internet. O hipercubo é uma estrutura escalável por definição, apresentando características topológicas importantes como: simetria, diâmetro logarítmico e boas propriedades para tolerância a falhas. O hipercubo virtual é mantido pelo algoritmo DiVHA (Distributed Virtual Hypercube Algorithm) que garante propriedades do hipercubo mesmo quando o número de nodos não é uma potência de dois e em ambientes sujeitos a situações dinâmicas de falhas, nas quais nodos falham e se recuperam continuamente, deixando o sistema para depois serem reintegrados. O HyperBone foi implementado e são apresentados resultados experimentais obtidos a partir da execução de um conjunto de aplicações MPI paralelas sobre um hipercubo virtual construído com nodos do PlanetLab espalhados pelo mundo.*

1. Introdução

Diversos sistemas distribuídos de larga escala têm surgido na Internet nos últimos anos. O desenvolvimento destes sistemas é consequência da Internet ter permitido interconectar um grande número de recursos computacionais. Além disso, protocolos e padrões vêm derrubando a barreira da heterogeneidade das tecnologias. A computação em grade [Foster and Kesselman 1997, OurGrid 2005], ou *grid computing*, e as redes peer-to-peer

(P2P) [Clark 2001], são exemplos de tecnologias que têm permitido acessar e utilizar esses recursos, formando sistemas distribuídos de grande porte e dinâmicos, nos quais nodos não são permanentes, se integrando e deixando continuamente o sistema, que está sujeito a falhas e recuperações também contínuas.

Uma abordagem promissora para construir esses sistemas distribuídos de larga escala na Internet são as redes overlay [Amir and Danilov 2003]. Em princípio as aplicações distribuídas poderiam gerenciar seus recursos acessando diretamente a rede subjacente, mas uma rede overlay pode oferecer serviços como manutenção da rede, segurança e alta disponibilidade que dificilmente poderiam ser providos no nível de rede. Em redes P2P as redes overlay têm sido utilizadas como forma de tornar o sistema escalável [Doval and O'Mahony 2003], outras redes overlay destinam-se a melhorar os serviços oferecidos pelos protocolos da Internet, como suportar QoS ou multicast [QBone 2005, Eriksson 1994] ou ainda oferecer formas de roteamento alternativas [Andersen et al. 2001].

O HyperBone é uma rede overlay formada por nodos espalhados pela Internet interligados por enlaces virtuais, que são arestas de um hipercubo virtual, implementadas como conexões TCP (*Transmission Control Protocol*) persistentes. Os nodos do HyperBone são hosts da Internet capazes de realizar tarefas de processamento. Se o número de nodos é uma potência de dois e todos nodos estão sem falhas, a topologia do sistema é um hipercubo completo.

A principal característica do sistema é a escalabilidade, o hipercubo é uma estrutura escalável por definição, que apresenta características topológicas importantes como: simetria, diâmetro logarítmico e boas propriedades para tolerância a falhas [Tien and Raghavendra 1993, Krull et al. 1992, Tzeng and Chen 1994]. Deve ser destacado neste trabalho que as características que dificultam a utilização dos hipercubos para arquiteturas de máquinas paralelas não estão presentes no hipercubo virtual: o número de nodos não precisa ser potência de dois e o custo de adicionar ligações, que são virtuais, é baixo.

Mesmo quando o número de nodos não é potência de dois ou alguns nodos não estão disponíveis, o HyperBone ainda é capaz de manter as propriedades logarítmicas do hipercubo. A topologia é mantida pelo HyperBone utilizando o algoritmo DiVHA (*Distributed Virtual Hypercube Algorithm*), também proposto neste trabalho. O DiVHA efetivamente calcula os enlaces do hipercubo virtual, permitindo aos nodos se auto-organizarem mantendo características do hipercubo como o diâmetro logarítmico. Desta forma, o sistema é dinâmico, ou seja, os nodos podem falhar e se recuperar constantemente. O HyperBone implementa um sistema de monitoração distribuída, possibilitando que cada nodo mantenha uma lista dos nodos que estão apresentando comportamento estável e que podem ser utilizados para executar tarefas de processamento.

O HyperBone é capaz de rotear mensagens pelos enlaces do hipercubo virtual, oferecendo um serviço de comunicação escalável que permite a qualquer par de nodo se comunicar diretamente usando rotas com distância logarítmica, sem a necessidade de estabelecer enlaces entre todos os nodos. Todo o tráfego entre os nodos é tunelado pelos enlaces virtuais, facilitando a instalação de nodos atrás de firewalls restritivos. O roteamento no hipercubo virtual ainda permite encontrar rotas alternativas àquelas usadas pelos

nodos quando se comunicam diretamente pela Internet.

O HyperBone foi implementado no PlanetLab [PlanetLab 2005], um ambiente que permite avaliar aplicações distribuídas em escala global sob condições reais. Os resultados experimentais mostram o funcionamento do sistema e a capacidade de executar aplicações distribuídas em escala mundial utilizando a rede overlay oferecida pelo HyperBone.

O restante deste trabalho está organizado da seguinte maneira. A seção 2 apresenta o modelo do sistema e a construção e manutenção de hipercubo virtual com o HyperBone e o DiVHA, incluindo exemplos de execução e provas formais de correção. Resultados experimentais obtidos a partir da implementação no PlanetLab são apresentados na seção 3. A seção 4 apresenta trabalhos relacionados. Finalmente, a seção 5 conclui o trabalho.

2. HyperBone: Uma Rede Overlay Escalável

O HyperBone é uma rede overlay formada por nodos espalhados pela Internet interligados por enlaces virtuais. Um enlace é implementados como uma conexão TCP persistente. Se o número de nodos é uma potência de dois e todos os nodos estão funcionando sem falhas, a topologia do sistema é um hipercubo completo. Quando o número de nodos não é potência de dois, ou alguns nodos não estão disponíveis, o HyperBone ainda é capaz de manter as propriedades logarítmicas do hipercubo. O algoritmo DiVHA (*Distributed Virtual Hypercube Algorithm*), permite aos nodos calcular de forma independente seus enlaces virtuais na rede. A seguir o modelo do sistema é introduzido, bem como o algoritmos que mantêm o hipercubo virtual.

2.1. Modelo do Sistema

Considere um sistema S composto por um conjunto de N nodos, n_0, n_1,...,n_{N-1}. Alternativamente nos referimos ao nodo n_i como *nodo* i. O sistema é considerado totalmente conectado, ou seja, qualquer par de nodos pode se comunicar diretamente. Os relógios não são sincronizados e não são feitas asserções sobre as velocidades relativas dos processadores.

