MoBiDiCk: Uma ferramenta para computação distribuída na internet

Marcos Camponogara[1], Tiago Silva da Silva[1], Tiago Thompsen Primo[1]

[1]Pontifícia Universidade Católica do Rio Grande do Sul(PUCRS) – Faculdade de Informática – Av. Ipiranga 6681, Porto Alegre, RS – Brasil - 90619-900

{mcamponogara,silvadasilva,tiagoprimo}@gmail.com

Resumo. *Neste artigo aborda-se um apanhado geral sobre o MoBiDick (Modular big distributive kernel) uma abordagem baseada em CGI para computação distribuída e paralela. O MoBiDick é um sistema orientado a base de dados, que através do protocolo http, e clientes CGI é utilizado para realizar processamento de tarefas com o uso de diversos processadores distribuídos. Sua motivação foi solucionar o problema de protein folding.*

1. Introdução

O MoBiDick, foi desenvolvido com o intuito principal de ajudar no problema de *protein folding,* que seria o processo pelo qual uma proteína assume sua forma, ou configuração funcional. Todas as moléculas de proteínas são cadeias não ramificadas de aminoácidos, mas colidindo-as em uma forma tri-dimensional específica elas se tornam hábeis a executar sua função biológica. Segundo (Dharsee et al, 2000), autor do artigo principal sobre o tema, eles não estão sozinhos nesta busca de recursos computacionais, citando que a IBM está desenvolvendo um supercomputador especificamente para atuar na área que busca soluções de alto desempenho para resolver o problema de *protein folding*.

Os autores do projeto MoBiDick não podem ser considerados como novatos, já que mencionam o fato de trabalharem com o desenvolvimento de softwares para tratar o problema de *protein folding* por algum tempo, tendo como resultado expressivo o desenvolvimento de um método para a geração plausível de proteínas em espaço real. Detalhes sobre este trabalho podem ser observados em (Feldman & Hogue, 2000).

O desenvolvimento do MoBiDick teve como idéia fazer do uso de outro sistema chamado FOLDTRAJ em milhares de computadores distribuídos. O FOLDTRAJ é uma aplicação desenvolvida usando o kit de ferramentas do National Center for Biotechnology Information(NCBI), este o qual, possui códigos fontes de diversas aplicações como a Entrez, uma base de dados integrada sobre bioinformática; BLAST, uma ferramenta para procurar DNA e seqüências de proteínas em bases de dados; Cn3D uma ferramenta para visualizar estruturas de moléculas em três dimensões; Incluindo diversas outras ferramentas e fontes de informações. Todas estas funcionando sobre o protocolo HTTP, fato que facilitou a comunicação com o MoBiDick.

Sobre o usa da computação distribuída, um exemplo de sua exploração em projetos complexos, seria o projeto SETI@home, que demonstrou que milhares de processadores

distribuídos pela internet podem cooperar em problemas complexos, e que as pessoas gostam de compartilhar processamento de CPU para tais causas. Detalhes podem ser observados em (SETI@Home).

O objetivo a ser alcançado com o desenvolvimento do MoBiDick é de utilizar do poder computacional dos mais diversos tipos de computadores, sejam eles: servidores; computadores pessoais; clusters; estações de trabalho entre demais, desde que estes possuam conexão com a internet e a possibilidade de instalação de um software servidor web.

2. Projeto do Sistema

2.1 Arquitetura do Sistema

O MoBiDick usa de tecnologias CGI para computação distribuída e paralela. Ele opera sobre um conjunto de nodos interconectados a uma rede TCP/IP. Um nodo é simplesmente um computador com rede que pode atuar como um servidor WEB.

Um nodo pode vir a ser uma estação de trabalho, um cluster ou uma estação multiprocessada. Requisitos de recursos locais, tais como espaço em disco, memória, banda de rede e de I/O são restritas a computações em particular. Ele tem como premissa, desenvolver aplicações que necessitem o mínimo de consumo destes recursos locais.

Na configuração mais simples que pode ser usada pelo MoBiDick, o *Kernel Server* é implementado de forma que possa atingir a todos os nodos em processamento através de requisições HTTP.

Cinco módulos de kernel são instalados junto ao Kernel do servidor: *Dispatcher, Status, Statekeeper, Collector e DataManager.*

A Figura 1 apresenta uma visão geral da arquitetura do sistema.

