3
O BioProvider

O BioProvider foi implementado com o objetivo, primeiramente, de prover dados ao BLAST realizando um gerenciamento de *buffer* eficiente para este e controlando também o escalonamento dos processos. Com o desenvolvimento da ferramenta, objetiva-se criar uma solução aplicável também a outras ferramentas de Bioinformática.

O BioProvider leva em consideração características específicas do BLAST no acesso ao banco para torná-lo mais eficiente. Foram usadas algumas idéias propostas em Lemos et al. (2003) para realizar o gerenciamento de *buffer*. Para permitir a utilização da ferramenta em diferentes versões e implementações do BLAST, assim como uma futura extensão da ferramenta para funcionar com outros aplicativos de Bioinformática, optou-se por implementar a comunicação com a ferramenta de maneira não intrusiva no código do BLAST. Para isto, foi implementado um programa provedor de dados que se comunica com os processos BLAST por meio de um *driver*, como sugerido em Mauro et al. (2004).

Analisando-se o comportamento de diferentes versões e implementações do BLAST, assim como os formatos dos arquivos do banco de dados, foram criadas estratégias de fornecimento dos arquivos de maneira genérica, de modo a atender às versões de BLAST estudadas. Foram criadas também estratégias de atendimento e escalonamento dos processos BLAST de maneira eficiente, garantindo ao mesmo tempo que todos os processos em execução possam ser atendidos.

3.1.
A Estratégia Utilizada

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. A arquitetura simplificada do BioProvider é mostrada na figura 5.

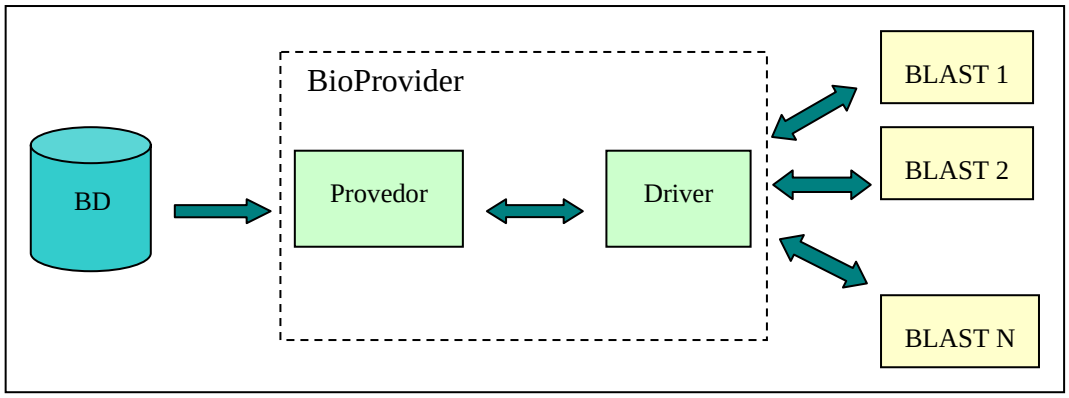

Figura 1: Arquitetura do BioProvider

Os arquivos do banco de dados do BLAST são substituídos por arquivos especiais associados ao driver. Deste modo, ao ler esses 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.

3.1.1.
Gerenciamento de Buffer

O gerenciamento de *buffer* realizado pelo processo provedor é semelhante ao apresentado na seção 2.4.1. 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.

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 de a atualização ser feita em páginas de tamanho fixo. Além do mais, 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. Outra diferença em relação à política FIFO é que a atualização do anel é feita independente de requisições, não sendo orientada a demanda, já que a informação a ser requisitada é prevista e copiada para o *buffer* antes da requisição.

3.1.2. Tratamento do Banco de dados

O fato de processos BLAST lerem o conteúdo do arquivo de seqüências em uma ordem diferente da requisitada faz com que tenham uma visão do arquivo diferente da verdadeira. Este fato gera problemas caso o BioProvider controle 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 podem 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.

O erro descrito pode ser visto na figura 6. O exemplo mostra, primeiramente, os arquivos originais de um banco hipotético de 4 seqüências, e em seguida mostra os arquivos enxergados por um processo que começou a ler o arquivo de seqüências a partir do início da terceira seqüência. Para simplificar, considera-se que todas as seqüências possuem o mesmo tamanho. No banco enxergado pelo processo, o primeiro ponteiro do arquivo de índices, que deveria apontar para a primeira seqüência, aponta na verdade para a terceira seqüência. Como conseqüência, a terceira seqüência será associada à anotação da primeira seqüência. O ponteiro que deveria apontar para a segunda seqüência apontará para a quarta, e assim por diante, fazendo com que as seqüências sejam associadas a anotações erradas. Se as seqüências não possuírem todas o mesmo tamanho, a associação será ainda mais errada, pois alguns ponteiros do arquivo de índices poderão apontar para posições no meio de seqüências, e não no início.