Um nodo n_i pode assumir um de dois estados, *working* ou *unresponsive*. Um *evento* é definido como a mudança de estado de um nodo, tanto de *working* para *unresponsive* como de *unresponsive* para *working*. A coleção dos estados de todos os nodos é o *estado do sistema*. Se considera também um modelo dinâmico para os estados do sistema, ou seja, os nodos podem alternar entre o estado *unresponsive* e o estado *working* continuamente.

Os nodos são capazes de realizar testes em outros nodos, assim um nodo testador determina o estado do nodo testado. Os testes são realizados periodicamente, dentro de um intervalo de testes fixo, que pode variar dependendo da tecnologia do sistema, de alguns poucos nanosegundos a vários minutos. Um nodo no estado *working* é considerado como capaz de testar outros nodos e informar corretamente os resultados destes testes. Um teste consiste em uma tarefa atribuída pelo nodo testador ao nodo testado. A tarefa é executada e o resultado devolvido para o testador que compara o resultado com o esperado. Um intervalo de *timeout* é empregado para limitar a espera pelo resultado, a ocorrência do *timeout* é equivalente a uma resposta incorreta. Como o sistema é assíncrono o timeout não é um indicativo concreto de que o nodo está falho (*crashed*),

assim um teste indica apenas que o nodo ou está funcionando corretamente ou não respondeu corretamente a um teste enviado ou ainda que não respondeu dentro de um intervalo esperado.

Uma *rodada de testes* é definida como o período de tempo em que todos os nodos no estado *working* executam seus testes. A *latência* do sistema é definida como o tempo necessário para que todos nodos no estado *working* descubram a ocorrência de um novo evento.

Consideramos também que os nodos podem passar por períodos de instabilidade que causam falhas de desempenho que fazem com que seu estado alterne continuamente entre *unresponsive* e *working*. Os *parâmetros de disponibilidade* são definidos por um período mínimo de tempo pré-definido em que o nodo deve permanecer no estado *working* para ser considerado *available*, e por um período mínimo de tempo que o nodo deve permanecer no estado *unresponsive* para ser considerado *unavailable*. Esta estratégia facilita a monitoração de sistemas altamente instáveis, com mudanças contínuas de estado dos nodos. Cada nodo mantém informações sobre os estados de disponibilidade de todos os outros nodos.

2.2. Construindo e Mantendo o Hipercubo Virtual

Seja o *grafo de testes*, também chamado $T(S)$, um grafo direcionado cujos nodos são os nodos de S. Existe uma aresta direcionada do nodo i para o nodo j se o nodo i testa o nodo j na rodada de testes mais recente. Assim $T(S)$ representa todos os testes executados na última rodada de testes. Quando todos os nodos estão no estado *working*, $T(S)$ é um hipercubo. Quando alguns nodos são considerados *unresponsive*, $T(S)$ deixa de ser um hipercubo e novas arestas são adicionadas com o objetivo de preservar duas propriedades do hipercubo: (1) o diâmetro logarítmico, ou seja, a distância entre qualquer par de nodos i e j em $T(S)$ nunca é maior que $log_2 N$; e (2) o número total de arestas em $T(S)$ nunca é maior que $N log_2 N$.

O algoritmo mostrado na figura 1 é executado por todos nodos para determinar e obter informações sobre o estado dos outros nodos do sistema com objetivo de manter o hipercubo. A cada rodada de testes todos nodos no estado *working* executam seus testes, o conjunto de testes atribuídos ao nodo i chamado $t(i)$ é o conjunto de nodos adjacentes ao nodo i em $T(S)$. Seja a *distância* do nodo i para o nodo j, chamada $d_{i,j}$ ou $d(i,j)$, o tamanho do caminho com menor número de arestas do nodo i para o nodo j em $T(S)$. Seja $D_{i,k}$ o conjunto de todos os nodos p tal que $d_{i,p} \leq k$.

Quando o nodo i testa um nodo j no estado *working* ele obtém informação sobre todos os nodos k com distância de j menor o igual a $log_2 N - 1$, ou seja $k \in D_{j,log_2 N-1}$, tal que $d_{j,k} < d_{i,k}$. Quando todos os nodos em $T(S)$ estão no estado *working*, em um único teste o testador obtém informações de estado de sobre $N/2$ nodos. Ao executar todos seus testes o nodo i obtém informações sobre o estado de todos os nodos do sistema.

É possível que um nodo i obtenha informação sobre um nodo k através de um ou mais nodos. Para garantir que o nodo i obtenha apenas a informação mais nova de estado do nodo k utilizamos o mecanismo de *timestamps*. Esta estratégia permite datar a informação de estado, ou seja, é possível determinar se uma informação recebida é mais nova que outra também recebida ou já mantida. O timestamp é implementado como um contador das mudanças de estados, em outras palavras o timestamp é incrementado

```
HyperBone Construction & Maintenance
T(S)=DiVHA(S)
DO
  FOR EACH j IN t(i)
     test(j)
     IF the state of tested node has changed THEN
        update status information about j
     IF node j is working THEN
        get from j information about nodes k in Dk,logN−1 such that dj,k < di,k
        update local information comparing timestamps
     IF new events have been discovered THEN
        update T(S) with DiVHA
  END FOR
  SLEEP until the next Testing Interval
FOREVER
```

Figura 1. O algoritmo para criação e manutenção do hipercubo virtual.

sempre que um teste é executado e o testador descobre que o estado no nodo testado foi alterado. Quando um nodo detecta um evento, ele executa o algoritmo DiVHA, apresentado abaixo, que determina o novo conjunto de testes que devem ser executados pelo nodo.

2.3. O Algoritmo DiVHA

Os nodos do HyperBone executam o algoritmo DiVHA (*Distributed Virtual Hypercube Algorithm*) com o objetivo de determinar o conjunto de testes a ser executado dado o estado percebido do sistema. A especificação do DiVHA é dada na figura 2.

```
DiVHA(system S)
T(S) initially does not have any edge
FOR s=1 to logN do
  FOR distance=1 to s do
    FOR each fault_free node i=0...N–1 do
      FOR each node j that belongs  c(i,s) such that  h(i,j) = distance
        IF (d(i,j) > s or non_existent) THEN
          add edge(i,j) to T(s)
          next i
UNTIL  d(i,j) <= s, such that node i is working and node j belongs to c(i,s)
```

Figura 2. Especificação do algoritmo DiVHA.

