5
Soluções Propostas

Nos capítulos anteriores discutimos as dificuldades e abordagens referentes à utilização da ferramenta BLAST em um agrupamento de computadores. No Capítulo 2, listamos alguns problemas mais pontuais quando utilizadas estratégias de replicação ou fragmentação da base de dados e no Capítulo 3 as possibilidades de desvio de tempo inerente ao processamento BLAST.

Neste Capítulo, exploramos as soluções propostas para alguns dos problemas citados, com o objetivo principal de averiguar as possibilidades de ganho na execução da ferramenta BLAST utilizando estratégias de banco de dados distribuídos e visando o paralelismo de acesso aos dados. Pelos motivos já discutidos, consideramos importante que isso seja obtido de forma não intrusiva.

Como soluções propostas, são sugeridas duas diferentes estratégias de execução BLAST em ambiente paralelo. A principal diferença entre elas se refere à alocação da carga de trabalho entre as várias máquinas para que se complete o processamento.

A primeira estratégia a ser discutida será descrita na Seção 5.1, Estratégia Sob Demanda. A segunda é a Estratégia Corretiva, definida na Seção 5.2. Embora essas estratégias possuam distintas formas de alocação de carga, ambas contêm a mesma disposição da base de dados. Essa alocação será explicada detalhadamente a seguir.

5.0.1

Disposição da base de dados

Ao analisarmos o processo de execução em ambientes paralelos da ferramenta BLAST utilizando bases de dados replicadas, é possível perceber que qualquer seqüência de consulta pode ser transferida para qualquer máquina arbitrária, e o processamento irá ocorrer normalmente. Isso facilita o controle no balanceamento de carga, favorecendo a reutilização dos recursos disponíveis. Por outro lado, demanda alto custo de acesso ao disco ao comparar uma seqüência com a base de dados completa, pois em muitos dos casos são maiores que a memória primária.

Observando a execução em ambientes paralelos fazendo uso da base de dados fragmentada, verificamos o ganho de tempo quando grandes bancos de seqüências são divididos entre diversas máquinas. Com essa estratégia, cada estação de trabalho não mais carrega todas as seqüências da base de dados, obtendo assim um paralelismo de E/S. Contudo, essa última estratégia restringe a flexibilidade na disposição de tarefas entre as diversas máquinas, dificultando o reuso de recursos que estejam ociosos.

Diante das verificações feitas, implementamos uma alocação da base de dados que junta as principais vantagens das estratégias comentadas, isto é, flexibilidade e paralelismo de E/S. Dessa forma, todo o banco de seqüências é distribuído de forma que esteja replicado entre as diversas máquinas de forma fragmentada. Ou seja, uma base de dados copiada em cada estação de trabalho, mas composta de pedaços com diferentes conteúdo e similar tamanho.

Denominamos esta alocação de *replicada com fragmentos primários*, ilustrada na Figura 5.2. O termo primário caracteriza um subgrupo de fragmentos alocados em cada estação, que serão prioritariamente utilizados. Cada estação contém distintos fragmentos primários e os fragmentos não primários são denominados secundários. Os termos aqui utilizados não se referem aos termos da estratégia de fragmentação primária ou derivada, citado em (26)). Assim, cada estação de trabalho é responsável (ao menos inicialmente) por somente uma parte da base de dados.

Uma importante característica dessa alocação do banco é que, embora uma máquina tenha disponível todos os fragmentos da base de dados (primários e secundários), no início do processamento somente serão utilizados os fragmentos primários. Na medida em que for necessário, a fim de manter o sistema balanceado, os fragmentos secundários poderão ser utilizados. Na

Figura 5.2 os fragmentos primários estão destacados em cada estação de trabalho.

Embora as implementações que serão apresentadas considerem essa estrutura da base de dados, a idéia aqui apresentada não obrigatoriamente exige que cada estação de trabalho tenha uma cópia inteira do banco de dados, pois possivelmente alguns fragmentos secundários não serão utilizados.