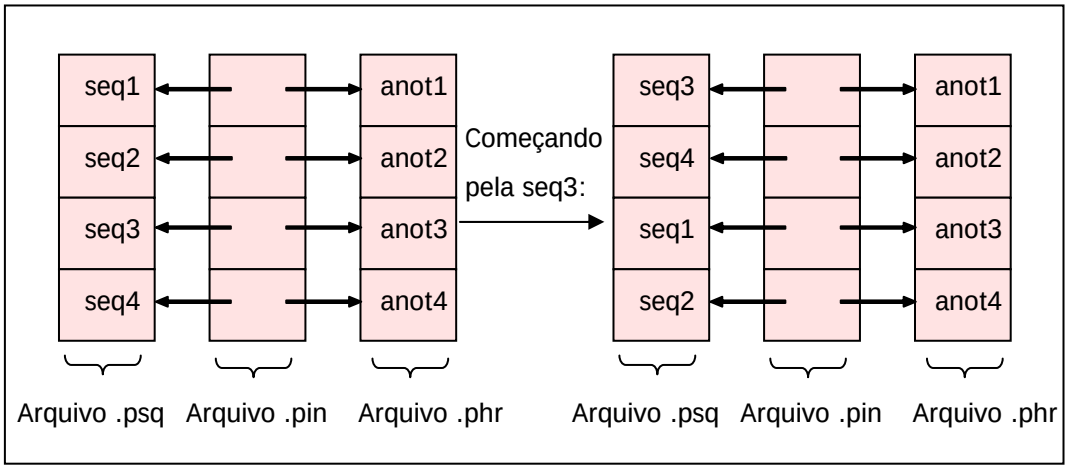

Figura 2: Erro dos ponteiros do arquivo de índices.

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 dos formatos não apenas do arquivo de seqüências, mas também do arquivo de índices, para poder modificá-lo em tempo de execução. O formato do arquivo de índices é complexo, varia entre diferentes implementações e versões do BLAST e pode ser difícil descobrir seu formato real já que as mudanças de uma versão do BLAST para outra nem sempre são informadas pelos desenvolvedores. O programa servidor de banco de dados teria que ser modificado para funcionar com cada implementação do BLAST que lesse arquivos de índices em um formato diferente.

A solução usada no BioProvider inclui a utilização da ferramenta do BLAST de geração do banco de dados a partir do mesmo em formato FASTA. Com esta ferramenta são criados, em uma etapa de pré-processamento do banco, os arquivos de índices e anotações correspondentes a diferentes visões do arquivo de seqüências que os processos BLAST poderão ter, dependendo da ordem das seqüências que estes lerão. Logo, se um processo começou a ler o arquivo de seqüências a partir da seqüência 10, ele receberá também os conteúdos dos arquivos de índices e anotações gerados pelo arquivo FASTA no qual as seqüências estavam ordenadas começando pela seqüência 10. Para criar as

diferentes instâncias do banco, basta modificar a ordem das seqüências no arquivo FASTA e usar a ferramenta fornecida junto ao BLAST para gerar o banco a partir do arquivo FASTA. As diferentes instâncias do arquivo de seqüências podem ser descartadas, exceto a gerada com as seqüências na ordem original.

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 7. Supondo-se a divisão das seqüências em 4 blocos, os processos BLAST poderão ler as seqüências do arquivo de seqüências 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 uma dessas 4 ordens são criados arquivos de índices e anotações correspondentes que deverão ser fornecidas aos processos BLAST.