Inicialmente o grafo $T(S)$ tem apenas vértices, correspondentes aos nodos do sistema, e nenhuma aresta. Os nodos são organizados de forma hierárquica para calcular as arestas. *Clusters* são utilizados para agrupar nodos, que são conjuntos de nodos cujo tamanho cresce progressivamente em potências de 2. Um cluster de p nodos $n_j, ..., n_{j+p-1}$, onde p é uma potência de 2 e é maior que 1, é formado pela união de dois clusters, um contendo os nodos $n_j, ..., n_{j+p/2-1}$ e outro $n_{j+1}, ..., n_{j+p-1}$. A figura 3 mostra um sistema de 8 nodos organizado em clusters. Um cluster $C_{i,s}$ pode ser definido recursivamente pela expressão: $C_{i,s} = C_{j,s-1} \cup C_{i,s-1}, j = i \oplus 2^{s-1}$ e $C_{i,1} = j, j = i \oplus 1$

O algoritmo considera os clusters do menor para o maior, ou seja, o tamanho do cluster é incrementado com s variando de 1 até $log_2 N$. A cada iteração o algoritmo computa arestas de nodos *working* i, variando i indo de 0 até $N-1$, para nodos pertencentes a Ci, s. Desta forma são adicionadas arestas entre o nodo 0 e nodos pertencentes ao cluster $C_{0,s}$, entre o nodo 1 e os nodos pertencentes ao cluster $C_{1,s}$, e assim por diante até o nodo $N-1$ e os nodos pertencentes ao cluster $C_{N-1,s}$. Uma aresta é adicionada entre o

nodo i e o nodo $j \in Ci, s$ sempre que não existir um caminho do nodo i para o nodo j em $T(S)$ com distância $\leq s$. Uma iteração termina quando existe pelo menos um caminho com tamanho $\leq s$ entre todo nodo i no estado *working* e os nodos pertencentes a Ci, s.

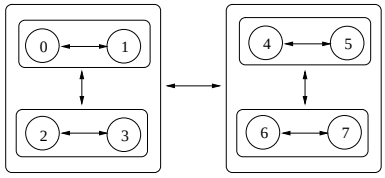


Figura 3. Sistema de 8 nodos agrupados em clusters pelo algoritmo DiVHA.

A adição de arestas é feita de forma distribuída entre os nodos do sistema baseando-se nas distâncias do hipercubo. A *distância no hipercubo* do nodo i para o nodo j, chamada $h_{i,s}$, é o número de arestas no caminho mais curto entre o nodo i e o nodo j em um hipercubo perfeito. Esta estratégia para adição de arestas permite formar um hipercubo quando todos os nodos estão no estado *working* e distribuir entre os nodos de forma balanceada as arestas adicionais que podem ser necessárias quando existem nodos considerados *unresponsive*.

2.4. Provas Formais

Agora apresentamos a provas formais que o DiVHA garante o diâmetro logarítmico do sistema mesmo quando o número de nodos no estado *working* não é potência de 2, e que o número total de arestas em $T(S)$ não é maior que $Nlog_2N$. Também provamos que a latência do HyperBone é log_2N no pior caso.

Teorema 1. *No grafo $T(S)$ determinado pelo algoritmo DiVHA, a distância de qualquer nodo* working *para qualquer outro nodo é $\leq log_2N$.*

Demonstração. Considere o algoritmo da figura 2, o laço mais externo a cada iteração insere arestas que garantem $d(i, j) \leq s$ para todo n_i *working* $\in S$ e $n_j \in C_{i,s}$. Vamos provar por indução que ao término de cada iteração s temos $d(n_a, n_b) \leq s$ para todo $n_a, n_b \in C_{i,s+1}$. Dado que $C_{i,log_2N+1} = S$, ao final da última iteração a distância de qualquer nodo *working* para outro nodo é menor ou igual a log_2N.

Base. Na iteração em que $s = 1$, $d(n_a, n_b) = 1$ para todo n_a *working*, $n_b \in C_{i,2}$.

Na primeira iteração são inseridas arestas garantindo $d(n_i, n_j) \leq 1$ para todo n_i *working* $\in S$ e $n_j \in C_{i,s}$. Por definição $C_{i,2} = C_{i,1} \cup C_{j,1}, i = k \oplus 1$, ou ainda, $C_{i,2} = C_{i,1} \cup \{n_i\}$. Suponha n_a *working*, $n_b \in C_{i,2}$. Então $n_b \in C_{a,1}$ e logo $d(n_a, n_b) = 1$, já que $d(i, j) \leq s$ para todo n_i *working* $\in S$ e $n_j \in C_{i,s}$.

Hipótese de Indução. Na iteração s, $d(n_a, n_b) \leq s$ para todo n_a *working*, $n_b \in C_{i,s+1}$.

Passo. Ao final da iteração $s + 1$ a $d(n_a, n_b) \leq s + 1$ para todo n_a *working*, $n_b \in C_{i,s+2}$.

Na iteração $s + 1$ são inseridas arestas garantindo $d(n_i, n_j) \leq s + 1$ de todo n_i *working* $\in S$ para todo $n_j \in C_{i,s+1}$. $C_{i,s+2}$ é dado por $C_{i,s+1} \cup C_{j,s+1}, j = i \oplus 2^s$. Suponha n_a *working* e $n_b \in C_{i,s+2}$. Se n_a, n_b pertencem ao mesmo cluster $s + 1$ pela hipótese de indução, $d(n_a, n_b) \leq s$; Se n_a e n_b pertencem a clusters $s + 1$ distintos, então $n_a \in C_{b,s+1}$

e $n_b \in C_{a,s+1}$. Logo $d(n_a, n_b) \leq s+1$ porque $n_b \in C_{a,s+1}$ e $d(n_i, n_j) \leq s+1$ para todo n_i *working* e todo $n_j \in C_{i,s+1}$. Portanto $d(n_a, n_b) \leq s+1$ para todo n_a *working*, $n_b \in C_{i,s+2}$. □

Teorema 2. *No grafo $T(S)$ determinado pelo algoritmo DiVHA, o número máximo de arestas é $Nlog_2N$.*

Demonstração. Considere o algoritmo da figura 2, a cada iteração s são adicionadas arestas entre clusters $C_{i,s}$ e $C_{j,s}$ distintos formando $C_{i,s+1}$. Suponha que uma aresta é adicionada do nodo $n_k \in C_{i,s}$ para o nodo $n_l \in C_{j,s}$, como mostra a figura 4. Da demonstracão indutiva do teorema 1, $d(n_i, n_k) \leq s-1$ para todo n_i *working* $\in C_{i,s}$, com a aresta inserida aresta $i_j \rightarrow j_k$, temos $d(n_i, n_l) \leq s$ para todo n_i *working* $\in C_{i,s}$. Assim o DiVHA não insere nenhuma outra aresta para n_l nesta iteração, ou seja, um nodo recebe no máximo uma aresta por iteracão, como o número total de iterações é log_2N, cada nodo em $T(S)$ recebe no máximo log_2N arestas, e $T(S)$ tem no máximo $Nlog_2N$ arestas. □

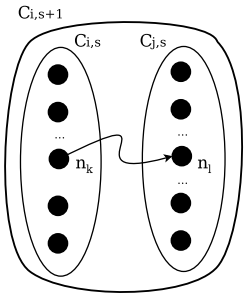


Figura 4. Aresta adicionada do nodo $n_k \in C_{i,s}$ para o nodo $n_l \in C_{j,s}$.

