4
Avaliação paralela do BLAST

Se considerarmos um número alto de seqüências de consulta, o tempo do BLAST pode ser bastante demorado. Para amenizar o longo tempo de processamento, os desenvolvedores da ferramenta implementaram o uso de *threads*, a partir da versão 1.4 . Em computadores com mais de uma unidade de processamento ou processadores *multi-threading*, o BLAST consegue obter comparações mais rápidas, obtendo concorrência na CPU ou serviços paralelos em CPUs diferentes. Contudo o ganho obtido pela estratégia de *threads* é dependente do número de CPUs em uma única máquina, algo limitado e geralmente de custo bem elevado. Além disso, segundo estudos feitos em (24, 31) o maior custo no processamento do BLAST não está no uso da CPU, mas sim no tempo de troca de dados entre as memórias principal e secundária.

Ao considerarmos arquiteturas que permitem *multi-threading* ou multi-processamento (SMP - Symmetric Multi-Processing), o compartilhamento de memória primária pode reduzir o tempo de execução, pois processos diferentes poderão acessar distintas regiões da base de dados. O trabalho (6) discute o gargalo no acesso de vários processadores acessando a mesma memória compartilhada.

Algo que pode facilitar o entendimento do uso de multiprocessadores na execução BLAST é observarmos que o seu funcionamento se trata de uma aplicação vergonhosamente paralela(36). Dizemos isso pois, ao executar várias seqüências de consulta, não existe nenhuma necessidade de comunicação durante o processamento para que o resultado seja obtido. A partir desta observação, vários trabalhos passaram a investigar estratégias de execução paralela da ferramenta BLAST.

O objetivo deste Capítulo é fazer um estudo das diversas possibilidades de execução da ferramenta BLAST em ambientes paralelos. Para melhor

compreensão dos trabalhos relacionados, este Capítulo será dividido em dois grupos distintos; trabalhos que utilizam a estratégia Replicação Total e trabalhos que utilizam a estratégia Fragmentada. Na estratégia com Replicação Total cada estação de trabalho mantém toda a base de dados e recebe somente algumas seqüências de consulta. Na estratégia Fragmentada, a base de dados é dividida entre as diversas estações, e uma seqüência é executada várias vezes, cada execução com uma parte da base de dados até que todo o banco tenha sido usado para a comparação.

4.1
Estratégia com Replicação Total

Se tratando de estratégias em que a base de dados se encontra replicada, uma solução considerada de fácil implementação é, a partir de uma lista de seqüências de consulta, dividi-la para que cada estação de trabalho processe parte desta lista (9). Contudo este tipo de abordagem pode gerar desbalanceamento de carga se não for feito um tratamento adequado da forma de dividir a lista de seqüências de consulta.

Foi pensando nisso que o trabalho (10) utiliza a mesma estratégia, no entanto, ordenando todas as seqüências de consulta pelo tamanho e usando o algoritmo circular (*round-robin*) ao enviar uma seqüência para cada máquina. Esta estratégia favorece o balanceamento considerando que cada máquina receberá a mesma quantidade de dados para processamento. Contudo outros fatores de desbalanceamento como o nível de similaridade entre as seqüências e a base de dados, a capacidade de processamento de cada máquina, e também o número de processamentos externos não são tratados por esta estratégia. Neste sentido, além do Capítulo 3, o trabalho (9) demonstra, através de vários experimentos, a influência destas características no desbalanceamento de carga.

No trabalho (35) a estratégia com bancos de dados replicada é utilizada e impõe uma forma mais dinâmica para tratar o desbalanceamento de carga. Como alternativa, o trabalho propõe o envio de seqüências sob demanda, variando a quantidade de seqüências submetidas. A idéia é enviar pequenos grupos de seqüências até que metade de todas as seqüências seja alcançado, e a partir daí um número maior é enviado. Embora a estratégia seja discutida, não são mostrados os ganhos obtidos com variações do tamanho das submissões e ainda o desbalanceamento que possa ter permanecido.

Também tratando de forma dinâmica o balanceamento de carga, o trabalho (36) submete às estações de trabalho grupos de seqüências de consulta. Isto é feito até que 90% do total de seqüências sejam processadas. Os 10% restantes são enviados em pequenos grupos, à medida que as estações de trabalho se tornam ociosas.

Já em (37, 16) podemos ver uma abordagem de balanceamento diferente. Ao invés de trabalhar com requisições sob demanda, as informações sobre o número de seqüências processadas e a fila de processos do processador são trocadas entre as estações de trabalho e de gerência. Os trabalhos mostram poucas análises de balanceamento, além de não comparar com outras estratégias existentes. Ambos disponibilizam uma interface web para execução paralela, facilitando a interação com o usuário.