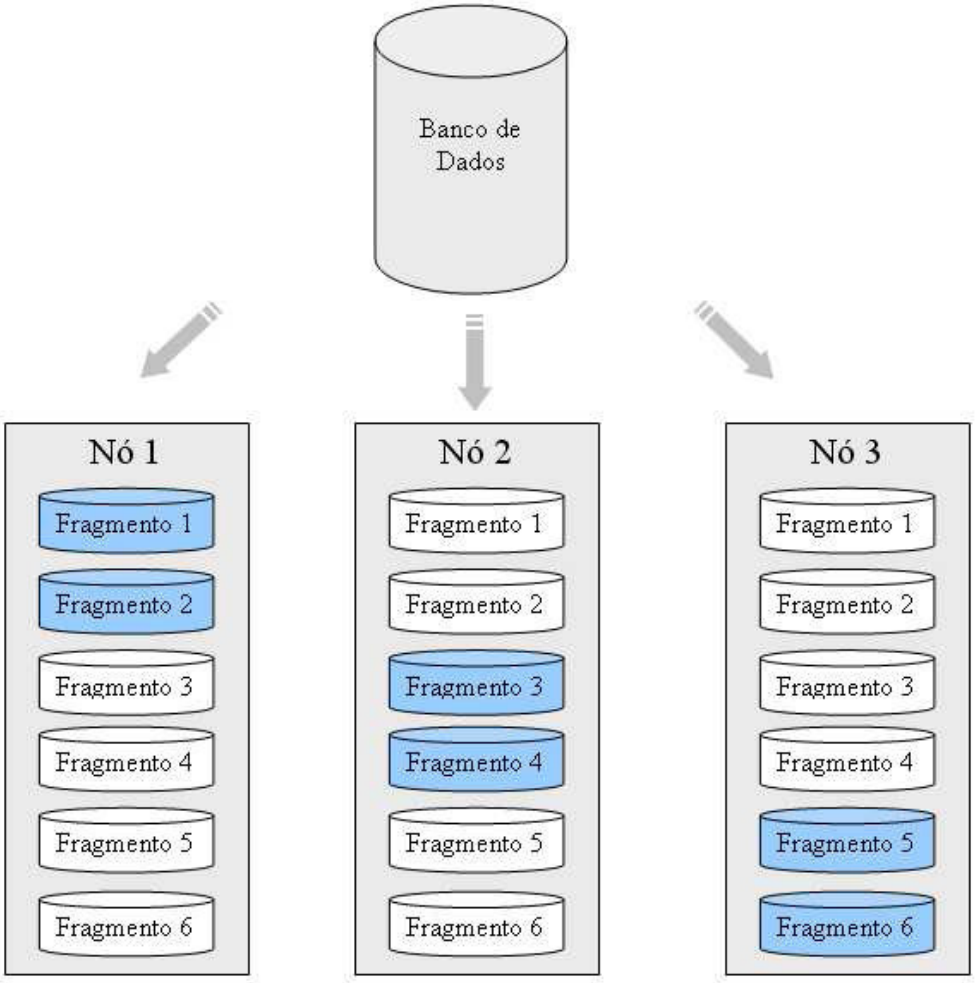

Figura 5.1: *Exemplo da alocação da base de dados com replicação total com fragmentos para três máquinas. Em cada máquina de trabalho o banco inteiro está fragmentado em distintas partições, e em destaque estão os fragmentos primários para cada estação.*

Com essa estratégia de fragmentação, será necessário incluir um novo pré-processamento na base de dados, no sentido em que todo o banco deve ser fragmentado. Somente após esta divisão poderemos formatar cada fragmento para ser então utilizado pela ferramenta BLAST. Neste sentido, desenvolve-

mos um programa que, a partir de uma base de seqüências não formatada, irá quebrar a mesma em fragmentos com similar quantidade de caracteres. Para cada fragmento executamos o programa *formatdb*, disponibilizado pelo BLAST, para que todos os fragmentos estejam formatados e disponíveis para comparação.

Uma questão relevante na fragmentação da base de dados é saber em quantos fragmentos a base deve ser dividida. Já que o intuito das estratégias propostas é evitar o acesso ao disco, é importante que os fragmentos sejam menores que a memória primária. Contudo, o número excessivo deles, uma vez que já estão menores que a memória, vai aumentar o *overhead* para montagem dos sub-relatórios de saída do BLAST.