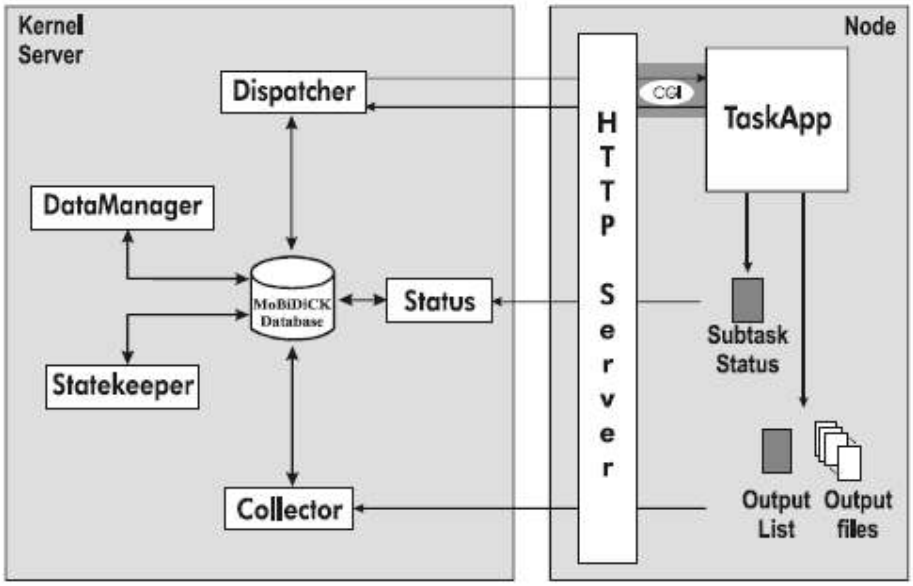

Figura 1. Visão geral da arquitetura do sistema

As interações entre os módulos CGI, envolvem apenas 1 ou 2 requisições HTTP. Por exemplo, quando uma computação é requisitada por um nodo, o *dispatcher* envia a requisição para o servidor web do nodo, e está feita a requisição. Apenas no momento que a sub-tarefa será executada, o nodo envia a informação de inicio para o Kernel de servidor. Isto faz com que seja executado um novo processo no *dispatcher* que irá salvar esta informação. Enquanto isto o *taskApp* continua sua computação.

O gerenciamento dos dados, é feito através de uma base de dados relacional, implementada com o CodeBase (Sequier Software), que quando integrada aos módulos de Kernel, provêem um sistema de base de dados relacional, funcional e independente de plataforma.

O objetivo do gerenciamento de dados é armazenar informações referentes a todos os dados do sistema, incluindo informações descritivas e estatísticas relacionadas a nodos e suas computações, valores dos parâmetros de entrada e os arquivos de saída.

A modularidade do Kernel, permite que os recursos de CPU e I/O sejam otimizados. Funções de gerência de sistema podem ser distribuídas através de um espalhamento dos módulos de Kernel em diversos servidores. Por exemplo, uma configuração de Kernel distribuída, pode envolver quatro servidores, onde o *dispatcher* é instalado em um determinado servidor, o *collector* em um segundo, *status e statekeeper* em um terceiro, e o *data manager* em um quarto. A base de dados tem cada parte montada em um servidor e é compartilhada dentre todos os servidores. O uso de um servidor de Kernel multiprocessado, é uma configuração alternativa que pode vir a beneficiar banda de CPU já que os módulos de Kernel podem rodar simultaneamente em diversos processadores.

2.2 Modelo de comunicação

Os módulos do MoBiDick se comunicam através de requisições ‘’GET’’ e ‘’POST’’, provenientes do protocolo HTTP. A Figura 2 ilustra o modelo geral de comunicação entre o módulo m_1 funcionando através do servidor S_1 pertencentes ao nodo N_1 com o módulo m_2 funcionando através do servidor S_2 pertencente ao nodo N_2.