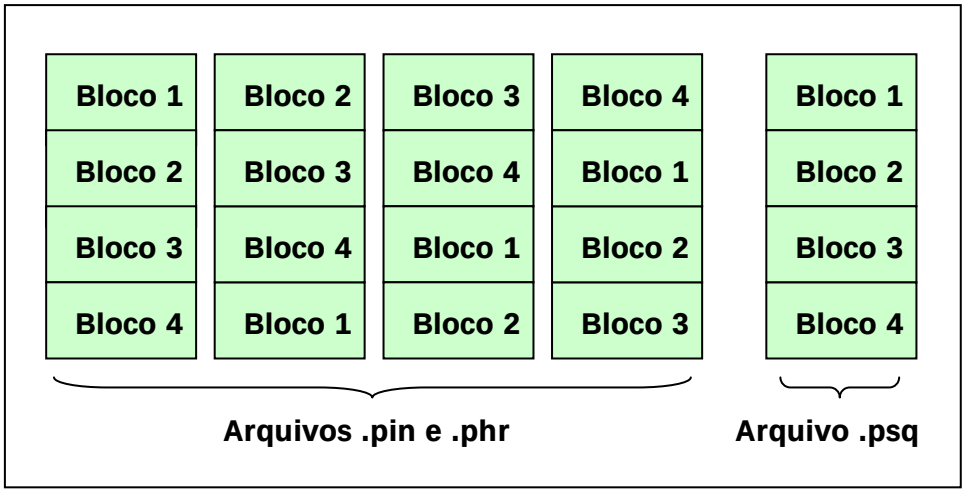

Figura 3: Pré-formatação do banco de dados

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 para identificar as posições de início das seqüências (e, conseqüentemente, dos blocos). Como foi observado no capítulo 2, os arquivos de seqüências de ambos os formatos de banco estudados são muito semelhantes, sendo que o caractere separador de seqüências possui o mesmo código (zero). Para encontrar as posições de início das seqüências, o BioProvider precisa identificar no arquivo apenas os caracteres separadores de seqüências. Logo, o uso desta estratégia facilita a criação de uma ferramenta mais genérica, que funcione com várias versões diferentes do BLAST.

3.1.3. Melhorias no Desempenho

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, sem que o espaço ocupado prejudique demais o funcionamento dos processos, o próprio sistema operacional deverá acessá-lo no disco apenas uma vez e mantê-lo em memória enquanto houver processos acessando. Esta é a situação ótima, na qual os processos terão melhor desempenho, não sendo necessário políticas alternativas de gerenciamento de *buffer*. Por outro lado, se o banco de dados for grande e não puder ser mantido inteiramente na memória, os processos deverão sofrer grande perda de desempenho, pois partes do banco de dados serão copiadas novamente para a memória quando diferentes processos acessarem o mesmo banco. Essa queda de desempenho só não ocorre quando todos os processos são iniciados em tempos bem próximos e lêem sempre o banco em posições próximas, de modo que partes do banco sejam retiradas da memória apenas quando todos os processos a tiverem lido.

De modo geral, o BioProvider diminui muito o número de cópias do disco para a memória quando há vários processos em execução concorrente lendo do mesmo banco e este não pode ser mantido inteiramente na memória. 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. É desejável, também, que a todo momento haja algum início de bloco no anel de modo que novos processos não precisem esperar o aparecimento de um início de bloco para começar a ler do anel. 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, novos processos BLAST podem demorar para começar a ler do banco de dados. É vantajoso também que os processos comecem a ler do anel em posições mais próximas do seu início, pois poderão aproveitar melhor as informações já disponíveis no anel. Ambos os problemas podem ser diminuídos aumentando-se o número de blocos nos quais o arquivo de seqüências é dividido. 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. O fato de haver um número grande de instâncias dos arquivos de índices e anotações dificulta também o gerenciamento de *buffer* realizado pelo sistema operacional sobre estes arquivos, tornando mais difícil o aproveitamento de dados dos mesmos já em memória, pois diferentes processos poderão estar lendo de diferentes arquivos.

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

- **Etapa de execução:** se a prioridade for sempre dada a processos que estão na etapa de leitura 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 estratégias implementadas no BioProvider incluem a possibilidade de priorizar pedidos de processos que não estão lendo do anel ou alternar esta prioridade entre pedidos de leitura do anel e pedidos de dados fora do anel.
- **Posição do arquivo:** 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.
- **Possibilidade de *starvation*:** É importante garantir que os processos BLAST em execução não fiquem bloqueados eternamente à espera da resposta ao seu pedido de leitura, sofrendo *starvation* (fome). Entretanto, a partir do momento em que um processo BLAST começa a ser atendido, este pode bloquear a atualização do anel durante um tempo considerável, 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. O processo só começará a ler do anel após ler o arquivo de índices, e o bloco do anel pelo qual sua leitura se iniciará não poderá ser removido da memória até que o processo o leia. Logo, se novos processos iniciarem sempre em períodos muito curtos, a atualização do anel ficará sempre parada. A solução implementada para tratar este problema é, de tempos em tempos, deixar de atender os processos que estão realizando o primeiro pedido de leitura. Deste modo, objetiva-se garantir que sejam realizadas

atualizações no anel antes deste ser bloqueado novamente durante o período no qual novos processos serão atendidos.

