RECIIS
Revista Eletrônica de Comunicação Informação & Inovação em Saúde
[www.reciis.cict.fiocruz.br]
ISSN 1981-6278

SUPLEMENTO – BIOINFORMÁTICA E SAÚDE

Artigos originais

Projeto e implementação de uma ferramenta para o escalonamento de processos e acesso aos dados biológicos

DOI: 10.3395/reciis.v1i2.Sup.99pt

Maíra Ferreira de Noronha
Departamento de Informática da Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brasil
maira@inf.puc-rio.br

Sérgio Lifschitz
Departamento de Informática da Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brasil
maira sergio@inf.puc-rio.br

Resumo
Este trabalho discute o projeto e implementação de uma ferramenta que provê dados voltados para programas de Bioinformática, denominada BioProvider. A ferramenta utiliza políticas ad-hoc de gerenciamento de buffer que lidam com dados de seqüências muito grandes e possibilita o controle do escalonamento de processos. Foi usada uma abordagem não intrusiva para viabilizar a utilização da ferramenta com os programas aplicativos de uso corrente pelos usuários. Neste trabalho, BioProvider foi instanciada para funcionar com o programa BLAST de comparações de seqüências.

Palavras-chave
Escalonamento de processos, dados biológicos, bioinformática, BioProvider

Introdução

Este trabalho propõe uma estratégia de implementação não intrusiva para uma ferramenta provedora de seqüências biológicas que gerencia eficientemente o *buffer* e permite controlar o escalonamento dos processos. A ferramenta – denominada *BioProvider* - é usada de maneira transparente aos aplicativos pois a comunicação com estes é realizada por meio de um *device driver* que substitui as chamadas das funções de leitura e escrita de arquivos do banco de dados do sistema operacional.

Uma das tarefas mais importantes na análise dos dados de Biologia Molecular é a de comparação de seqüências, base para várias outras manipulações mais elaboradas. Informações acerca da funcionalidade de genes e proteínas, a localização dos genes nos cromossomos, entre outras informações, são inferidas através de comparações com seqüências conhecidas, armazenadas junto com suas anotações em bancos de dados como o Genbank (Benson et al., 2005) e o SWISS-PROT (SIB, 2007). Os programas da família BLAST (ALTSCHUL et al., 1990) são os mais utilizados para tais comparações.

Assim, instancia-se neste artigo a ferramenta BioProvider para uso com programas BLAST. Por ser transparente aos programas, a ferramenta aqui apresentada é facilmente extensível, podendo ser modificada para prover dados para outros aplicativos, usar outras políticas de substituição de páginas na gerência de *buffer* ou mesmo prover dados armazenados em formatos diferentes dos lidos por processos clientes, convertendo-os em tempo de execução. Em particular, neste trabalho mostramos uma grande vantagem da abordagem não intrusiva: BioProvider pôde ser usada com duas implementações distintas do programa BLAST, também as mais populares, NCBI BLAST (NCBI, 2007) e WU-BLAST (WU, 2007).

Este artigo está organizado da seguinte forma: na próxima Seção, discute-se uma proposta de gerenciamento de buffer específica para o BLAST, objetivo de uma das instanciações da *BioProvider* aqui realizadas. Depois, na Seção 3, a arquitetura da ferramenta é descrita em mais detalhes, já levando em consideração a política de substituição de páginas mais adequada. A Seção 4 traz um detalhamento da parte experimental no que diz respeito ao escalonamento dos processos. Por fim, a Seção 5 traz alguns resultados que ilustram as vantagens do uso de BioProvider quando instanciada para o BLAST e a Seção 6 encerra o artigo com as conclusões.

Gerenciamento de Buffer para o Blast

O algoritmo do BLAST consiste em uma heurística para o alinhamento local de biosseqüências, e é usado para comparar seqüências de consulta com bancos de dados de seqüências. Os bancos de dados lidos pelo BLAST possuem um formato específico (WU, 2007) e são compostos em geral por 3 arquivos: um de seqüências (.psq), um de anotações associadas às seqüências (.phr) e um com índices relacionando as seqüências às suas anotações (.pin).