Em ambientes de execução não exclusivos, calcular previamente o tamanho do fragmento para que seja menor que a memória primária pode não refletir ganhos efetivos. Contudo, mesmo que processos externos estejam concorrendo pelo uso da memória primária, com a utilização dos fragmentos o impacto referente a leitura de toda a base de dados será menor.

Embora nossa implementação tenha sido aplicada em ambientes de memória distribuída, necessitando que em toda estação de trabalho tenha espaço suficiente para armazenar a base de dados, é possível que os fragmentos possam ser acessados a partir de uma estação de memória compartilhada, possibilitando que somente os fragmentos necessários sejam copiados para as máquinas de processamento. Entretanto, esse tipo de acesso pode implicar em um gargalo, já que várias máquinas estarão concorrendo por cópias dos fragmentos em um número limitado de máquinas.

5.1 Estratégia Sob Demanda

Considerando que a base de dados esteja alocada de forma replicada com fragmentos primários, o objetivo da Estratégia Sob Demanda é enviar tarefas para serem processadas na medida em que uma estação de trabalho se encontre ociosa, buscando assim maior reutilização de recursos disponíveis.

Para entender o funcionamento, vamos definir uma expressão que represente uma tarefa enviada para qualquer estação de trabalho, da seguinte forma:

Expressão 1

$$T(x:y)(z:w)$$

Consideremos que: $\forall x, y, z \ e \ w; \ \ x \leq y \ e \ z \leq w$.

As variáveis x e y representam a faixa de seqüências de consulta, e as variáveis z e w a faixa dos fragmentos a serem utilizados na tarefa. Por exemplo, a tarefa T(1 : 5),(6 : 8) significa: execução das seqüências de 1 até 5, contra todos os fragmentos entre 6 a 8.

Tanto para seqüências como para fragmentos, existe um intervalo entre seqüências e fragmentos de uma dada tarefa. No exemplo anterior, o intervalo de seqüências de consulta é 5, e de fragmentos é 3. Esses intervalos serão representados por variáveis rS e rF, respectivamente. No intuito de facilitar nosso entendimento, para explicar essa estratégia consideraremos que rS e rF são valores constantes, o que não necessariamente ocorre na prática.

No início do processamento, cada estação de trabalho receberá uma tarefa em que o valor de x é igual a 1, e y é o número de seqüências enviadas por tarefa (rS). Após uma máquina executar uma tarefa recebida, o resultado será enviado à estação de gerência e a máquina estará ociosa aguardando outra tarefa. O processamento permanecerá enviando tarefas para as estações ociosas até que não tenha mais tarefas a serem executadas.

Ao observar essa estratégia, percebemos que uma tarefa é dividida em várias outras que serão executadas paralelamente. Considerando um processamento BLAST completo (sem subdivisões) representado por T(1 : n),(1 : m), com n o número de seqüências de consulta e m o número de fragmentos, podemos formular as suas divisões (ou subtarefas) a serem paralelizadas da seguinte forma:

Sejam: $\Delta n = \lfloor n/rS \rfloor, \Delta m = \lfloor m/rF \rfloor$

Consideremos sem perda de generalidade, que os restos das divisões n/rS e m/rf são iguais a zero.

Expressão 2

T(1 : n)(1 : m)

$\equiv$

$$\sum_{j=0}^{\Delta m-1} \sum_{i=0}^{\Delta n-1} T(i * rS + 1 : (i+1) * rS)(j * rF + 1 : (j+1) * rF)$$

Sabe-se que o processamento geral terminou quando o sistema verificar que a tarefa completa T(1 : n),(1 : m), ou a expressão equivalente, tiver sido executada. Para facilitar o entendimento, ilustramos na Figura 5.2 o fluxograma mostrando o funcionamento da estratégia sob demanda.