Teorema 3. *No HyperBone uma alteração percebida no estado de um nodo demora no máximo log_2N rodadas de testes para ser propagada para todos os nodos.*

Demonstração. Suponha que um evento ocorra no nodo e. Vamos demonstrar por indução que todo nodo no estado *working* a distância d do nodo e descobre o evento em no máximo d rodadas de testes. Como a distância máxima no grafo $T(S)$ dado pelo DiVHA é log_2N a indução é suficiente para provar o teorema.

Base: Todo nodo n_i *working* tal que $d(n_i, n_e) = 1$ descobre o evento em n_e em no máximo 1 rodada de testes. Trivial, todo nodo *working* testa todos nodos a distância 1 a cada rodada de testes.

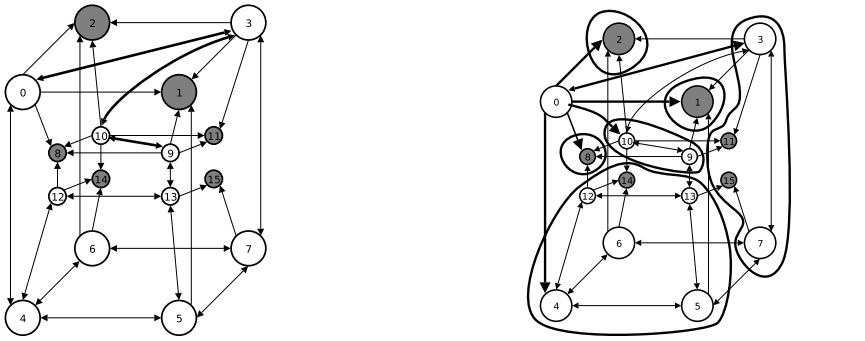
Hipótese de Indução: Todo nodo n_i *working* tal que $d(n_i, n_e) = d$ descobre o evento em n_e em no máximo d rodadas de testes.

Passo: Todo nodo n_i *working* tal que $d(n_i, n_e) = d+1$ descobre o evento em n_e em no máximo $d+1$ rodadas de testes.

Considere n_j tal que $d(n_j, n_e) < d+1$, existe portanto um caminho $\{n_j = n_0, ..., n_{d+1} = n_e\}$ com $d+1$ arestas entre o n_j e n_e. O caminho $\{n_1, ..., n_{d+1} = n_e\}$ tem d arestas, logo n_1 está a distância d do nodo e e pela hipótese de indução descobre o evento em no máximo d rodadas de testes. Como n_j é adjacente a n_1 e testa n_1 em no máximo uma rodada de testes, dado que $d(n_1, n_e) \leq log_2N$ e $d(n_1, e) < d(n_j, e)$, n_j obtém informação sobre o novo estado do nodo n_e ao tester n_1. Logo, n_j descobre o evento em n_e em no máximo $d+1$ rodadas de testes. □

2.5. Exemplos de Execução

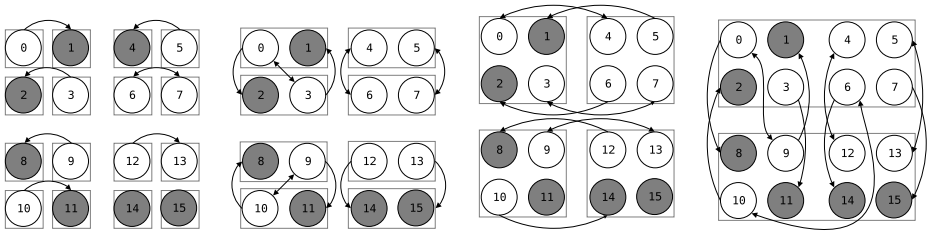
Considere um sistema com 16 nodos, e o estado inicial ilustrado na figura 5.a, na qual os nodos em cinza estão no estado *unresponsive*. As arestas finas são as arestas do hipercubo, as arestas grossas indicam arestas adicionadas pelo DiVHA. A figura 5.b mostra o fluxo de informações de estados do sistema para o nodo 0, os nodos agrupados representam o conjunto de nodos sobre os quais o nodo 0 obtém informação durante os testes, por exemplo ao testar o node 3, o nodo 0, obtém informações sobre o nodo 11, nodo 15 e nodo 7.



a:Enlaces do overlay. b:Fluxo de informações para o nodo 0.

Figura 5. Exemplo de sistema com 16 nodos executando o HyperBone.

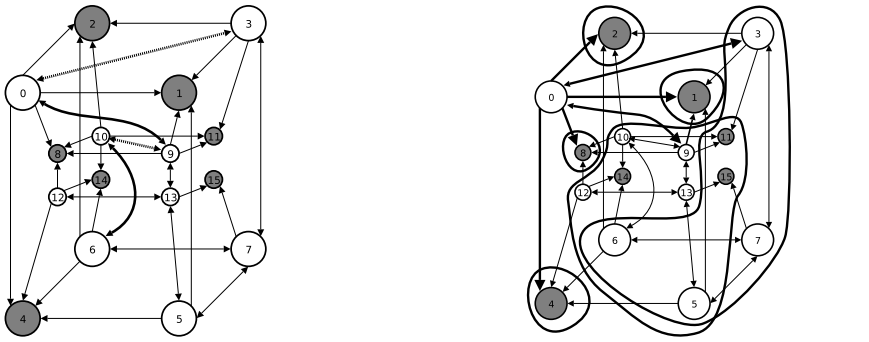
Suponha que o nodo 4 se torna *unresponsive*, cada nodo ao obter o novo estado do nodo 4 executa o DiVHA determinando a nova topologia do sistema. A figura 6 representa a execução do DiVHA a cada iteração s. Na primeira iteração são adicionadas arestas para formar caminhos com $tamanho \leq 1$ entre todo nodo i e os clusters $C_{i,0}$, essa iteração é trivial incluindo apenas arestas do hipercubo. A iteração (2) garante caminhos com $tamanho \leq 2$ para clusters $C_{i,2}$. Nessa iteração se verifica a inexistência de caminhos entre nodo 0 e nodo 3, nodo 3 e nodo 0, nodo 9 e nodo 10, e nodo 10 e nodo 9, assim são adicionadas as arestas $0 \rightarrow 3$, $3 \rightarrow 0$, $0 \rightarrow 9$, $9 \rightarrow 0$. A iteração (3) garante caminhos com $tamanho \leq 3$ para os clusters $C_{i,3}$, no caso não são necessárias arestas além das convencionais do hipercubo. Finalmente, na iteração (4) são incluídas as arestas $0 \rightarrow 9$, $9 \rightarrow 0$, $6 \rightarrow 10$ e $10 \rightarrow 6$ para garantir os caminhos de tamanho 4 entre esses nodos.



iteração 1 iteração 2 interação 3 interação 4

Figura 6. Representação das iterações do algoritmo DiVHA.