O algoritmo básico do BLAST possui três etapas e seu funcionamento é detalhado em Altschul et al. (1990). Na segunda etapa do BLAST, toda a base (ou arquivo) de seqüências é varrida. Esta é também a etapa na qual há um maior número de acessos ao disco. Já que o BLAST lê diretamente de arquivos do sistema operacional, a leitura das seqüências do banco nesta etapa do algoritmo é feita de maneira ineficiente em algumas situações muito comuns. Esta ineficiência ocorre quando o arquivo de seqüências não pode ser mantido inteiramente na memória e diversos processos são executados ao mesmo tempo, já que não são usadas técnicas de gerenciamento de *buffer* para aproveitar partes do banco de seqüências presente em memória. Deste modo, a execução de cada processo deverá fazer com que parte do banco de seqüências seja lida mais de uma vez do disco para a memória durante a segunda etapa do algoritmo.

A segunda etapa do algoritmo BLAST possui as seguintes características:

• as seqüências do banco de dados são lidas de maneira seqüencial. Logo, é possível prever as próximas páginas que serão necessárias aos processos e trazê-las para o *buffer* antes mesmo de serem requisitadas, realizando o *prefetching*;

• vários processos podem acessar o banco de dados ao mesmo tempo e, por isso, é desejável que eles compartilhem páginas do *buffer*;

• não é importante a ordem em que é realizada cada comparação da seqüência de consulta com as seqüências do banco de dados. Por isso um processo BLAST pode começar sua comparação por qualquer seqüência do banco desde que, ao final da execução, todas as seqüências tenham sido utilizadas na comparação.

Devido às características descritas, foi sugerida em LEMOS et al. (2003) uma estratégia *adhoc* de gerenciamento de *buffer* para o BLAST. São usadas estruturas de armazenamento de seqüências na memória, denominadas *anéis*. Estes consistem de *buffers* na memória para o qual o arquivo de seqüências do banco vai sendo carregado aos poucos, sendo sua política de atualização circular, semelhante à política de substituição de páginas FIFO. Um anel é lido pelos vários processos BLAST de maneira compartilhada, sendo que as atualizações de partes do anel ocorrem quando todos os processos em execução sendo atendidos já tiverem consumido a informação disponível.

Uma das vantagens desta estratégia é permitir que cada processo BLAST, quando iniciado, possa começar a ler o arquivo de seqüências a partir do ponto em que este se encontra disponível no anel, já que a ordem de leitura das seqüências não é importante. Como o início do arquivo de seqüências é carregado novamente para o anel ao final de um ciclo de leitura, o novo processo lerá todas as seqüências anteriores à seqüência inicialmente considerada, completando a leitura do arquivo.

Esta estratégia de uso eficiente do *buffer* é interessante para ser oferecida com o BLAST, pois, além de possibilitar o *prefetching*, permite que vários processos compartilhem um *buffer*, diminuindo o tempo das leituras de páginas contendo seqüências do disco para a memória. Este gerenciamento de *buffer* especial será instanciado na ferramenta BioProvider, como será detalhado a seguir.

Abordagem não-intrusiva de BioProvider

Pode-se implementar a ferramenta BioProvider de forma que os aplicativos da biologia molecular sofram modificações (intrusiva) em seu código ou de forma transparente e independente dos programas de aplicação (não-intrusiva), sem alteração do código.

A maneira intrusiva substitui, no código dos aplicativos, cada chamada às funções de leitura de seqüências por outras funções que se comunicam com um processo provedor de dados, que por sua vez irá realizar o gerenciamento de memória e do *buffer*. Para a implementação não-intrusiva no código foi sugerida, em MAURO et al. (2005) a utilização de um *driver* de dispositivo que simule o funcionamento dos arquivos do banco de dados e realize a comunicação entre os processos BLAST e o processo provedor.

Um *driver* de dispositivo é uma camada de *software* do sistema operacional que possibilita a comunicação de aplicativos com dispositivos de *hardware* e *software*, escondendo a maneira como é realizada a comunicação direta com os dispositivos. A idéia proposta em MAURO et al. (2005) consiste em substituir os arquivos do banco de dados biológicos (seqüências) por arquivos especiais associados a um *driver* de dispositivo de caractere. Ao executar funções de abertura e leitura destes arquivos, os aplicativos da bioinformática chamam na realidade as funções implementadas pelo *driver*, que realizam a comunicação com o processo provedor de banco de dados.