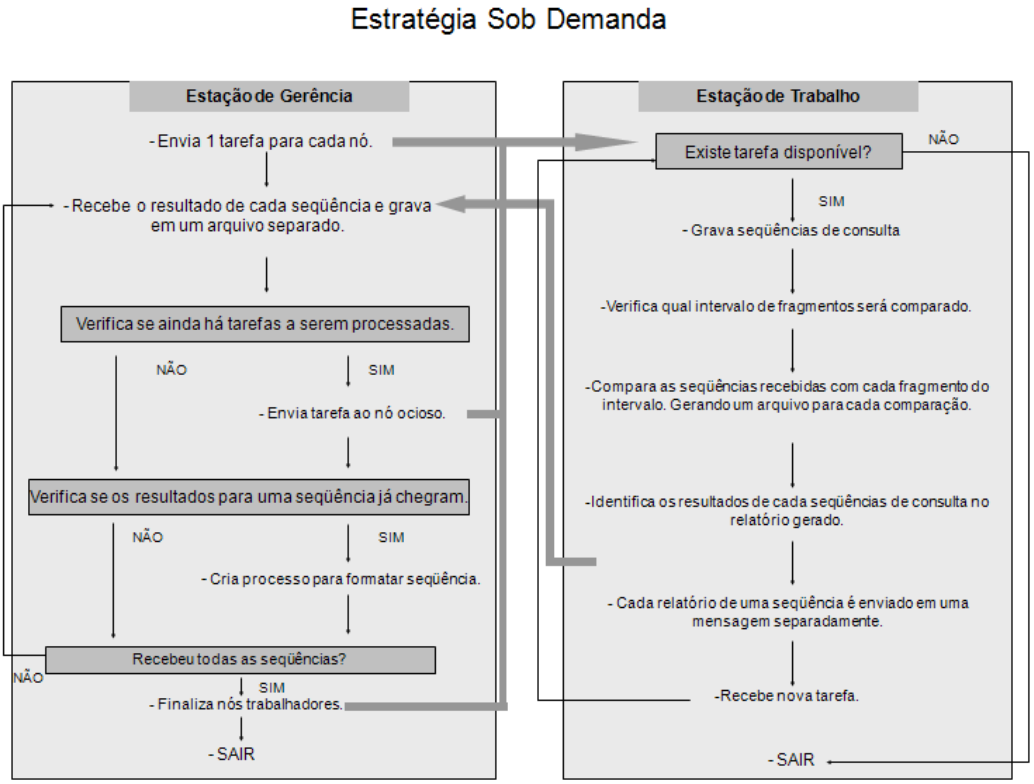

Figura 5.2: *Algoritmo de funcionamento da estratégia Sob Demanda.*

Em um primeiro momento, preferencialmente, cada máquina receberá tarefas a serem processadas com os respectivos fragmentos primários, até que todas as seqüências de consulta sejam processadas. A partir desse momento, se ociosa, a estação de trabalho recebe seqüências para outros fragmentos, a fim de manter o sistema balanceado, ajudando também a completar a execução do BLAST de forma mais rápida.

Ou seja, considerando-se que toda a tarefa:

T(1 : n)(z : w),

tenha sido executada pela primeira máquina ociosa, a máquina poderá receber tarefas com outros valores para z e w.

Durante todo processamento do sistema, sempre que uma estação de trabalho executar uma tarefa, o resultado será enviado a uma única estação de gerência. Esta verifica se a seqüência recebida já foi comparada com todos os fragmentos da base de dados. Em caso positivo, o sistema cria um novo processo filho com o objetivo de montar um único relatório de saída para uma seqüência de consulta. Para evitar que muitos processos sejam iniciados concorrentemente, existe um parâmetro que define o número máximo de processos em execução. O processo de montagem do resultado final basicamente mescla os *hits* de cada arquivo buscando otimizar o acesso ao disco. A partir do momento que todas as seqüências já estiverem montadas, os resultados são concatenados em um único arquivo de saída.

5.1.1 Balanceamento de Carga

A Estratégia Sob Demanda divide uma única tarefa em várias outras e uma tarefa é enviada por vez a cada estação de trabalho, e assim segue; sempre que esta estiver ociosa, poderá receber uma nova tarefa.