Em (5) é sugerida uma estratégia para execução da ferramenta BLAST em ambientes de grades computacionais (*grids*) com distintas configurações de *hardware*. O sistema recebe as seqüências de consulta e as divide em vários arquivos (definido pelo usuário) de seqüência. Cada um destes arquivos passa ser a unidade de processamento. O trabalho sugere que os arquivos de seqüências gerados não devem ser muito pequenos para evitar tráfego na rede. Novamente não é apresentada nenhuma análise do balanceamento de carga, mesmo assim é possível prever que a mesma quantidade de seqüências submetidas a computadores com taxa de processamento diferentes deve gerar desbalanceamento. Preocupado com a tolerância à falhas o artigo mostra controle caso alguma máquina pare de funcionar, mantendo o sistema em andamento. O sistema também consegue se recuperar caso a estação de gerência por algum motivo não funcione mais, criando para isso um arquivo com registros de todas as seqüências já processadas. Então quando a máquina gerente retoma os trabalhos após alguma parada, o início do processamento será feito a partir do ponto que havia parado.

Outra alternativa para gerenciar as execuções BLAST submetidas entre as estações foi utilizado no trabalho (28). Este último utiliza um programa externo para coordenar as submissões, permitindo que serviços possam ser interrompidos para que outros com maior prioridade entrem em execução. Embora o tratamento para o balanceamento de carga seja transferido para um software de escalonamento geral, este não fará ajustes finos e específicos ao comportamento da ferramenta BLAST. Além disso, principalmente para serviços de larga-escala, existe pouca interação entre aplicações BLAST e o

escalonador, e este tem dificuldades para alocar os recursos da melhor forma (30).

4.2 Estratégia Fragmentada

Embora a paralelização da ferramenta BLAST em bases replicadas tenha possibilitado reduzir o tempo de processamento, com o crescimento da base de dados a comparação de uma seqüência demanda elevado acesso ao disco. Pois, para cada seqüência de consulta o banco de dados é carregado inteiramente para a memória primária.

Foi neste contexto que trabalhos passaram a sugerir que uma seqüência fosse executada várias vezes, mas cada uma destas execuções com uma parte da base de dados. Então as estratégias passaram a utilizar os conceitos de bancos de dados fragmentados (26), dividindo a base por distintas estações de trabalho.

Diferente de quando a base de dados é replicada, na estratégia fragmentada o principal ajuste de balanceamento de carga se refere ao tamanho dos fragmentos submetidos às estações de trabalho e não à quantidade de seqüências de consulta.

Nesta seção iremos abordar estratégias com bases fragmentadas, divididos em dois grupos: abordagens intrusivas e não intrusivas.

Soluções intrusivas ocorrem quando o código da ferramenta BLAST é alterado. Se isso é necessário para que a estratégia paralela possa ser executada, existe um alto acoplamento entre o programa BLAST e a estratégia de execução paralela. Neste caso a estratégia de balanceamento estará dependente da ferramenta BLAST e de uma específica implementação desta, dificultando e em alguns casos impossibilitando, que futuras alterações de versão, ou até mesmo diferentes ferramentas possam ser utilizadas. Em (23) comenta-se que, dada a grande necessidade na comparação de seqüências biológicas, a ferramenta BLAST já não é tão eficiente para abordar as necessidades vigentes. Além de nós mesmos termos verificado esta falta de eficiência, o que nos incentivou ao estudo de estratégias paralelas, é possível que novas implementações seriais que substituam a ferramenta BLAST estejam por vir.

Desta forma, consideramos intrusivos todo trabalho que de alguma forma necessita que a ferramenta BLAST seja alterada. Mesmo que esta alteração não modifique as idéias do algoritmo do BLAST mas sim, algumas parte do código ou mesmo do ambiente do sistema operacional.

4.2.1 Estratégia Fragmentada Não Intrusiva

No trabalho (28), após a implementação da estratégia com bases replicadas, comenta-se que a utilização de bases fragmentadas pode trazer ganhos significativos, reduzindo o espaço de disco em cada máquina, e diminuindo o tempo de processamento total. Contudo o mesmo não implementa a estratégia fragmentada.

Já em (10) a estratégia fragmentada é implementada, onde várias formas de alocação dos fragmentos foram testadas, como por exemplo, fragmentos com seqüências de mesmo comprimento, e fragmentos com mesmo número total de caracteres. Além disso, houve grande número de análises e comparações a fim de investigar a melhor forma de alocar os fragmentos entre as estações de trabalho. Em seus testes se pode concluir que em bancos de dados pequenos a estratégia replicada possui melhor desempenho, devido à alocação da toda a base de dados na memória. Já para bancos de dados grandes, a estratégia fragmentada foi melhor, devido ao paralelismo de E/S obtido. O trabalho (17) também utiliza fragmentos com similar quantidade e tamanho de seqüências no intuito de evitar o desbalanceamento de carga.