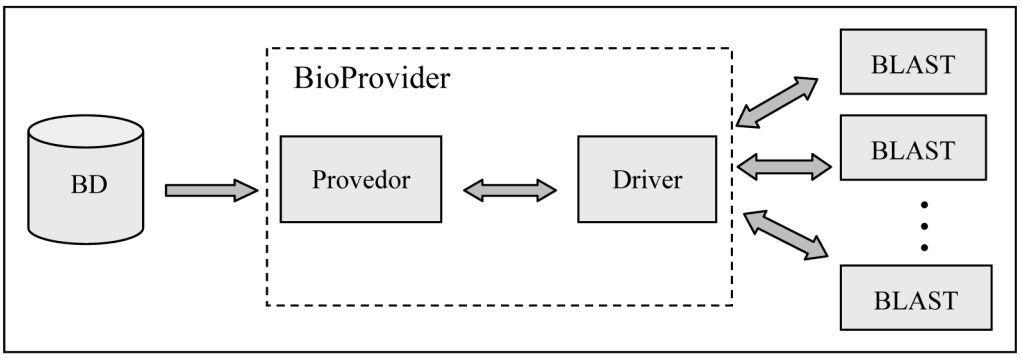
Neste trabalho, optou-se por implementar o gerenciamento de *buffer* de maneira não-intrusiva através da instanciação de um *driver*. Deste modo, não é necessário modificar o código do BLAST (aplicação teste) para

acrescentar o gerenciamento de *buffer*. Além disso, a ferramenta criada pode ser usada para diferentes versões do BLAST sem alterações no código da ferramenta.

Houve a opção pela implementação de um *driver* para o sistema operacional Linux como um módulo do núcleo (*kernel)*. Os arquivos do banco de dados são substituídos por arquivos de dispositivos associados ao *driver*. Ao executar funções de abertura e leitura destes arquivos, o programa BLAST chama na realidade as funções implementadas pelo *driver*, que realizam a comunicação com o processo provedor de banco de dados e controlam ao mesmo tempo a concorrência e o bloqueio de processos.

A Figura 1 mostra a arquitetura básica da BioProvider e seu uso com BLAST.

Figura 1 – Arquitetura do BioProvider



Na arquitetura utilizada pelo BioProvider, um processo provedor do banco de dados é o responsável pelo gerenciamento de um anel de memória e o atendimento a requisições de leitura do banco de dados feitas pelos processos BLAST. Já as funções implementadas pelo *driver* são usadas para a comunicação entre os processos BLAST e o processo provedor. Os arquivos do banco de dados do BLAST são substituídos por arquivos especiais associados ao *driver*. Deste modo, ao ler estes arquivos como se fossem o banco de dados, os processos BLAST executam na realidade as funções implementadas pelo *driver*. O processo provedor conhece a verdadeira localização dos arquivos do banco de dados e fornece seus conteúdos aos processos. Estes aguardam bloqueados pela informação requisitada e são acordados pelo processo provedor quando a informação estiver disponível. Logo, o processo provedor possui também o papel de escalonador dos processos BLAST à espera de respostas de leitura, pois o mesmo deve escolher qual processo será atendido em cada momento.

O gerenciamento de *buffer* realizado pelo processo provedor é semelhante ao apresentado na seção 2. A estratégia utiliza apenas um anel em memória, sendo que a existência de múltiplos anéis poderá ser implementada em versões futuras. Para o anel é lido, aos poucos, o arquivo de seqüências, que possui todas as seqüências do banco de dados em formato compactado, uma seguida da outra. Os processos BLAST que estão na etapa de leitura seqüencial do arquivo iniciam a leitura do mesmo pelo conteúdo já presente no anel. Deste modo, se o conteúdo do início do arquivo, correspondente ao início da primeira seqüência, não estiver no anel quando o processo o requisitar, o processo provedor enviará como resposta o conteúdo da posição de início de alguma seqüência que já esteja no anel.