Os passos envolvidos na transmissão da mensagem de m_1 até m_2 são os seguintes:

1. m_1 abre uma conexão TCP, C, entre N_1 e N_2
2. m_1 envia uma requisição http contendo a mensagem, M, para m_2 através de C
3. S_2 recebe a requisição, inicia m_2, e passa a requisição para m_2
4. m_2 extrai M da requisição, processa M, e da um retorno R
5. S_2 captura R, coloca um cabeçalho HTTP e o envia através de C
6. m_1 recebe a resposta de S_2 e lê R
7. m_1 fecha C

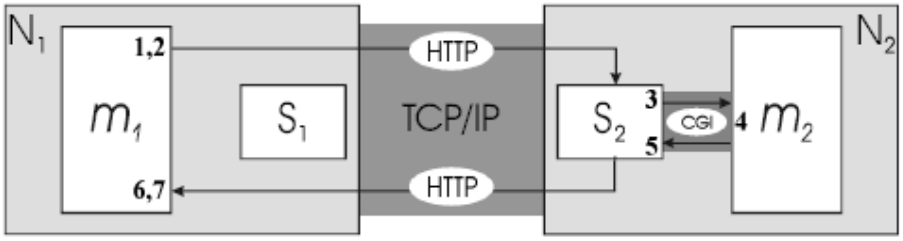

Figura 2. Modelo Geral da comunicação

2.3 Registro de um nodo

Um nodo de processamento pode ser registrado e agendado (escalonado) através de um *browser*, invocando o módulo *Datamanager* para gravar as informações na base de dados. O Registro consiste em fornecer atributos-chave, tais como *host* ou endereço IP, tipo e velocidade da CPU, número de CPUs, sistema operacional, capacidade de disco e memória principal, e também informações para contato (em caso de processador voluntário).

Uma vez registrado, o processador pode ser agendado (escalonado) selecionando quais horários e dias da semana que ele executará as tarefas. Pode-se especificar também quais tarefas são permitidas que o processador execute.

Este é um modelo de agendamento (escalonamento) *all-or-none*: o processador participa da computação apenas quando seu agendamento (escalonamento) permite. O modelo *all-or-none* é indicado para pessoas que desejam disponibilizar uma quantidade de tempo de CPU conhecida, ou seja, saber o quanto sua CPU está sendo utilizada, por exemplo, utilizar computadores de um escritório fora do horário comercial.

2.4 Execução da tarefa

Uma computação distribuída é solicitada ao módulo *Dispatcher* através de um *browser*. Uma vez invocado, o *Dispatcher* dá inicio a uma seleção de nodo interativa

(*interactive selection node*), compilando uma lista de nodos candidatos à execução de alguma tarefa. Para se candidatar a uma tarefa, o nodo tem que cumprir as seguintes condições:

- Registro: o nodo é registrado na base de dados;
- Participação: o nodo é registrado para participar de alguma tarefa;
- Acessibilidade: acesso ao nodo momentaneamente (currently) permitido pelo agendamento (escalonamento) do nodo;
- Conectividade: o nodo é (reachable) pelo *Dispatcher*;
- Configuração: o *TaskApp* operacionalizado no nodo.

O Registro e a Participação são feitos apenas com uma consulta na base de dados: para ser utilizado, um nodo deve estar registrado na base de dados e a tarefa direcionada a ele deve aparecer na lista de participação do nodo indicando sua voluntariedade para executar tal tarefa.

A Acessibilidade é determinada pela agenda (escala) de acesso do nodo a fim de verificar se o nodo está no momento aceitando requisições e se ele permanecerá disponível por tempo suficiente para a execução da tarefa.

A Conectividade e a Configuração são feitas em um único "aperto de mão" (*handshaking*), através do qual o *Dispatcher* envia requisição "teste" para o *TaskApp* em um nodo. Se nenhuma resposta é recebida do nodo, ou um erro é encontrado no estabelecimento da conexão, então nenhuma das condições é encontrada (suprida). Se o servidor Web do nodo responder com um erro, então a condição de Conectividade é alcançada, mas a condição de Configuração falha no nodo.Se uma resposta válida é recebida do *TaskApp* conclui-se que o servidor Web está apto para rodar o *TaskApp* com sucesso, e assim o nodo cumpre as duas condições, de Conectividade e Configuração.

Outras condições de seleção podem ser adicionadas pelos usuários como restrições adicionais, tais como valor de corte (*cut-off*) para classificação do nodo, espaço de armazenamento, memória total e memória temporária, e número de CPUs. A seleção pode ser restrita, restringida por categorias específicas de nodos, tais como, local, remoto, dedicado ou compartilhado. A seleção manual de nodos específicos também pode ser feita como caminho alternativo às condições anteriores.