A figura 7.a mostra o hipercubo virtual final para o sistema com o nodo 4 falho. As arestas mais grossas indicam arestas adicionais, sendo que as arestas pontilhadas representam os testes adicionais que já existiam antes da falha do nodo 4. A figura 7.b mostra a nova configuração do fluxo de informações para o nodo 0.



a:Enlaces do overlay. b:Fluxo de informações para o nodo 0.

Figura 7. Exemplo de sistema com 16 nodos após falha do nodo 4.

3. Implementação e Resultados Experimentais

O HyperBone foi implementado como *daemons* de rede que executam em cada nodo agregado ao sistem. Os nodos se comunicam através de conexões TCP/IP permanentes, correspondentes aos enlaces virtuais definidos pelo algoritmo DiVHA. Os testes consistem no envio de uma mensagem para o nodo testado que deve ser respondida dentro de um período de tempo definido.

Um aspecto importante da implementação é que o HyperBone é transparente para as aplicações. Se uma aplicação deseja se comunicar através do hipercubo virtual basta que use endereços IP exclusivamente (*Internet Protocol*) alocados para o HyperBone. Por exemplo, se a rede 10.0.0.0/8 está reservada para o HyperBone, basta a aplicação usar endereços desta faixa para se comunicar usando o hipercubo virtual. Esse recurso é implementando através da interface `tun/tap` [TunTap 2005] que implementa um dispositivo de rede virtual que força o recebimento de pacotes de uma aplicação em espaço de usuário, ao invés de recebê-los do protocolo da camada de enlace.

O ambiente de experimentação escolhido foi o PlanetLab [PlanetLab 2005], que é um ambiente composto por computadores distribuídos em todo mundo conectados pela Internet, atualmente contando com 631 máquinas em mais de 25 países. Os experimento se destinaram a avaliar o HyperBone no aspecto de estabilidade, sistema de monitoração, sistema de comunicação/roteamento e capacidade de execução de aplicações. Os experimentos realizados e os resultados obtidos são descritos nas próximas subseções.

3.1. Sistema de Monitoração

Para avaliar o sistema de monitoração o HyperBone foi instalado em 128 máquinas do PlanetLab e sua execução acompanhada durante cerca de 3 semanas. Os sistema foi executado com diversos parâmetros, ao final considerando os tempos médios de resposta no PlanetLab conclui-se que um valor de timeout adequado seria de 7 segundos e o intervalo entre os inícios de duas rodadas de testes foi fixado em 10 segundos. Os resultados apresentados são relativos aos dados obtidos do registro de execução do sistema com estes parâmetros durante 6 dias.

Um dos experimentos realizados visou descobrir o tempo em que os nodos permanecem em um dado estado antes de sofrerem um novo evento. Considerando o timeout relativamente pequeno para a carga do PlanetLab, os nodos não eram capazes de atender por períodos longos e contínuos os requisitos temporais impostos pelos testes, o resultado

é que foi detectado um grande número de eventos. Registramos cerca de 200.000 mil eventos em 6 dias o que representa uma taxa aproximada de 1388 eventos/hora. O gráfico da figura 8 mostra a distribuição acumulada para o tempo de permanência nos estados *working* e *unresponsive* antes da ocorrência de um novo evento. Podemos observar que, na grande maioria dos casos, os nodos permaneceram *working* por menos de 10 minutos e *unresponsive* por menos de 120 segundos.

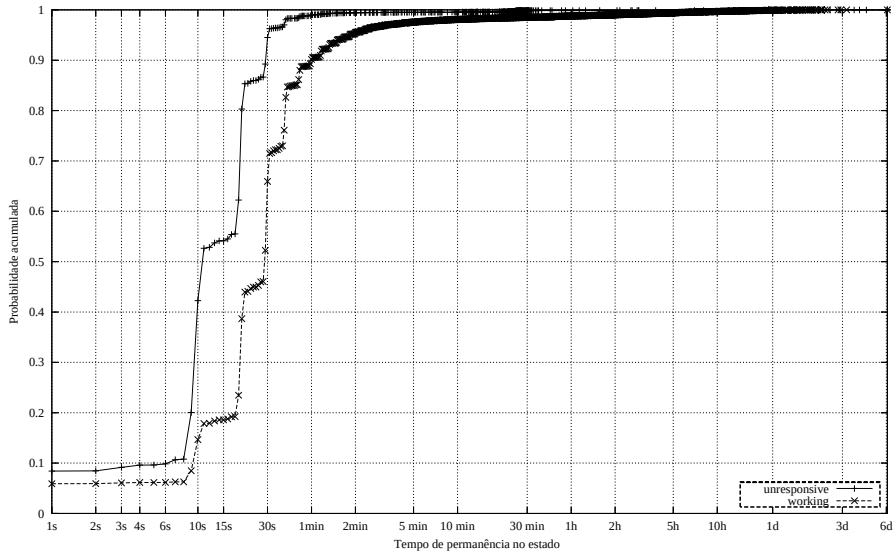


Figura 8. Probabilidade acumulada para o tempo de permanência nos estados *working* e *unresponsive*.

O grande número de eventos registrados justificou na prática a utilidade dos parâmetros de estabilização introduzidos no HyperBone. Baseado nos tempos observados no gráfico da figura 8 decidiu-se considerar um nodo *unavailable* apenas após permanecer 120 segundos no estado *unresponsive*, e considerar um nodo *available* após permanecer 600 segundos no estado *working*. Com esses parâmetros o número de eventos percebidos para os estados *unavailable* e *available* foi de apenas 1050, ou seja, uma taxa menor que 8 eventos/hora, números significativamente reduzidos considerando o resultado acima descrito.

Na prática esta forma de classificar os nodos permite descobrir o conjunto de nodos que apresentam comportamento mais estável em comparação aos outros. Pode-se dizer que aumenta a granularidade com que se exerga o estado do sistema. Por exemplo, um nodo *available* apenas deixa de ser considerado *available* se passar mais que 120 segundos como unresponsive; um nodo *unavailable* apenas será considerado *available* se passar por mais de 600 segundos sendo considerado *working*.

Desta forma, nodos que ficam silenciosos por pequeninos intervalos de tempo também são considerados estáveis. O gráfico da figura 9 mostra a probabilidade acumulada para o tempo de permanência nos estados *available* e *unavailable*. Observamos períodos maiores em que nodos permanecem em um único estado, por exemplo, em cerca de 40% dos casos os nodos permaneceram mais de 1 hora no estado *available* antes de um evento e em 80% dos casos os nodos permaneceram mais que 30 minutos no estado *unavailable*.