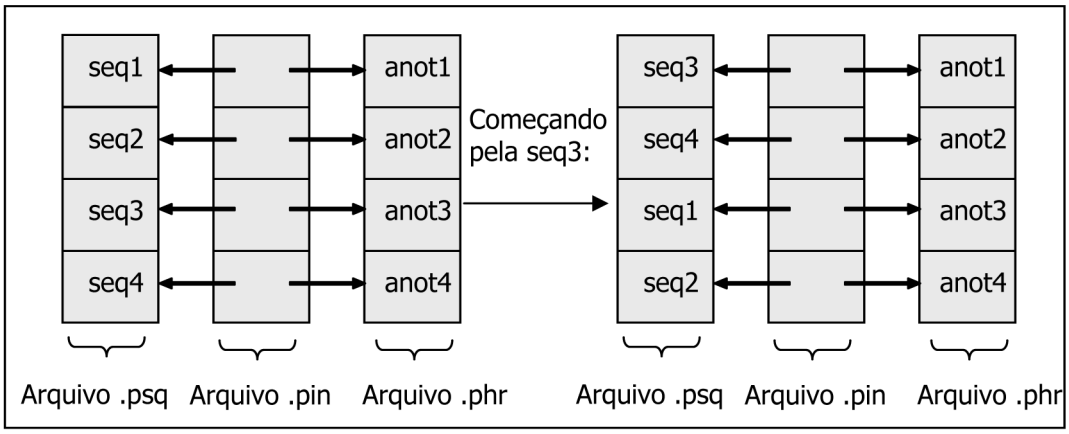
É necessário que os processos BLAST comecem a leitura do arquivo pelo início de alguma seqüência, caso contrário, interpretarão partes de seqüência como seqüências inteiras. Para isto, o processo provedor deverá identificar os inícios de seqüências no arquivo, o que pode ser feito identificando-se o caractere separador de seqüências, que é igual nas duas versões de banco de dados estudadas. Uma vez que a leitura do arquivo de seqüência foi iniciada, todos os pedidos de leitura subseqüentes do mesmo receberão como resposta o conteúdo de uma posição defasada do arquivo, somando-se a posição requisitada à posição pela qual o processo iniciou sua leitura. Depois de receber o conteúdo do final do arquivo de seqüências, o próximo conteúdo recebido será o do início do arquivo, de modo que o processo leia também o conteúdo anterior à posição pela qual sua leitura foi iniciada. Quando todos os processos sendo atendidos tiverem consumido a informação de uma posição do anel, esta é atualizada com um novo conteúdo do arquivo de seqüências.

Uma característica importante desta estratégia de gerenciamento de *buffer* é que a informação a ser requisitada é prevista e copiada para o *buffer* antes da requisição. A atualização do *buffer* em memória é feita de maneira semelhante à estratégia FIFO. Entretanto, regiões de tamanhos aleatórios do *buffer* podem ser atualizadas de uma só vez, ao invés da atualização ser feita em páginas de tamanho fixo.

Outra observação relevante quanto à esta estratégia de gerenciamento de *buffer* é que os processos podem receber uma informação diferente da requisitada, diferenciando-se das técnicas tradicionais de gerenciamento de *buffer*. Deste modo, os processos são "enganados", o que é necessário para acrescentar o gerenciamento de *buffer* de maneira não intrusiva no código do BLAST. Os processos que iniciaram a leitura do arquivo de seqüências de uma posição diferente da primeira terão uma visão do arquivo diferente da verdadeira. Este fato gera problemas caso o processo provedor e o *driver* controlem a leitura apenas do

arquivo de seqüências, enquanto que outros arquivos do banco de dados são fornecidos diretamente pelo sistema operacional, sem modificações. Como o arquivo de índices possui ponteiros para o arquivo de seqüências, estes poderão estar apontando para posições erradas do arquivo de seqüências "enxergado" pelos processos BLAST. O arquivo de índices associa as seqüências no arquivo de seqüências às anotações no arquivo de anotações, logo, esta associação será feita de modo errado pelos processos. Este erro pode ser visto na Figura 2.