Em (21) a estratégia fragmentada é utilizada para demonstrar os ganhos obtidos em relação a uma execução serial da ferramenta BLAST. O trabalho não apresenta a estratégia de balanceamento de carga, mas evidencia os custos adicionais na estratégia fragmentada: escalonar seqüências de consulta, tempo de montagem final dos relatórios gerados e variações de desempenho de carga para cada máquina.

O trabalho (32) também utiliza a estratégia de fragmentação da base de dados, contudo, o mesmo adota duas formas para balancear o processamento. Na primeira, a fim de evitar o desbalanceamento causado pela diferença de similaridade entre as seqüências de consulta e os fragmentos da base de dados, existe um pré-processamento que mistura a ordem das seqüências no banco, evitando o agrupamento de mesma família. Na segunda forma de balancea-

mento, preocupado com a heterogeneidade das máquinas, diferentes tamanhos de fragmentos são alocados seguindo o potencial de processamento de cada estação.

A ferramenta TurboBLAST(2) disponibiliza uma estratégia a qual divide todo o processamento em pequenas unidades de trabalho, gerenciadas pelo sistema TurboHub(2). TurboBLAST permite que as estações de trabalho solicitem às estações de gerência fragmentos da base de dados para que possam ser processados. Considerando que a tarefa de processamento já esteja disponível na estação de trabalho, esta poderá sub-dividir a tarefa recebida com o intuito de aumentar o poder de execução. Cabe observar que TurboBLAST é uma ferramenta comercializada, não disponível gratuitamente.

4.2.2 Estratégia Fragmentada Intrusiva

Em (29) é apresentada uma estratégia em que o código fonte do BLAST foi alterado para tratar fragmentos menores que a memória principal. O trabalho sugere reduzir o custo da montagem dos resultados, dividindo a base em poucos fragmentos. Para isso, o usuário tem a opção de definir o tamanho ou adotar o padrão de 700 MB por fragmento. Este último valor foi dado considerando-se a configuração das máquinas utilizadas para testes. A estratégia não abordou nenhuma solução para balanceamento de carga.

No trabalho (17) a estratégia fragmentada é implementada utilizando um agrupamento de arquitetura SMP. O trabalho procura obter ganhos no paralelismo existente na própria ferramenta BLAST quanto executado em computadores com varias CPUs. Para isso fragmentos da base de dados são submetidos para as estações de trabalho de forma pré-definida pela linha de comando. Na estação existe uma *thread* que disponibiliza outros pedaços menores do fragmento para execução nos processadores, seguindo uma distribuição sob demanda. O acesso remoto aos arquivos de seqüência, e a base de dados é feita via compartilhamento de disco. Com isso, além do uso de máquinas com multiprocessadores exigir alto custo, a estratégia proposta pode ser prejudicada pelo gargalo causado por várias requisições em ambiente de memória compartilhada.

A estratégia de base de dados fragmentada intrusiva também é utilizada no trabalho (30), sendo que a montagem para as comparações das seqüências

com cada fragmento é feita de forma diferente das até aqui vistas. Este procedimento é feito mantendo-se uma estrutura em memória principal que armazena as informações relevantes para cada relatório de saída. Uma vez que as informações de uma única seqüência estejam completas, são então agrupadas formando um único arquivo. O programa também permite que grandes seqüências de consultas sejam divididas e comparadas separadamente, para futura montagem dos resultados. Embora o trabalho demonstre suas sugestões, o excessivo armazenamento dos relatórios de saída em memória primária pode dificultar o uso da mesma durante o processamento BLAST. Esta ferramenta também não é disponibilizada gratuitamente.

Já bem divulgada na comunidade científica a ferramenta mpiBLAST(11) propõe a estratégia fragmentada, permitindo que um grupo de computadores estejam preparados para vários fragmentos da base de dados. Neste caso, as estações de trabalho recebem os fragmentos da base de dados, comparam com as seqüências de consulta previamente recebidas, e retorna os resultados. A proposta é obter o balanceamento de carga na distribuição dos fragmentos da base. Esta distribuição é feita utilizando um algoritmo guloso a medida que as estações de trabalho vão retornando os resultados à máquina gerente. Consideramos esta aplicação intrusiva, pois para que a mesma seja instalada é necessário que o código fonte do BLAST seja alterado. Isso ocorre para possibilitar os ajustes referentes às estatísticas de similaridade em bases fragmentadas.

Contudo, a estratégia escolhida para obter balanceamento possui algumas desvantagens. Uma delas é a alta concorrência quanto é grande o número de processos solicitando os fragmentos compartilhados (20). Além disso, quanto maior o número de fragmentos para obter melhor balanceamento, maior será o custo para montar o resultado final. Observando também nossos testes (apresentados no Capítulo 6), uma vez que o fragmento seja menor que a memória primária, dividi-lo ainda mais não trará benefício, pois aumentará o custo em montar os resultados.