Após um nodo ser selecionado, o *Dispatcher* é estimulado a realizar a fase de mapeamento da tarefa (*task mapping phase*), dividindo a tarefa em sub-tarefas e mapeando as sub-tarefas para os nodos selecionados. Mais de uma sub-tarefa pode ser designada a um nodo, assim como pode ser solicitado para um nodo multiprocessado. O relacionamento sub-tarefa-CPU de um nodo pode ser definido nas informações de registro e podem ser modificados pela prioridade de envio do usuário.

O mapeamento da tarefa assume um passo de carga inicial de equilíbrio (*initial load balancing*) que computa a carga de cada tarefa baseada na associação performance-classificação do nodo. Carga da sub-tarefa (*subtask load*) representa uma parte da carga de trabalho que pode ser distribuída para o nodo o qual a sub-tarefa está mapeada.

A classificação do nodo pode ser obtida rodando um módulo de (benchmarking) *TaskApp* que mede a capacidade do nodo em ponto-flutuante e inteiro aritmético, acesso a memória, espaço em disco, e comunicação. Sabendo-se a classificação do nodo, pode-se calcular a carga da sub-tarefa.

Para um conjunto de nodos de processamento $p_1, p_2, ..., p_n$ com respectivas classificações $r_1, r_2, ..., r_n$, a classificação de sobrecarga (*weighted rating*) para p é:

$$R_i = r_i / \sum(r_1, ..., r_n) \quad (1)$$

A carga de sub-tarefa é obtida pela divisão da classificação de sobrecarga (*weighted rating*) pelo número de sub-tarefas designadas ao nodo. Portanto, a carga de sub-tarefa para o nodo p_i que está designado a k sub-tarefa é:

$$L_i = R_i / k \quad (2)$$

Na etapa de alocação de parâmetros (*parameter allocation*), parâmetros de tarefas podem ser herdados pela sub-tarefa e valores são designados a esses parâmetros. O valor designado a um parâmetro de sub-tarefa depende primeiramente do tipo de parâmetro. Se ele é constante então basta impor o valor definido para a tarefa. Se for um parâmetro variável com uma partição com alcance numérico, então o valor é encontrado pela aplicação da carga a este alcance. Por exemplo, dado um parâmetro numérico, X, com um alcance delimitado pelo limite inferior x_L e limite superior x_U, o valor de X para o nodo p_i é:

$$X_i = L(x_U - x_L) \quad (3)$$

O processo de mapeamento pode ser repetido se o usuário desejar modificar as condições de candidatura do nodo ou os atributos e parâmetros de uma tarefa. Na solicitação do usuário, o *Dispatcher* inicia a computação enviando solicitações de sub-tarefa (*subtask requests*) para os nodos selecionados. Uma solicitação de subtarefa tem sucesso se o *Dispatcher* receber uma mensagem de confirmação do módulo *TaskApp*. Se todas as solicitações tiverem sucesso, a instância de tarefa está então no estado ativo (*active state*).