Figura 2 – Erro de ponteiros do arquivo de índices

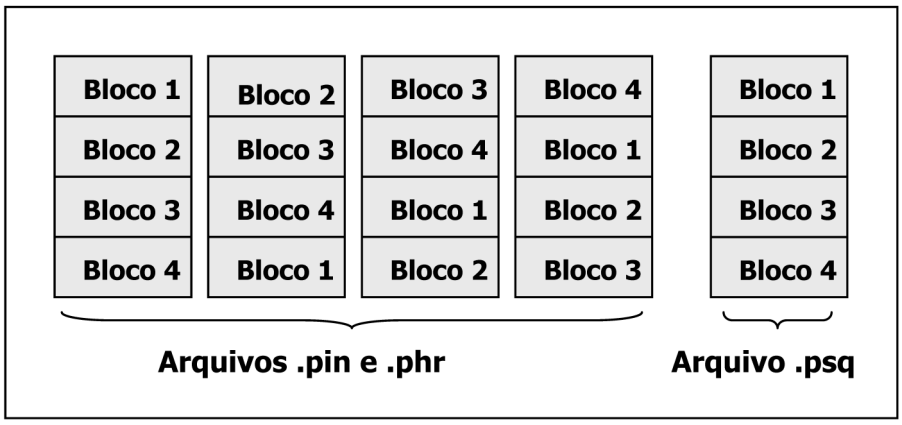


Uma possível solução para este problema seria controlar também a leitura do arquivo de índices, modificando em tempo de execução seu conteúdo ao enviá-lo a cada processo BLAST, de modo a corrigir seus ponteiros para o arquivo de seqüências. Esta solução traz alguns problemas, como um maior gasto de tempo de processamento e memória. A conseqüência mais grave é a necessidade de conhecimento do formato do arquivo de índices para modificá-lo, pois seu formato varia mais entre diferentes implementações e versões do BLAST que o formato do arquivo de seqüências. Logo, o programa servidor de banco de dados teria que ser modificado para funcionar com cada implementação do BLAST que leia arquivos de índices em um formato diferente.

Uma solução alternativa seria utilizar a ferramenta do BLAST de geração do banco de dados a partir do mesmo em formato FASTA para criar, de antemão, os arquivos do banco de dados correspondentes a diferentes "imagens" do arquivo de seqüências que os processos BLAST poderão ter, dependendo da ordem das seqüências que estes lerão. Para isto, basta modificar a ordem das seqüências no arquivo FASTA antes de gerar as diferentes instâncias do banco de dados com a ferramenta fornecida junto ao BLAST, guardando todos os arquivos menos os de seqüências, que só terá uma instância. Para não criar um número muito grande de instâncias do banco de dados, é importante limitar as posições do arquivo de seqüências a partir das quais os processos podem iniciar a leitura. Isto pode ser feito organizando as seqüências em n blocos de m seqüências. Os processos BLAST só poderão iniciar a leitura do arquivo de seqüências a partir do início de um bloco. A leitura dos outros arquivos do banco de dados deve ser controlada também pelo BioProvider, sendo que diferentes instâncias dos arquivos devem ser fornecidas a cada processo BLAST, dependendo do bloco pelo qual sua leitura do arquivo de seqüências foi iniciada.

A etapa de pré-formatação do banco de dados é ilustrada na figura 3. Supondo-se a divisão das seqüências em 4 blocos, os processos BLAST podem ler as seqüências do arquivo em 4 diferentes ordenações. Estas ordenações correspondem às sucessões de blocos 1-2-3-4, 2-3-4-1, 3-4-1-2 e 4-1-2-3. Para cada ordem são criados arquivos de índices e anotações correspondentes, que são fornecidas aos processos BLAST.

Figura 3 – Pré-formatação do banco de dados



Apesar de aumentar o uso de espaço em disco, a utilização desta estratégia torna a implementação do programa provedor independente dos formatos dos arquivos de índices e anotações, já que apenas o formato do arquivo de seqüências deve ser conhecido pelo processo provedor. Logo, esta foi a estratégia utilizada neste trabalho. Como o arquivo de índices é bem menor que os outros arquivos e o arquivo de anotações só é acessado para se obter anotações das seqüências mais similares à seqüência sendo comparada com o banco, o fato de processos lerem de diferentes instâncias destes arquivos não aumenta muito o número de leituras do disco para a memória realizadas pelo sistema operacional.

Escalonando processos

Além de gerenciar o anel em memória, o processo provedor é responsável por decidir qual processo deverá ser atendido a cada momento e a partir de qual posição do anel. Foram implementadas algumas heurísticas para otimizar este atendimento, como dar preferência a processos que estão mais atrasados na leitura do anel ou que estejam pedindo dados de outros arquivos do banco. Um objetivo importante é garantir que todos os processos possam terminar suas execuções, não havendo *starvation*, o que pode ocorrer se novos processos são iniciados a todo momento, bloqueando a atualização do anel. Este problema é resolvido atrasando-se, de tempos em tempos, o início do atendimento de novos processos, para possibilitar a atualização do anel durante o período do atraso e, conseqüentemente, o avanço dos processos já sendo atendidos.