Esta flexibilidade em dividir uma grande tarefa permite que o sistema possa se manter balanceado, e obter maior reutilização dos recursos disponíveis. Quando uma tarefa é submetida, ela pode conter uma ou mais seqüências de consulta. Nesse caso, quanto menor for o número de seqüências em cada tarefa, maior será o número de mensagens enviadas durante todo o processamento, aumentando o uso da rede e o tempo de espera da estação de trabalho, contudo, facilita manter o sistema balanceado.

Se em um extremo podemos enviar uma seqüência dentro de cada tarefa, em outro podemos enviar todas as seqüências em uma única tarefa, dificultando o balanceamento de carga. Para evitar grande discrepância, é necessário um ajuste fino entre o número de seqüências submetidas. Neste ajuste espera-se obter ganho em dois pontos: custo de mensagens trafegando na rede e o intervalo de tempo para obter outra seqüência de processamento em função do poder de remanejamento entre as estações de trabalho.

Esta variação no tamanho da tarefa enviada se refere à granularidade do processamento. No Capítulo 2 comentamos que em um processamento paralelo para a ferramenta BLAST, a granularidade pode ser ajustada pela variação de envio das seqüências de consulta, pela utilização da base de dados ou por ambas as formas. Na nossa estratégia em questão, a granularidade é ajustada por ambas as formas, pois cada tarefa permite informar quais as seqüências de consulta e os fragmentos da base de dados que serão utilizados. As variáveis que definem a granularidade são rS e rF, respectivamente. Quanto maior forem seus valores, maior será a unidade de processamento.

A fim de ilustrar melhor este ajuste de granularidade, a Figura 5.3 apresenta o balanceamento de carga quando a variável rS possui valor igual a 2. Isto é, em cada tarefa duas seqüências são enviadas para processamento. A Figura 5.4 mostra o balanceamento para rS igual a 32, e as Figuras 5.5, 5.6 e 5.7 para rS igual a 64, 128 e 256, respectivamente. Para todos esses testes foram utilizadas 1.000 seqüências de consulta e a base de dados nt, com 24 fragmentos. Veja no Apêndice 1 o ambiente de teste utilizado. À medida que a granularidade (valor de rS) aumenta, o ambiente se torna mais desbalanceado.