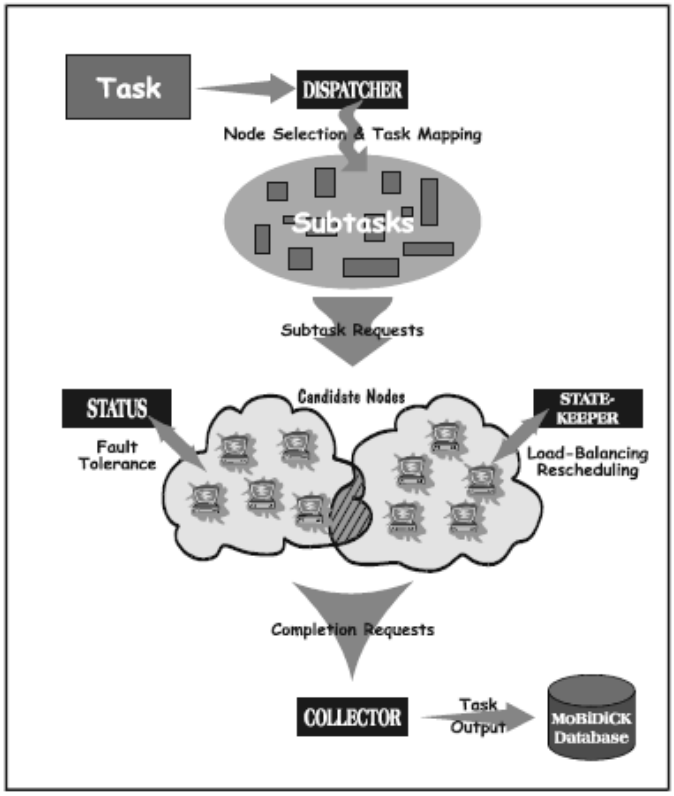

Figura 3. Execução da Tarefa

Se a tarefa é executada com sucesso, o *Dispatcher* invoca um novo processo de Status e um novo processo de *StateKeeper* para monitorar processos *TaskApp*. Durante esta execução, uma *TaskApp* regularmente atualiza um arquivo local de *SubtaskStatus* com informações sobre o progresso da sub-tarefa. Tolerância a falhas durante a computação é assegurada pelo módulo *Status* fazendo *downloads* periodicamente do arquivo *SubtaskStatus* de cada nodo. Se a subtarefa falhar em um nodo, o módulo Status redireciona a sub-tarefa. A função do *Statekeeper* é monitorar o agendamento (escalonamento) de *overflows* do nodo e manter o equilíbrio de carga (*load-balancing*) dinâmico. Podendo re-agendar (re-escalonar) subtarefas através do término de subtarefas e iniciando novas, dessa forma, mantendo a computação no estado *all-or-none* conforme o agendamento (escalonamento) de cada nodo na base de dados. Ele pode re-mapear toda a tarefa para um novo conjunto de nodos, caso muitos conflitos sejam encontrados durante a computação ou caso a disponibilidade do nodo mude drasticamente.

Arquivos locais de saída produzidos pelo *TaskApp*, são gravados em um arquivo *OutputList*. Quando o *TaskApp* conclui uma subtarefa, ele envia uma solicitação de conclusão (*completioin request*) ao *Collector*. O *Collector* atualiza o status da sub-tarefa na base de dados, obtém a *OutputList* do nodo, e reúne os dados na base de dados, caso seja solicitado. O *Collector* é capaz de armazenar objetos de dados arbitrários como arquivos binários na base de dados, e (*iterator*) que é uma rotina para acessar itens em uma matriz são fornecidos na API para sumarizar ou combinar resultados uma vez enviados, logo, as tarefas estão completas. Depois que toda saída foi recebida, o *Collector* envia uma (*cleanup request*) requisição de limpeza para o *TaskApp*,

solicitando a ele que apague os arquivos de saída produzidos pelo sistema de arquivos do nodo, informado na *OutputList* do *TaskApp*.

2.5 API (Application Program Interface) da tarefa

Módulos de tarefas são programados usando a *Task API*, esta mesma integrada com as bibliotecas do kit de ferramentas da *NCBI*. A *Task API* facilita o desenvolvimento independente de plataforma, aplicações CGI as quais podem ser operadas tanto de forma executável *stand-alone* como uma interface regular de linha de comando, e como programas CGI que podem ser invocados por um servidor Web, quando esta recebe a requisição HTTP de cliente para rodar uma aplicação. Em outras palavras, o uso da Task API não restringe a aplicação ao sistema MoBiDiCK. Usando o mesmo executável, o usuário pode escolher executar a computação manualmente, ou instalar o programa por trás dos servidores HTTP em um conjunto de nodos possibilitando então que a tarefa possa vir a ser distribuída usando o kernel do MoBiDiCK.

3. Resultados

3.1 RAMAPLOT

Usando a API Task foi desenvolvido um módulo TaskApp chamado RAMAPLOT. Este programa usa estruturas tridimensionais de proteínas, provenientes da base de dados de modelagem molecular (*Molecular Modeling Database* - *MMDB*) do NCBI (*National Center for Biotechnology Information*) para gerar o gráfico de Ramachandran, que é um gráfico da distribuição dos ângulos de rotação de proteínas de carbono α. A tarefa foi realizada com 15 nodos, cada nodo configurado com dois processadores Intel Pentium II 400MHz, 512Mb de memória, sistema operacional Linux RedHat e servidor HTTP apache. O servidor principal que foi utilizado era um Sun Sparc Ultra-1 executando solaris 2.6. A base de dados MMDB foi copiada para o disco rígido de cada nodo.