Quando a possibilidade de *starvation* é tratada, deve haver cuidado na definição do período durante o qual o atendimento de novos processos é proibido. Se o tempo for muito longo, poderá haver grandes atrasos para que novos processos comecem a ler do banco. Por outro lado, se o tempo for muito curto, este pode não ser o suficiente para garantir que o anel seja sempre atualizado, não resolvendo o problema de *starvation*.

Finalmente, é importante destacar a influência das seqüências de entrada no desempenho dos processos BLAST. 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 naturalmente 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 para o problema é 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.

3.2. Implementação

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 (Red Hat b,

2006) e no Linux Red Hat 8 (Red Hat a, 2006). 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.

O *driver* foi implementado como um módulo do Linux, podendo ser carregado dinamicamente no núcleo. Como o *driver* é usado para substituir um arquivo comum, este foi implementado como um *driver* de dispositivo de caractere. O *driver* é acessado por meio dos arquivos especiais associados ao mesmo, reimplementando as funções que realizarão operações nestes arquivos. As funções reimplementadas são *open*, *close*, *read*, *write*, *llseek*, etc. Estas funções são executadas na região de memória do núcleo (*kernel space*), e não na memória dos processos que as executam (*user space*). Logo, possuem privilégios do núcleo e, ao mesmo tempo, algumas limitações, sendo necessário utilizar funções específicas do núcleo para diversas operações. Os dados acessados pelas funções implementadas residem também na região de memória do núcleo, sendo compartilhados entre os processos que as executam. Logo, foram utilizados semáforos para controlar a concorrência no acesso a alguns dados, como as estruturas que armazenam informações sobre os processos BLAST em execução. Maiores detalhes sobre o funcionamento de *drivers* de dispositivos e a maneira de implementá-los são apresentados no apêndice C deste documento.

Os arquivos do banco de dados do BLAST devem ser substituídos por arquivos especiais associados ao *driver*. Ao executar funções de abertura e leitura destes arquivos, o BLAST chama na realidade as funções implementadas pelo *driver*, que realizam a comunicação com o processo provedor do banco de dados. Para diferenciar os tipos de arquivos sendo acessados, o *driver* utiliza o número menor associado ao arquivo especial no momento de sua criação. Arquivos especiais de seqüências são criados com número menor 1, os de índices possuem número menor 2 e os de anotações, 3.

O processo provedor foi implementado para atender os processos BLAST, realizar o escalonamento dos processos e gerenciar o *buffer* em memória correspondente ao anel. O provedor se comunica com os processos BLAST por meio das funções do *driver*. Para identificar a etapa de execução de um processo

BLAST, são usadas informações sobre as requisições do banco de dados feitas pelo mesmo.

Um módulo do linux só pode ser removido do núcleo quando nenhum processo o estiver acessando. Quando um processo abre um arquivo especial associado a um *driver*, seu contador de acessos é incrementado, sendo decrementado quando o processo liberar o arquivo. Entretanto, o contador não é decrementado quando o processo é finalizado à força, com um sinal de interrupção do Linux. Logo, para finalizar a execução do processo provedor, foi implementado um programa matador. Este executa a função de escrita do *driver* para descadastrar o processo provedor, acordá-lo e informá-lo que este deve ser finalizado. Se houver processos BLAST acessando o *driver* no momento, estes também são acordados e, como não há provedor cadastrado, recebem respostas com conteúdo vazio. Isso fará com que os processos finalizem a execução apresentando mensagem de erro, pois o BLAST identifica o erro no conteúdo esperado, tendo verificado no início da execução os tamanhos dos arquivos.

O funcionamento do BioProvider é resumido por meio de exemplos das execuções dos processos nas situações a seguir.