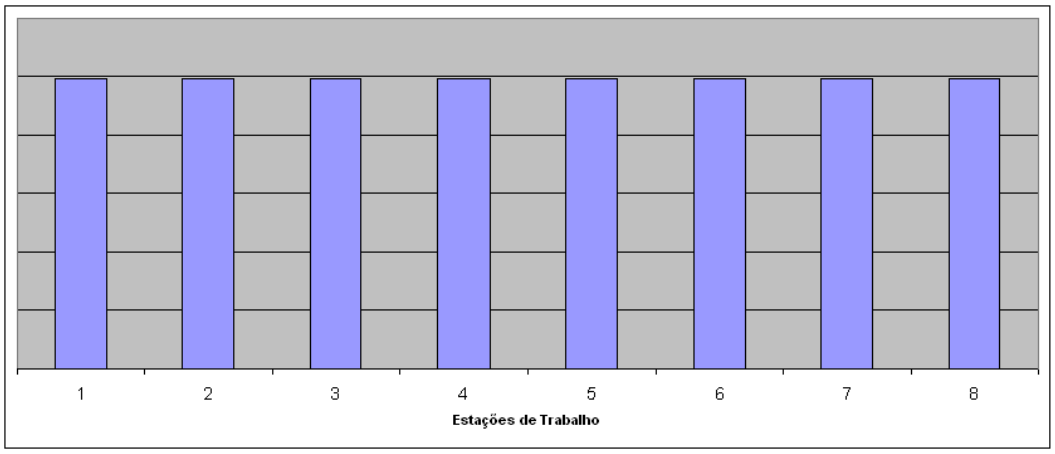

Figura 5.3: *Balanceamento de carga utilizando rS=2.*

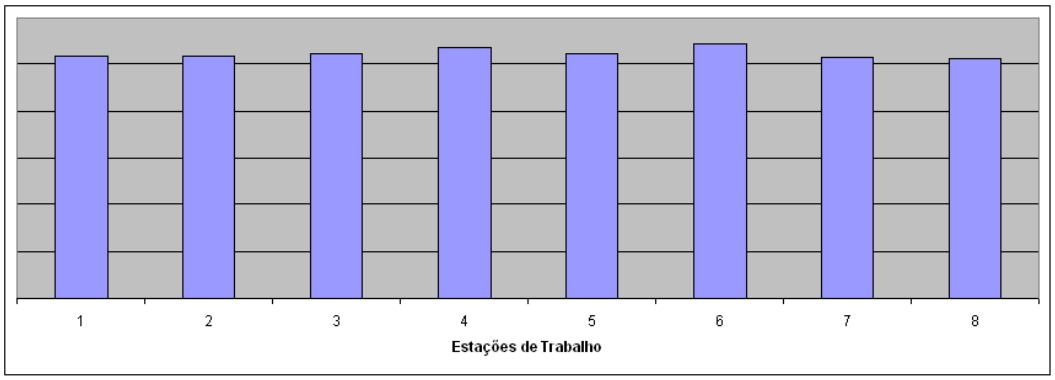

Figura 5.4: *Balanceamento de carga utilizando rS=32.*

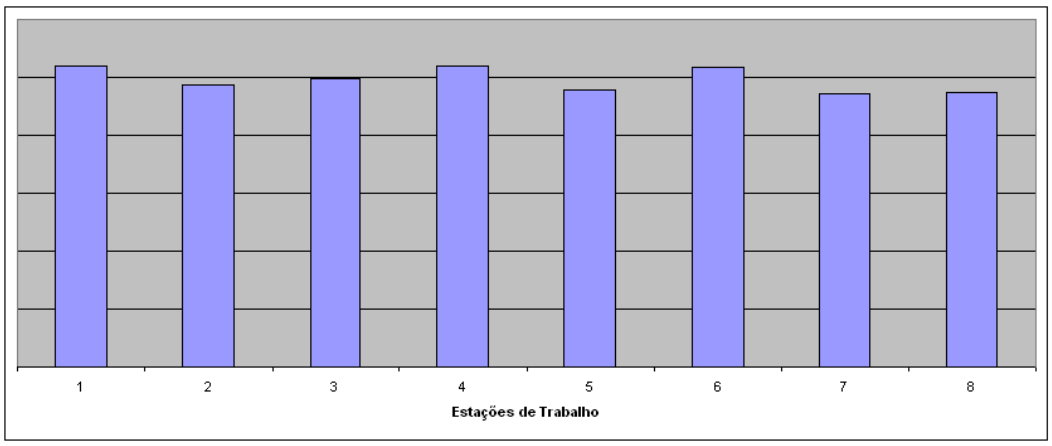

Figura 5.5: *Balanceamento de carga utilizando rS=64.*

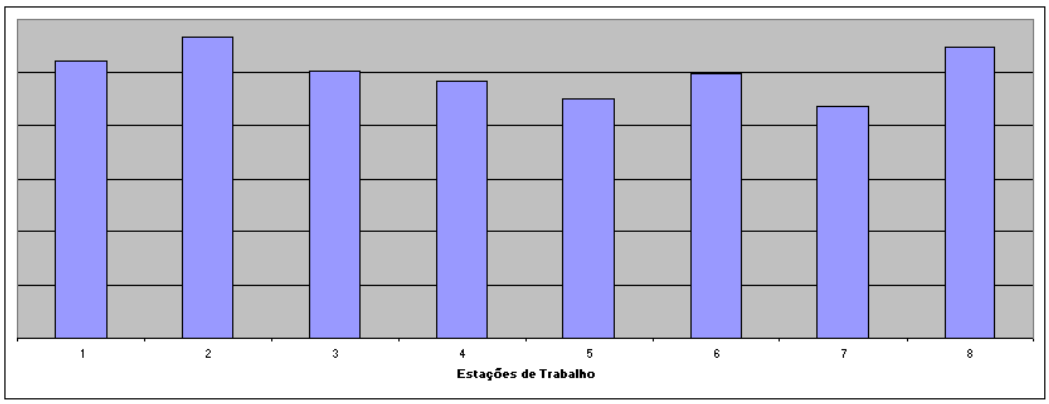

Figura 5.6: *Balanceamento de carga utilizando rS=128.*

5.1.2 Processamento em bancos de dados maiores que a memória principal

A primeira colaboração no intuito de obter melhores resultados em banco de seqüências maiores do que a memória primária é fornecida pela