O objetivo da tarefa era de gerar um gráfico de Ramachandran para cada uma das 851 estruturas de proteína da base de dados. A tarefa RAMAPLOT utiliza dois parâmetros de entrada, *dbsize* e *dbstart*. O parâmetro *dbsize* representa o número de registros a ser processado, que varia de 1 a 851 (primeiro e último registro da base de dados, respectivamente), e o parâmetro *dbstart* define o número do registro inicial na base de dados. Foram realizados testes com 15 instâncias da tarefa RAMAPLOT, iniciando com um único nodo e adicionando um novo nodo para cada nova instância, sendo que cada nodo possuía somente uma CPU. Para cada instância foram especificados o tempo de execução, o *speedup*, e a eficiência, sendo que os resultados estão concentrados na Tabela 4.1 e exibidos nas Figuras 4.1, 4.2 e 4.3.

Instance	Subtasks	Time (s)	Speedup	Efficiency
1	1	714	0.972	97.2
2	2	378	1.84	92.0
3	3	265	2.62	87.2
4	4	195	3.57	89.1
5	5	165	4.20	84.1
6	6	144	4.82	80.3
7	7	116	6.01	85.8
8	8	106	6.55	81.9
9	9	107	6.47	71.9
10	10	96	7.19	71.9
11	11	79	8.79	79.9
12	12	87	8.00	66.7
13	13	76	9.09	69.9
14	14	66	10.49	74.9
15	15	69	10.07	67.1

Tabela 4.1- Resultados do RAMAPLOT usando o MoBiDiCK.

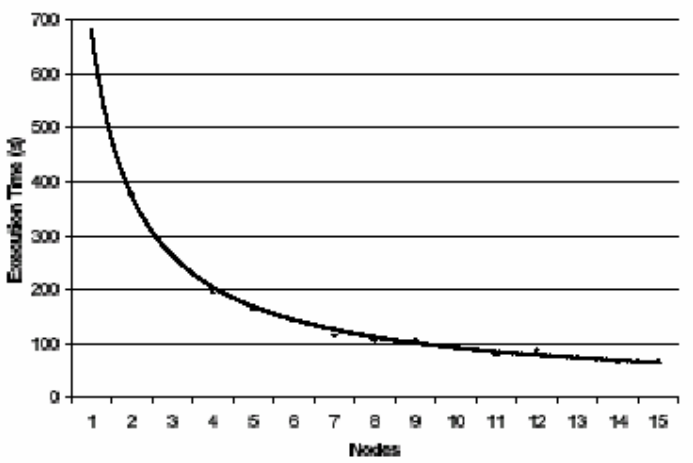

Figura 4.1 – Tempo de execução da tarefa RAMAPLOT.

A Figura 4.2 representa o *speedup*, que é calculado através da razão entre o tempo de execução paralelo e o melhor tempo de execução serial, que foi de 695 segundos. Foi obtido um incremento gradual do *speedup* através do aumento no número de nodos.

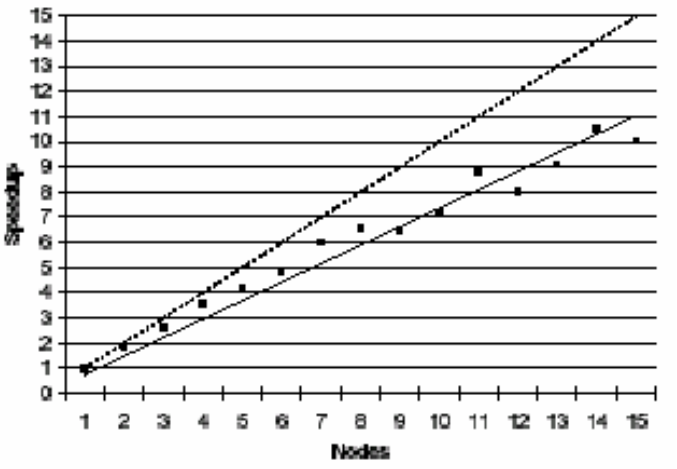

Figura 4.2. Speedup para o RAMAPLOT.