Outra medida obtida foi a taxa de disponibilidade dos nodos baseada nos estados

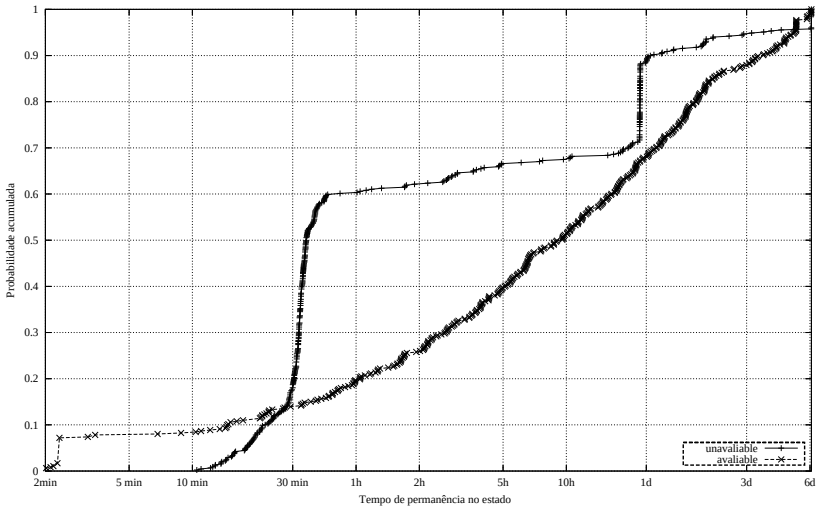


Figura 9. Probabilidade acumulada para o tempo de permanência nos estados *available* e *unavailable*.

available e *unavailable*. O histograma da figura 10 mostra a distribuição de freqüência da disponibilidade dos nodos. Apesar das instabilidades do sistema observamos que 47 nodos apresentaram disponibilidade entre 80% e 90% e 20 nodos acima de 90%. As taxas de disponibilidade muito baixas foram relativas a nodos que estavam inacessíveis ou falhos.

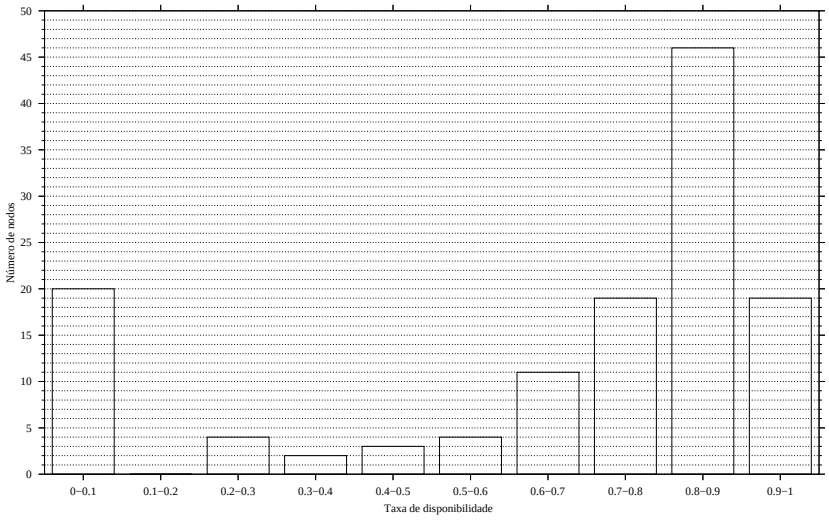


Figura 10. Histograma da taxa de disponibilidade dos nodos.

Por fim, examinamos os limites de tempo para a propagação dos eventos no hipercubo. Escolhemos um período e um nodo que estivesse alternando repetidamente de estado, e observamos o estado deste nodo a partir de outros nodos com diferentes distâncias no hipercubo. O nodo 4 teve seus eventos observados, os pontos de observação foram: nodo 5 à distância 1, nodo 7 à distância 2, nodo 77 à distância 3, nodo 118 à distância 4, nodo 44 à distância 5, nodo 122 à distância 6 e finalmente nodo 123 à distância 7, a maior possível tendo em vista o número total de nodos do hipercubo, 128. A figura 11 mostra os resultados obtidos, o relógio utilizado para registrar os eventos foi o relógio das máquinas, que estava sincronizado por NTP (*Network Time Protocol*). Pode-se compro-

var por estes resultados que os nodos obtém as informações dentro do intervalo de tempo esperado.

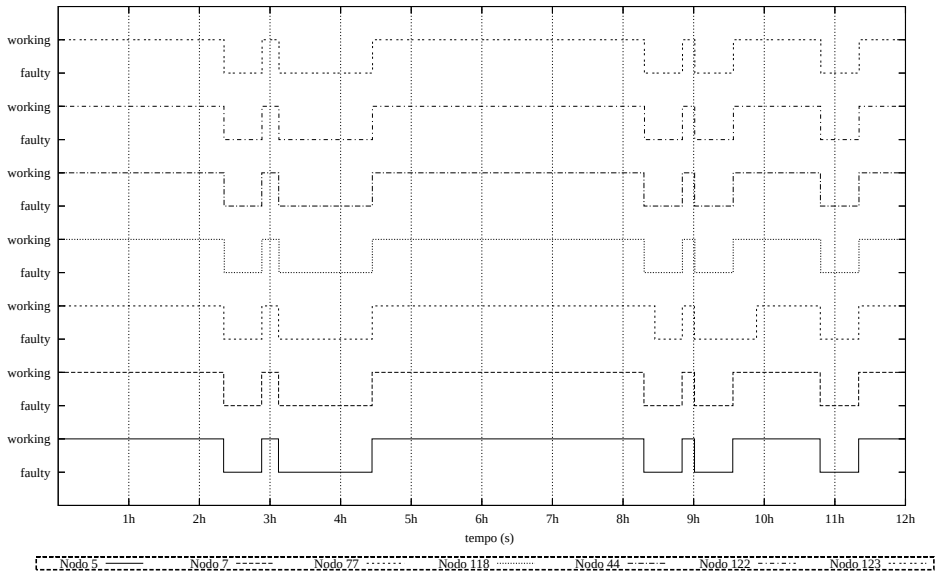


Figura 11. Propagação da informação de um eventos no nodo 4.

3.2. Executando Aplicações Paralelas

A capacidade de execução de aplicações no HyperBone foi avaliada executando aplicações paralelas usando MPI (*Message Passing Interface*) [Forum 1994]. Nestes experimentos foi utilizado um hipercubo virtual formado por 256 máquinas do PlanetLab. A cada teste, baseado nas informações de monitoração disponibilizadas pelo HyperBone, se selecionava no hipercubo virtual o conjunto de nodos que apresentava o comportamento mais estável.