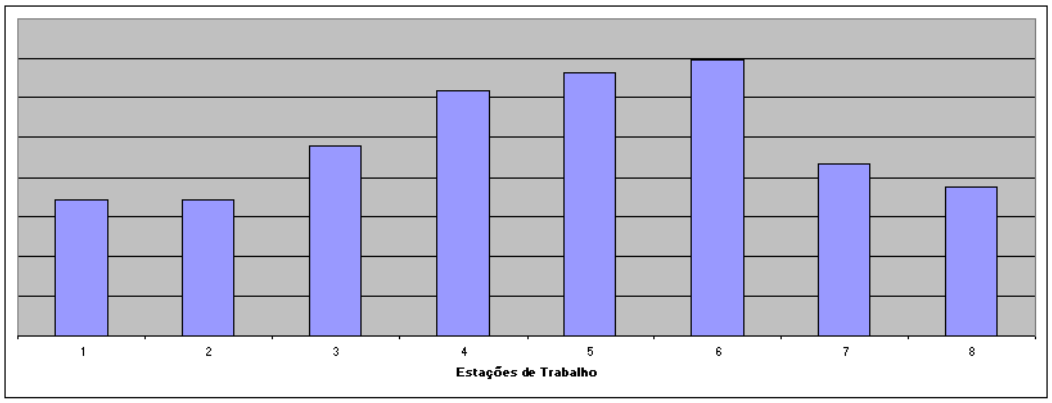

Figura 5.7: *Balanceamento de carga utilizando rS=256.*

estratégia de alocação da base de dados Replicada com Fragmentos Primários. Isso porque ela permite que cada máquina de trabalho trate fragmentos e não o banco de dados completo.

Existe ainda a segunda colaboração: a máquina de trabalho recebe uma tarefa, por exemplo, T(1 : 10),(1 : 5), e todas as seqüências (de 1 a 10) são processadas inicialmente contra o fragmento 1 e somente depois se usará o fragmento 2, até que o fragmento 5 seja utilizado. Dessa forma se espera um ganho de execução, pois ao processar a seqüência 2 é aproveitado que o fragmento da base de dados ainda se encontre na memória utilizado pela seqüência 1, evitando que a base seja carregada sempre que processar uma nova seqüência de consulta. Essa utilização dos dados mantidos em memória vai ocorrer até que a última seqüência seja processada, nesse caso a seqüência 10.

Ao analisarmos esses procedimentos para utilização dos dados em memória primária, com o aumento do número de rS é de se esperar que menor será o tempo de execução da tarefa - o que é verdade em um ambiente computacional dedicado ou exclusivo. Contudo, como vimos anteriormente, na medida em que o valor de rS aumenta, maior é a unidade de processamento, aumentando o desbalanceamento de carga. Em ambientes não exclusivos, a flexibilidade na utilização de recursos disponíveis irá favorecer o tempo final de processamento, obtidos com valores de rS menores.

Na subseção anterior, mostramos o desbalanceamento quando rS aumentava, entre: 2, 32, 64, 128 e 256. Nesta seção, para esses mesmos valores de rS, mostraremos na Figura 5.8 o tempo final de execução, comprovando a

diminuição no tempo de execução quando se aumenta o parâmetro rS.

A Figura 5.8 mostra que, para o nosso caso, o aumento excessivo no número de seqüências por tarefa não necessariamente conduzirá ao melhor desempenho do sistema. O mesmo ocorre quando poucas seqüências são inseridas por tarefa. Em nosso caso, concluímos que algo entre 32 e 64 seqüências sejam suficientes.