A linha tracejada representa o speedup ideal

O aumento no número de sub-tarefas fez a probabilidade de duas sub-tarefas acabarem ao mesmo tempo aumentar, o que leva a um aumento na contenção do tráfego de rede e de I/O no servidor principal, pois cada sub-tarefa finalizada gera uma chamada ao Collector. Isto gerou uma diminuição na eficiência à medida que o número de nodos aumentou, como pode ser visto na Figura 4.3.

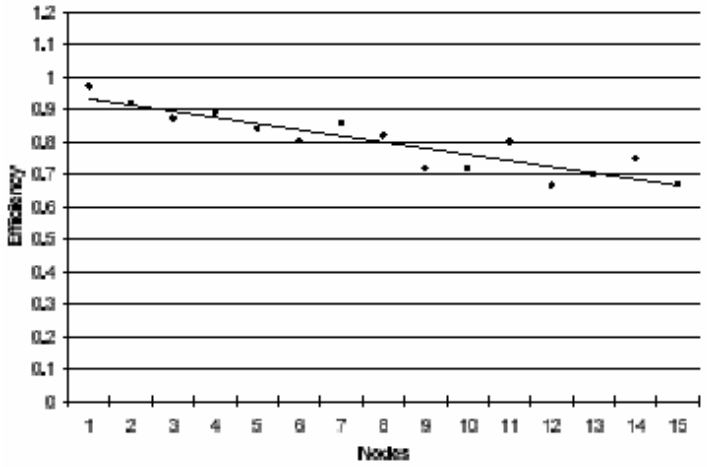

Figura 4.3 – Eficiência do RAMAPLOT.

3.2 FOLDTRAJ

O FOLDTRAJ foi desenvolvido para ser usado como uma aplicação *stand-alone* e também em um *framework* para computação distribuída. A API Task foi utilizada para migrar o FOLDTRAJ e fazer com que este funcione sob o MoBiDiCK. Esta aplicação é utilizada no problema do dobramento de proteínas, que provém do campo da biologia estrutural. Antes das proteínas poderem executar sua função bioquímica, elas se auto constroem, ou "dobram" (fold). O processo de dobrar proteínas, apesar de crítico e fundamental para virtualmente toda a biologia, seu funcionamento ainda é desconhecido. Adicionalmente, quando as proteínas não dobram corretamente, podem ocorrer sérios efeitos, incluindo muitas doenças bem conhecidas como Alzheimer, Vaca Louca, Parkinson e esclerose lateral amiotrófica.

Dado um arquivo de entrada conhecido como “distribuição de trajetória”, contendo informação de freqüência em um espaço angular 2D de um amino ácido sobre uma proteína particular, o FOLDTRAJ pode gerar um número randômico de proteínas quimicamente válidas, colocando cada uma em um arquivo separado em formato binário ASN.1 ou ASCII PDB. A corretude de uma estrutura prevista é medida pelo cálculo da RMSD (*Root Mean Squared Deviation*) relativo ao dobramento nativo das proteínas.

A tarefa para o FOLDTRAJ possui três parâmetros significantes: *filein*, *numstruc* e *fstart*. O parâmetro *filein* é uma string constante que representa o nome do arquivo de entrada, o parâmetro *numstruc* é o número total de estruturas a ser geradas, que varia de 1 a 500.000, o que significa que meio milhão de estruturas podem ser geradas. Finalmente o parâmetro *fstart* denota o número da estrutura inicial, que pode variar como o *numstruc*. Um exemplo de definição de tarefa e mapeamento é mostrado na figura 4.4.

O FOLDTRAJ foi executado regularmente, usando o MoBiDiCK para a realização de experimentos de previsão para várias proteínas. A Figura 4.5 exibe o resultado de 4 experimentos. Para cada um, 50.000 estruturas de proteínas foram geradas usando 15 nodos bi-processados de um cluster. O valor da distribuição de freqüência do RMSD (Root Mean Squared Deviation) resultante indica a corretude da previsão do átomo da estrutura da proteína feito com o FOLDTRAJ.