A partir dos problemas encontrados no mpiBLAST, o pioBLAST(20) propõe contribuições buscando principalmente minimizar o tempo de *overhead* da execução paralela, da seguinte forma: diminuir o tempo de execução no pré-processamento da estratégia fragmentada e acelerar o processo na montagem do resultado final. Para primeira contribuição foram utilizados os arquivos de índices do BLAST para selecionar somente um fragmento da base de dados, isto foi feito a partir do uso das bibliotecas de paralelização de

E/S, MPI-IO(34). Para utilizar estas bibliotecas, foi alterado o código fonte da ferramenta BLAST, evitando que os arquivos da base de dados fossem acessados pelo controle do sistema operacional.

A possibilidade de obter pedaços da base de dados sem que a base seja dividida fisicamente se refere ao conceito de fragmentação virtual(22). Esta evita a necessidade do pré-processamento da base de dados que a fragmentação física utiliza.

A segunda contribuição do trabalho pioBLAST é aperfeiçoar a paralelização na geração do resultado final, permitindo que, coordenado pelo mestre, cada máquina acesse paralelamente o mesmo arquivo de saída, assim todos concorrentemente estarão criando a saída do BLAST.

Duas análises são importantes quanto às contribuições deste trabalho. A primeira é que para obter a fragmentação virtual da base de dados, o sistema depende do formato imposto pelo arquivo de índices, o que pode alterar dependendo da versão do BLAST, assunto este também verificado em (24). Além disso, o uso abusivo de paralelização de E/S tem benefícios limitados (39), pois com o aumento do número de estações participantes o alto tráfego de submissões deteriora o desempenho, podendo aumentar o custo.

No trabalho (39) o código fonte da ferramenta BLAST é alterado para que os acessos de E/S sejam feitos utilizando bibliotecas PVFS (Parallel Virtual File System) e CEFT-PVFS (Cost-Effective Fault-Tolerant PVFS). Para que a alteração seja testada em ambiente paralelo foi utilizado a ferramenta mpiBLAST. O trabalho mostra que o ganho obtido na paralelização de E/S chega até o nível em que o custo de acesso concorrente das estações de trabalho e o próprio uso de CPU se tornem preponderantes. Além disso, o mesmo mostra que o uso de PVFS sobressai ao uso de CEFT-PVFS, devido à diminuição de nodos como fonte de dados e maior replicação dos dados. Contudo, o CEFT-PVFS mantém o sistema robusto caso alguma máquina falhe ou esteja em excessivo processamento de E/S.

Outro trabalho que também utiliza estratégia fragmentada de forma intrusiva é o ScalaBLAST(25). Este tem como escopo: distribuir a base de dados entre a memória primária disponível e explorar concorrência em ambientes paralelos afim de evitar acessos ao disco. As máquinas de trabalho são divididas em grupos com uma estação coordenadora que recebe uma base

de dados replicada. Utilizando uma estratégia estática de balanceamento de carga, cada grupo de máquinas recebe distintas seqüências de consulta a serem processadas. Contudo, as máquinas pertencentes a um grupo terão replicadas as seqüências de consulta destinadas ao grupo. Cada máquina irá processar as seqüências recebidas com diferentes fragmentos da base de dados. Para que as máquinas pertencentes ao grupo acessem fragmentos distintos, o código fonte da ferramenta BLAST foi alterado no intuito de facilitar o acesso aos arquivos remotos, na estação coordenadora. Algo que não foi apresentado pelo trabalho é o balanceamento de carga entre os grupos de estações de trabalho.

4.3 Conclusão

Vimos neste Capítulo que existem várias abordagens para execução da ferramenta BLAST em paralelo. Em meio a estes, acreditamos que ainda sejam necessários estudos que analisem e implementem conceitos de bancos de dados distribuídos.

Dentre os trabalhos aqui apresentados, foi possível observar uma maior quantidade de estratégias intrusivas. Embora manter o código fonte do BLAST intacto seja difícil, percebemos que igual é a dificuldade em manter várias destas soluções associadas às novas atualizações da ferramenta BLAST. De tal forma, e como será visto nos próximos capítulos, novas propostas com um fraco acoplamento entre o BLAST e estratégias de execução paralela tornam-se mais manuteníveis.

Nos trabalhos citados que usam abordagem fragmentada, verificamos que grande maioria deles obtém balanceamento de carga a partir da alocação dos fragmentos da base. Em alguns casos, permitindo que alguns fragmentos sejam replicados, a fim de usar os recursos disponíveis.

Como boa parte dos trabalhos se aplicam ao uso de *cluster* de computadores, o que aumenta a ocorrência de erros durante a execução, pouco tem sido comentado sobre a robustez da execução. Possibilitando que o sistema se recupere caso alguns erros de hardware possam ocorrer.