Devido às características do programa implementado e dos processos BLAST, alguns fatores possuem grande influência no desempenho dos processos. Os fatores mais importantes são o tamanho do banco de dados acessado e o número de processos BLAST executando de maneira concorrente. Se o banco de dados couber inteiramente na memória, o próprio sistema operacional deverá acessá-lo no disco apenas uma vez e mantê-lo em memória enquanto houver outros processos acessando. Esta é a situação ótima, na qual os processos terão melhor desempenho, não sendo necessária a utilização da ferramenta para gerenciamento de buffer. Por outro lado, se o banco de dados for grande e não puder ser mantido inteiramente na memória, os processos sofrerão grande perda de desempenho, pois parte do banco de dados deverá ser copiada novamente para a memória quando diferentes processos acessarem o mesmo banco. A ferramenta criada diminui muito o número de cópias do disco para a memória quando há vários processos em execução concorrentes, obrigando os processos a lerem sempre a parte do banco que já está na memória, sendo acessada por outros processos. Quanto maior for o banco e o número de processos executando, maior é a vantagem da utilização da ferramenta, pois maior é a probabilidade dos processos requisitarem dados que não estão disponíveis em memória.

Outros fatores que influenciam o desempenho dos processos BLAST são o tamanho do anel em memória e os tamanhos dos blocos nos quais o arquivo de seqüências é dividido. O anel deve ser grande o suficiente para armazenar parte do conteúdo do arquivo de seqüências que ainda não foi requisitado, não precisando de atualizações a todo momento, e conter sempre algum início de bloco. Entretanto, se o tamanho do anel for muito grande, este poderá interferir no desempenho do sistema e de outros processos, com a necessidade de utilização de memória virtual ou *swapping*. O tamanho desejável dos blocos do arquivo de seqüências depende do tamanho do anel. Se não houver sempre algum início de bloco no anel, é possível que novos processos BLAST demorem a começar a ler do banco de dados, já que deverão esperar o aparecimento de algum início de bloco no anel. Entretanto, se o número de blocos for muito grande, haverá várias instâncias de arquivos de índices e anotações, que poderão ocupar um espaço muito grande em disco.

A estratégia adotada pelo processo provedor para o atendimento dos pedidos de leitura e escalonamento dos processos é de extrema importância no desempenho destes. Logo, foram implementadas diferentes estratégias com algumas variações nas prioridades dadas aos processos. Cada vez que o provedor recebe pedidos de leitura, a escolha do processo a responder é feita segundo alguns critérios, mostrados a seguir.

1. Se a prioridade for sempre dada a processos lendo do anel, é possível que o tempo total de execução dos processos seja menor, pois leituras do anel são mais rápidas que leituras do disco. Entretanto, esta alternativa não é muito vantajosa. Como cada processo lê dados que não estão no anel tanto antes quanto depois da fase de leitura do anel, se houver sempre outros processos lendo do anel, é grande a possibilidade de processos atrasarem o início da leitura do anel ou demorarem para iniciar a etapa seguinte à de leitura do mesmo. Logo, as diferentes estratégias implementadas são de dar prioridade a pedidos de processos que não estão lendo do anel ou alternar esta prioridade.

2. Após a escolha entre dar prioridade aos processos na fase de leitura do anel ou fora desta, é escolhido o processo que realizou o pedido de menor posição do arquivo ou, caso os processos estejam lendo do anel, é escolhido o mais atrasado na sua leitura. Deste modo, os processos mais atrasados na leitura do anel não bloqueiem sua atualização por muito tempo.

3. É importante garantir que os processos BLAST em execução não fiquem bloqueados eternamente à espera da resposta ao seu pedido de leitura. Entretanto, a partir do momento em que um processo BLAST começa a ser atendido, este bloqueia a atualização do anel durante um tempo, atrasando os processos mais adiantados na leitura. Isto acontece porque o bloco do arquivo de seqüências pelo qual sua leitura será iniciada é definido quando este receber a resposta à primeira leitura, já que o processo deverá ler o arquivo de índices correspondente ao bloco, e o processo só começará a ler do anel após ler o arquivo de índices. Logo, se novos processos iniciarem sempre em períodos muito curtos, a atualização do anel ficará sempre parada. Uma solução para este problema é, de tempos em tempos, deixar de atender os processos que estão realizando o primeiro pedido de leitura.