- **O processo provedor é iniciado:** Ao iniciar sua execução, o processo provedor lê informações no arquivo de configuração como a posição de cada arquivo do banco, o número de seqüências por bloco, o tamanho do anel que deverá ser criado e a posição do arquivo especial que será usado para a comunicação com o *driver*. Os arquivos do banco são abertos e, em seguida, o processo cria o anel em memória e copia para este o início do arquivo de seqüências, realizando o *prefetching*. O processo provedor abre então o arquivo especial e, executando a função de escrita do *driver*, cadastra-se como provedor (sendo identificado pelo IP) e envia para o *driver* informações sobre o banco de dados. Finalmente, o processo executa a função de leitura do *driver* e é bloqueado à espera de requisições de leitura dos processos BLAST.
- **Um processo BLAST executa a função de leitura do *driver:*** Ao ler de um arquivo especial, o processo BLAST executa a função de leitura do *driver*. O pedido de leitura é, então, guardado, o processo provedor é

acordado para atender a nova requisição e o processo BLAST é bloqueado à espera de resposta.

- **O processo provedor recebe pedidos de leitura:** Ao ser acordado, o processo provedor verifica os pedidos recebidos de leitura do banco. O provedor seleciona o pedido que irá atender e envia ao processo BLAST a resposta, que é obtida diretamente dos arquivos do banco ou do anel em memória, caso o processo esteja realizando a etapa de leitura seqüencial do arquivo de seqüências. O processo BLAST é então acordado para receber a resposta e continuar sua execução. O processo provedor executa novamente a função de leitura do driver, solicitando os pedidos de leitura dos processos BLAST. Se não houver nenhum pedido no momento, o processo é bloqueado à espera de novos pedidos.
- **O processo matador é executado:** Para finalizar a execução do processo provedor, o processo matador deve ser executado. O processo lê o arquivo de configuração e executa a função de escrita do *driver*, enviando um pedido para descadastrar o processo provedor. Este é então acordado e finalizado.

Para implementar a estratégia que trata o *starvation*, é utilizado um parâmetro *t*, que corresponde ao tempo durante o qual novos processos podem ser atendidos. Este tempo começa a ser contado quando o primeiro novo processo é atendido. Após o intervalo de tempo *t*, serão atendidos apenas processos que já começaram a ler o banco. Os novos processos que realizarem o primeiro pedido de leitura deverão esperar, primeiramente, até que todos os processos sendo atendidos terminem a leitura inicial do arquivo de seqüências e comecem a ler do anel. Em seguida, os processos aguardam ainda por um período correspondente a 2*t*, durante o qual espera-se que o anel seja atualizado algumas vezes. Ao término desse período, é permitido novamente o atendimento de novos processos.

Foram implementadas também ferramentas para a automatização do pré-processamento do banco de dados e a criação de arquivos de configuração. Alguns parâmetros podem ser especificados no arquivo de configuração, como as posições dos arquivos do banco de dados e dos arquivos especiais, o tamanho do anel na memória, o número de seqüências por bloco e a estratégia utilizada pelo processo provedor no atendimento dos pedidos de leitura.

3.3.
Considerações Finais

Este capítulo descreveu as estratégias usadas para o desenvolvimento do BioProvider e um resumo de como foi realizada a implementação. Primeiramente, foi apresentada a arquitetura do BioProvider. Em seguida, foi explicado, na seção 3.1.1, o modo como é realizado o gerenciamento de *buffer*, mostrando-se também as diferenças em relação a políticas existentes. Algumas dificuldades surgiram devido aos formatos dos bancos de dados do BLAST. A seção 3.1.2 descreveu os problemas encontrados e a solução criada de modo a possibilitar o funcionamento do BioProvider com as versões estudadas do BLAST. Na seção 3.1.3 foram mostrados alguns dos fatores que influenciam no desempenho dos processos BLAST usando o BioProvider e algumas soluções criadas para melhorar o desempenho. Finalmente, a seção 3.2 descreveu o modo como o BioProvider foi implementado.

Uma pequena modificação no código do NCBI BLAST é necessária para que este possa usar o BioProvider. No arquivo *readdb.c*, a função de sistema *stat* é usada para identificar se o arquivo do banco possui tamanho maior que zero, pois se o BLAST verifica o tamanho zero este finaliza sua execução apresentando uma mensagem de erro. Como arquivos especiais não possuem tamanho, a função *stat* retorna valor zero quando chamada. O código da comparação deve então ser modificado para permitir que o BLAST aceite ler arquivos de tamanho zero. Já a versão 1.4 do WU-BLAST não precisa ser modificada para funcionar com o BioProvider. O código do BioProvider, assim como explicações de uso, podem ser obtidos no site www.inf.puc-rio.br/~blast.