O objetivo do primeiro experimento foi comprovar o funcionamento das primitivas de comunicação do MPI sobre o HyperBone. Foi utilizado o MPIbench [MPIbench 2005], um *benchmark* que exercita as principais funções do MPI, tanto de comunicação ponto a ponto (`MPI_Send` e `MPI_Receive`) como de comunicação em grupo, por exemplo difusão (`MPI_Broadcast`). A tabela 1 apresenta o resultado da execução do MPIbench. Cada coluna representa os resultados de um tipo de teste do MPIbench, as linhas mostram o tamanho de mensagens utilizado em cada teste. O resultado a ser observado não é o desempenho propriamente dito, mas sim o fato dos testes terem sido executado com sucesso e sem necessidade de modificar a aplicação mesmo num ambiente instável como o PlanetLab.

O segundo experimento realizado teve como objetivo verificar a possibilidade de executar aplicações reais em MPI usando o HyperBone. Foi executada uma aplicação em MPI que paraleliza a tarefa de encontrar uma senha em um determinado espaço de busca. A tabela 2 mostra taxa de chaves avaliada por segundo obtida em sistemas com 16 e 32 nodos.

Tamanho da mensagem	Latência μ s	Bandwidth KB/s	Broadcast KB/s	Reduce KB/s	Alltoall KB/s	AllReduce KB/s
1024	27.903999	3.544503	2.885645	1.575525	1.400713	4.113908
2048	39.040001	23.578215	4.771652	1.757580	2.615774	5.231043
4096	45.056000	12.087334	4.734047	5.409601	4.611366	4.634484
8192	56.959999	14.560066	7.056590	9.275836	23.130945	5.768235
16384	67.071999	47.326344	11.373193	9.261162	21.423141	10.898121

Tabela 1. Resultados da execução do MPIbench no HyperBone.

nodos	senhas por segundo
16	1566832695
32	1974491631

Tabela 2. Resultado da execução de um buscador de senhas no HyperBone.

4. Trabalhos Relacionados

Recentemente, muitas redes overlay têm sido propostas, especialmente para sistemas P2P, com as mais diversas topologias, como: mesh [Rowstron and Druschel 2001], anel [Stoica et al. 2001], torus d-dimensional [Ratnasamy et al. 2001] e borboleta [Fiat and Saia 2002, Saia et al. 2002]. Estas redes overlay organizam a rede subjacente de forma a facilitar a busca de objetos armazenados nos nodos que compõem o sistema. Esta estratégia melhora a escalabilidade do sistema evitando o uso de algoritmos de inundação para propagar requisições de busca, por exemplo. As tabelas hash distribuídas são uma das técnicas mais comuns empregadas para construir estas redes [Balakrishnan et al. 2003]. Estas redes apresentam uma abordagem distinta deste trabalho, estão destinadas para armazenar e recuperar conteúdo organizando a topologia do sistema em função das chaves dos objetos armazenados. O HyperBone visa oferecer serviços de comunicação e informação permitindo a execução de aplicações distribuída especialmente em sistemas instáveis. A instabilidade do sistema é abordada por alguns trabalho em P2P [Fiat and Saia 2002, Saia et al. 2002], mas novamente estes trabalhos são direcionadas especificamente para a busca de informação, o foco é como manter acessível toda ou parte da informação armazenada na rede mesmo em caso de falha dos nodos.

A questão da detecção de falhas em redes overlay é tratada em [Zhuang et al. 2005], onde são avaliadas diversas classes de algoritmos. O estudo emprega modelos analíticos, simulação e experimentação usando métricas como tempo de detecção, probabilidade de falsos positivos, overhead de controle e taxa de perda de pacotes. As conclusões deste trabalho evidenciam a importância do compartilhamento de informação de monitoração e necessidade de sistemas eficientes de monitoração.

Em [Andersen et al. 2001] são introduzidas as *Resilient Overlay Networks* (RON), cujo objetivo é oferecer alternativas para as rotas da Internet. Tendo por base informações de monitoração dos enlaces virtuais, os nodos decidem se enviam pacotes diretamente pelas rotas da Internet ou se roteam pacotes pelos nodos da RON. A arquitetura é baseado em malha completa, *full mesh*. Um problema desta topologia, apontado pelos autores, é que o sistema não é escalável para mais que 50 nodos.

A preocupação com escalabilidade aparece em [Schlosser et al. 2002], onde também é proposta uma rede overlay com topologia de hipercubo. O algoritmo pro-

posto baseia-se em posições livre e ocupadas em um hipercubo. Um nodo que funciona como peer de uma rede p2p e deseja se integrar ao sistema faz um pedido de entrada e recebe uma posição livre no hipercubo. Então, esta posição se torna ocupada até que este nodo deixe o sistema. Para manter a integridade do sistema uma mensagem de broadcast é enviada para avisar todos os nodos que uma posição do hipercubo se tornou ocupada ou vazia. Um problema no Hypercup é que o sistema somente é capaz de tratar a ocorrência de um único evento, entrada o saída de um único nodo a cada vez: a ocorrência de eventos simultâneos é apontada como trabalho futuro.

No contexto de grades computacionais também já se vê a utilização de redes overlay para tratar problemas de escalabilidade. Em [Hauswirth and Schmidt 2005] é apresentada a utilização de uma rede overlay para descobrimento de recursos, como alternativa escalável para os serviços de busca de sistemas como o Condor [Litzkow et al. 1988] e Globus [Foster and Kesselman 1997].

Quanto ao algoritmo DiVHA, podemos apontar trabalhos relacionados em diagnóstico distribuído [Masson et al. 1996]. As asserções sobre o sistema feitas pelos algoritmos de diagnóstico são bastante diferentes das feitas no DiVHA, especialmente porque os algoritmos de diagnóstico consideram que os nodos são capazes de determinar o estado de outros nodos com 100% de segurança [Preparata et al. 1968], o que é uma asserção que não pode ser cumprida em sistemas assíncrono [Fisher et al. 1985]. Por outro lado, a forma de organizar os testes e o fluxo de informação é bastante semelhante. Um algoritmo clássico de diagnóstico baseado em um anel virtual é apresentado em [Bianchini Jr. and Buskens 1991]. Em [Duarte Jr. and Nanya 1998] é apresentado um algoritmo de diagnóstico baseado em hipercubos. O algoritmo Hi-ADSD apresenta como o HyperBone diâmetro logarítmico, mas o número de arestas virtuais pode chegar a $O(N^2)$. Além disso o algoritmo foi proposto para redes locais, e não para formar hipercubos sobre a Internet.