Finalmente, é importante destacar a influência das seqüências de entrada no desempenho dos processos. De modo geral, os processos BLAST que recebem seqüências de entrada maiores têm maior tempo de execução, pois são encontradas mais semelhanças com as seqüências do banco e realizadas mais extensões nas comparações. Entretanto, seqüências de entrada muito semelhantes às existentes no banco também tendem a produzir o mesmo efeito. Como conseqüência, alguns processos BLAST serão mais rápidos que outros e, ao mesmo tempo que deverá haver processos sempre adiantados na leitura do anel e esperando sua atualização, haverá outros atrasados bloqueando a atualização do anel. Este problema pode ser minimizado se existirem múltiplos anéis em memória, pois processos com velocidades semelhantes poderão ler do mesmo anel. Entretanto, esta solução não foi implementada até o momento. Se for possível saber de antemão as seqüências de entrada que serão usadas em comparações separadas (uma comparação por processo BLAST), uma solução seria ordenar as seqüências de entrada pelo tamanho e executar os processos BLAST na ordem obtida ou em grupos separados, cada um comparando seqüências de tamanhos semelhantes.

Resultados

O BioProvider foi implementado inicialmente para o Linux com versões de núcleo 2.6 e 2.4, tendo sido testado nas versões 3 e 4 do Linux Fedora e no Linux Red Hat 8. Seus principais módulos são um programa provedor de dados, um *driver* e um programa que finaliza a execução do provedor. Devido a semelhanças nos formatos dos arquivos de seqüências das versões recentes do NCBI BLAST (2.0 em diante) e 1.4 do WU-BLAST, o BioProvider funciona com ambas as versões do BLAST. São incluídas ferramentas para a automatização do pré-processamento do banco de dados e a criação de arquivos de configuração. Alguns parâmetros podem ser especificados, como o tamanho do anel na memória e o número de seqüências por bloco.

Foram realizados testes para avaliar o desempenho do NCBI BLAST usando o BioProvider. Os testes foram feitos em um computador Pentium 4 com processador de 3 GHz de velocidade e 512 *megabytes* de memória RAM. Foi usado o banco de dados *nr*, de proteínas, disponível em (NCBI, 2007). O banco *nr* é um dos mais importantes para os biólogos e possui informações de diversos organismos. Seu arquivo de seqüências de proteínas possui aproximadamente 1,2 *gigabytes*. Foram avaliados os tempos de execução de 50 processos BLAST para diferentes configurações de tamanho de memória RAM e tamanho de anel, sendo que cada processo foi iniciado 1 minuto depois do anterior. O banco de dados *nr* foi dividido em 5 blocos com números iguais de seqüências. Foi possível limitar a memória da máquina para diferentes valores modificando-se o arquivo de configuração do programa GRUB (*boot loader*). Foram selecionadas aleatoriamente 50 seqüências pertencentes aos bancos de dados *ecoli. aa, swissprot e ptaa,* e cada processo BLAST utilizou uma destas como seqüência de consulta. As Tabelas 1 e 2 mostram os resultados obtidos.

Tabela 1 – Resultados da execução normal do BLAST

Tamanho da memória	256M	512M
Tempo médio de execução	2:40:57	1:11:15

Tabela 2 – Resultados da execução do BLAST usando o BioProvider

Tamanho da memória	256M			512M		
Tamanho do anel	25M	50M	100M	25M	50M	100M
Tempo médio de execução	27:55	29:24	42:11	24:05	34:45	25:26
Melhora de desempenho	83%	82%	74%	66%	51%	64%

Verifica-se que o desempenho dos processos usando o BioProvider foi muito melhor, variando entre 51% e 83% de melhora. Diversos outros resultados igualmente positivos foram obtidos e podem ser analisados e estudados em NORONHA (2006).

Conclusões

Uma versão inicial do BioProvider foi implementada para prover dados ao BLAST de maneira eficiente, realizando um gerenciamento de *buffer*, controle dos dados e escalonamento dos processos, que leva em consideração características específicas do BLAST no acesso ao banco. O BioProvider foi implementado de maneira não-intrusiva no código do BLAST, o que foi possível devido à utilização de um *driver* para realizar a comunicação com os processos BLAST. Logo, não é necessário modificar o código do BLAST para usar a ferramenta, e esta funciona ao mesmo tempo com o NCBI BLAST e o WU-BLAST 1.4. Além disso, o BioProvider pode ser facilmente extendido para prover futuramente soluções de bancos de dados a outras ferramentas de Bioinformática.