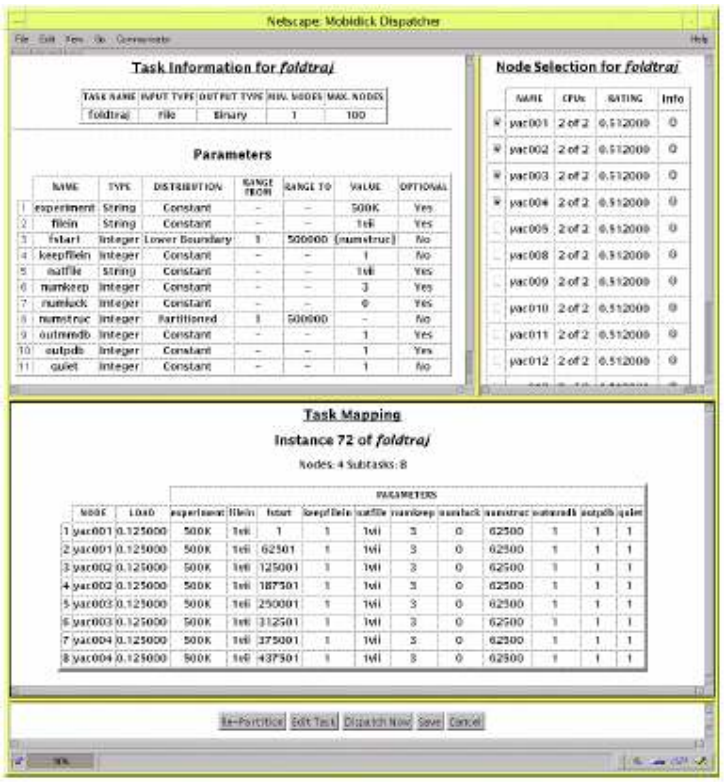

Figura 4.4 – Exemplo de definição e mapeamento de tarefa do FOLDTRAJ.

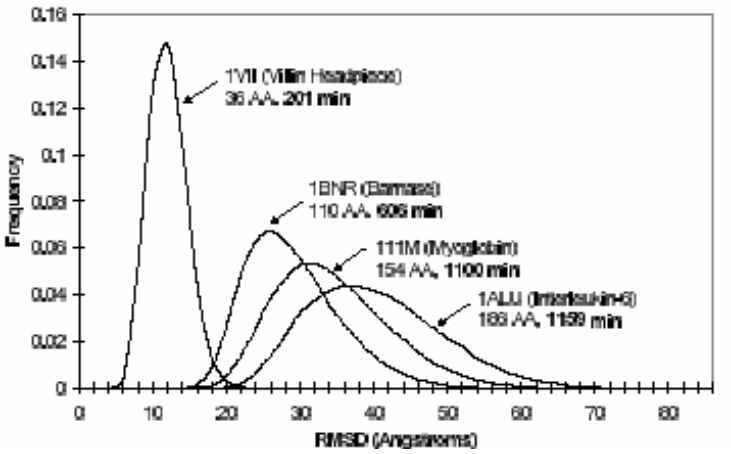

Figura 4.5 - Distribuição de freqüência de estruturas
de proteínas geradas pelo FOLDTRAJ.

5 Conclusões

O presente artigo tratou do MoBiDick um software desenvolvido com a intenção de tratar do problema de *protein folding,* este referente a junção de proteínas até que estas atinjam uma forma funcional. O sistema propõe o uso de outro sistema já existente o FOLDTRAJ, mas com um uso diferente do seu original, ou seja, o processamento distribuído entre máquinas.

Foram apresentados dois experimentos, um em que foi desenvolvida uma ferramenta chamada de RAMAPLOT, onde esta, fez testes com relação à geração de modelos tridimensionais de proteínas, e um com uma ferramenta chamada FOLDTRAJ, onde foram realizados testes sobre o processo de protein folding em si. Acredita-se que testes comparativos precisam ser realizados como uma forma de validar de forma mais concreta os resultados apresentados.

References

Dharsee, M. Hogue, C.W.V. MoBiDiCK: A Tool for Distributed Computing on the Internet. In Proceedings, 9 th Heterogeneous Computing Workshop, May 2000. http://citeseer.ist.psu.edu/dharsee00mobidick.html

H. J. Feldman and C. W. Hogue. A fast method to sample real protein conformational space. Proteins, 39(2):112--131, 2000.

Search for Extra-Terrestrial Intelligence; SETI@home website, http://www.seti.org/science/setiathome.html

Sequier Software inc. CodeBAse (Version 6): Database Management for Programmers. Getting Started, p 1-7