5. Conclusão

Este trabalho apresentou o HyperBone, uma rede overlay que aproveita características topológicas do hipercubo para oferecer serviços escaláveis de comunicação e monitoração para sistema distribuídos em larga escala na Internet. O HyperBone implementa um hipercubo virtual que é reconfigurado dinamicamente para se adaptar às constantes falhas, recuperações e períodos de instabilidade apresentados pelos nodos do sistema. O algoritmo DiVHA é utilizado pelo HyperBone e permite que os nodos do sistema mantenham, de forma distribuída, independente e determinística, a topologia do hipercubo virtual, garantido duas propriedade: (1) diâmetro logarítimo, a distância entre qualquer par de nodos é menor ou igual a $log_2 N$; (2) número máximo de enlaces $N log_2 N$. O DiVHA foi especificado neste trabalho e foram apresentadas provas formais de que a topologia determinada pelo algoritmo atende às propriedades esperadas.

Os nodos do sistema utilizam o HyperBone para se comunicar, todas as mensagens trocadas entre os nodos são tuneladas e roteadas através dos enlaces do hipercubo virtual, permitindo a qualquer par de nodo se comunicar através de rotas com distância máxima igual a $log_2 N$ enlaces. O HyperBone introduz também uma estratégia para lidar com sistemas nos quais os nodos apresentam comportamento bastante instável, alternando frequentemente entre estados. É possível determinar os nodos que estão apresentando

comportamento mais estável e têm a maior probabilidade de ser utilizados para tarefas de processamento. Um ponto importante é a eficiência deste sistema que permite que todos nodos detectem um evento em no máximo $log_2 N$ rodadas de testes de $N log_2 N$ mensagens cada.

O HyperBone foi implementado e testado no PlanetLab que oferece um ambiente dinâmico com os nodos sujeitos a altas carga de processamento e rede. Resultados experimentais foram apresentados que comprovaram o funcionamento do sistema.

Trabalhos futuros incluem a implementação de primitivas de comunicação em grupo no hipercubo virtual, permitindo realizar difusão e multicast de forma eficiente. Outro trabalho futuro é a integração do HyperBone com um sistema de buscas de informação, implementando a busca de forma a tirar proveito da topologia. O roteamento no hipercubo virtual é hoje estático, uma estratégia de roteamento baseada em medições de desempenho dos enlaces virtuais também está prevista.

Referências

Amir, Y. and Danilov, C. (2003). Reliable communication in overlay networks. *IEEE International Conference on Dependable Systems and Networks (DSN 2003)*, pages 511–520.

Andersen, D. G., Balakrishnan, H., and Morris, M. F. K. R. (2001). Resilient overlay networks. *Operating Systems Review*, pages 131–145.

Balakrishnan, H., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I. (2003). Looking up data in p2p systems. *Communications of the ACM*, 46(2):43–48.

Bianchini Jr., R. and Buskens, R. (1991). An adaptive distributed system-level diagnosis algorithm and its implementation. 21st International Symposium in Fault-Tolerant Computing (FTCS-21), pages 222–229.

Clark, D. (2001). Face-to-face with peer-to-peer networking. *IEEE Computer*, 34(1):18–21.

Doval, D. and O'Mahony, D. (2003). Overlay networks: A scalable alternative for p2p. *IEEE Internet Computing*, 7(3):2–5.

Duarte Jr., E. P. and Nanya, T. (1998). A hierarchical adaptive distributed system-level diagnosis algorithmm. IEEE Transactions on Electronic Computers, 47(1):34–45.

Eriksson, H. (1994). Mbone: The multicast backbone. *Communications of the ACM*, 37:54–60.

Fiat, A. and Saia, J. (2002). Censorship resistant peer-to-peer content addressable networks. *13th ACM-SIAM Symposium on Discrete Algorithms*, pages 94–103.

Fisher, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. Journal of the ACM, 32(2):374–382.

Forum, M. P. I. (1994). Mpi: A message-passing interface standard. *International Journal of Supercomputer Applications*, 8(3/4):165–414.

Foster, I. and Kesselman, C. (1997). Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128.

Hauswirth, M. and Schmidt, R. (2005). An overlay network for resource discovery in grids. *16th International Workshop on Database and Expert System Applications (DEXA'05)*, pages 343–348.

Krull, J. M., Wu, J., and Molina, A. M. (1992). Evaluation of a fault-tolerant distributed broadcast algorithm in hypercube multicomputers. *20th ACM Annual Computer Science Conference*, pages 459–462.

Litzkow, M., Livny, M., and Mutka, M. (1988). Condor - a hunter of idle workstations. *8th International Conference of Distributed Computing Systems (ICDCS'88)*, pages 104–111.

Masson, G. M., Blough, D., and Sullivan, G. F. (1996). *Fault-Tolerant Computer System Design*. Prentice-Hall.

MPIbench (2005). Mpibench home page. http://icl.cs.utk.edu/projects/llcbench/mpbench.html. Acesso em: dez. 2005.

OurGrid (2005). Ourgrid home page. http://www.ourgrid.org/. Acesso em: dez. 2005.

PlanetLab (2005). Planetlab: An open plataform for developing, deploying, and acess planetary-scale services. http://www.planet-lab.org. Acesso em: dez. 2005.

Preparata, F., Metze, G., and Chien, R. (1968). On the connection assignment problem of diagnosable systems. IEEE Transactions on Electronic Computers, 16:848–854.

QBone (2005). Qbone home page. http://qbone.internet2.edu. Acesso em: dez. 2005.

Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S. (2001). A scalable content addressable network. *ACM SIGCOMM*, pages 161–197.

Rowstron, A. and Druschel, P. (2001). Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350.

Saia, J., Fiat, A., Griblle, S., Karlin, A. R., and Saroiu, S. (2002). Dynamically fault-tolerant content addressable networks. *1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, pages 270–279.

Schlosser, M., Sintek, M., Decker, S., and Nejd, W. (2002). Hypercup - hypercubes, ontologies and p2p networks. *Lecture Notes on Computer Science*, 2530:112–124.

Stoica, I., Morris, R., Karger, D., Kaashoek, F., and Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM*, pages 149–160.

Tien, S. and Raghavendra, C. S. (1993). Algorithms and bounds for shortest paths and diameter in faulty hypercubes. *IEEE Transactions on Parallel and Distributed Systems*, 4(6):713–718.

TunTap (2005). Universal tun/tap driver. http://vtun.sourceforge.net/tun/. Acesso em: dez. 2005.

Tzeng, N. F. and Chen, H. L. (1994). Structural and tree embedding aspects of incomplete hypercubes. *IEEE Transactions on Computers*, 43(12):1434–1439.

Zhuang, S., Geels, D., Stoica, I., and Katz, R. H. (2005). On failure detection algorithms in overlay networks. *24th IEEE Infocom 2005*, pages 13–17.