Nos testes realizados com o NCBI BLAST, foi possível observar diversas situações nas quais houve uma melhora de desempenho do BLAST utilizando-se o BioProvider. Foi possível verificar também a influência de alguns fatores no desempenho do BLAST com o BioProvider. Em trabalhos futuros, este pode ser estendido em diversos pontos, listados a seguir:

• inclusão do fornecimento ao BLAST de arquivos de bancos de dados de nucleotídeos. Poderão ser usadas estratégias de gerenciamento de *buffer* e escalonamento de processos semelhantes às já criadas.

• criação e análise de desempenho de outras estratégias de atendimento de processos e gerenciamento de *buffer* a serem usadas com o BLAST. Uma possibilidade é a criação de múltiplos anéis em memória.

• utilização de outras técnicas de fornecimento do banco de dados ao BLAST, como a criação dinâmica do arquivo de índices a ser fornecido a cada processo. Isto tornaria desnecessária a divisão do banco de dados em blocos e a criação de diferentes instâncias dos arquivos de índices e anotações durante o pré-processamento do banco.

• extensão do BioProvider para prover soluções de bancos de dados para outros aplicativos de Bioinformática. Uma das extensões consiste em fornecer à ferramenta FASTA técnicas de gerência de *buffer* semelhantes às já estudadas.

O código do BioProvider, assim como explicações de uso, podem ser obtidos no site http://www.inf.puc-rio.br/~blast.

Referências bibliográficas

ALTSCHUL, S. et al. Basic Local Alignment Search Tool. **Journal of Molecular Biology**, n.215, p.403-410, 1990.

BENSON, D. A. et al. GenBank. **Nucleic Acids Research**, v.33, p.D34-D38, 1 jan. 2005.

LEMOS, M.; LIFSCHITZ, S. A Study of a Multi-Ring Buffer Management for BLAST. In: International Workshop on Biological Data Management, 1. DEXA, 2003, Prague, **Proceedings**, 2003, p.5-9.

MAURO, R.; LIFSCHITZ, S. An I/O Device Driver for Bioinformatics Tools: the case for BLAST. **Genetics and Molecular Research (GMR)**, v.4, n.3, p.563-570, 2005.

NCBI. NCBI BLAST. Disponível em: http://www.ncbi.nlm.nih.gov/BLAST/. Acesso em: 2007.

NORONHA, M.F. **Controle da Execução e Disponibilização de Dados para Aplicativos sobre Seqüências Biológicas**: o Caso BLAST. Dissertação (mestrado), Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2006.

SIB. Swiss-Prot and TrEMBL. Disponível em: http://bo.expasy.org/sprot/. Acesso em: 2007.

WU. WU-BLAST. Disponível em: http://blast.wustl.edu. Acesso em: 2007.

Sobre os autores

Maíra Ferreira de Noronha

Graduada em Engenheira de Computação (2003) e Mestre em Informática (2006), ambos pela Pontifícia Universidade Católica do Rio de Janeiro. Possui experiência de pesquisa em Bancos de Dados, Bioinformática e Inteligência Computacional, atuando principalmente nas áreas de Bancos de Dados para Bioinformática, Auto-Sintonia de Bancos de Dados, Algoritmos Genéticos e Redes Neurais. Trabalha atualmente como Analista de Sistemas na Petrobras.

Sérgio Lifschitz

Engenheiro Elétrico (1986), mestre em Engenharia Elétrica – Sistemas (1990) ambos pela Pontifícia Universidade Católica do Rio de Janeiro e doutor em Informática – Bancos de Dados – pela École Nationale Supérieure des Télécommunications, ENST Paris, França (1994). Atualmente é professor associado (adjunto S14) da PUC-Rio alocado no Departamento de Informática. Tem atualmente duas linhas de pesquisa principais dentro da área de bancos de dados: (1) computação autônoma e sistemas contemplando auto-sintonia (self-tuning) e (2) ferramentas e sistemas de gerência de dados para aplicações em bioinformática.