Acreditamos que o melhor ajuste de rS vai depender do ambiente em execução, se exclusivo ou não. Em execuções não exclusivas, com alta concorrência de máquinas, utilizar valores de rS não tão altos vai favorecer o balanceamento de carga.

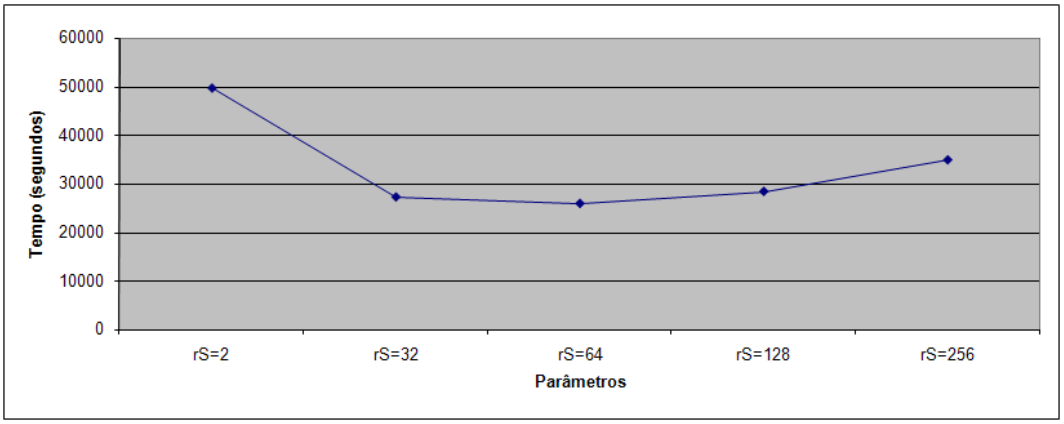

Figura 5.8: *Tempos de execução para estratégia Sob Demanda, variando os valores de rS. Para todos os tempos foram utilizadas as mesmas 1.000 seqüências de consulta e a base nt particionado em 24 fragmentos.*

5.1.3 Robustez

Em ambientes paralelos com número variado de máquinas é importante manter um sistema robusto. Em questão, nossa estratégia propõe robustez do sistema caso a estação de gerência pare de funcionar, assim como uma ou todas as estações de trabalho. Em especial, caso a ferramenta BLAST tenha qualquer problema de execução é considerado que a estação não estará mais operacional.

Para que nosso sistema possa se manter robusto, estamos considerando dois pontos de falha: nas estações de trabalho e na estação de gerência.

Manter o processamento em execução é facilitado pelo próprio funcionamento do sistema, já que uma tarefa é entregue à estação de trabalho somente se o resultado da última tarefa for recebido. Isso mantém a confiabilidade de funcionamento da estação de trabalho.

Para toda tarefa enviada a uma estação, é contabilizado o tempo da demora na entrega do resultado. Quando ocorre a identificação de que a tarefa expirou o tempo normal de processamento, a mesma tarefa será enviada a outra máquina, até que todas as tarefas sejam executadas.

Em referência às falhas ocorridas com a estação de gerência ou paralisação de todo o agrupamento, as consideramos com o mesmo tratamento. Para isso, nos baseamos na utilização de registros de processamento (log) normalmente utilizados em SGBDs (Sistema Gerenciador de Bancos de Dados). Em nosso caso, aplicamos o conceito de gravação adiada de dados (*write-ahead logging*) (14). Assim que uma quantidade de tarefas processadas for obtida pela estação de gerência, após gravado cada resultado em um arquivo de saída, será mantido um log registrando as seqüências processadas. O log é construído mediante confirmação de que o relatório de uma seqüência já foi corretamente gravado. Caso a estação de gerência ou todo o agrupamento pare de funcionar, ao retornar o funcionamento o sistema poderá continuar a partir de onde foi registrado no log.

As idéias aqui apresentadas para permitir robustez do sistema foram todas implementadas e testadas.

5.1.4 Estatísticas de Alinhamento

Outro problema também identificado no Capítulo 2 se refere à precisão dos dados nas estatísticas de alinhamento em fragmentos da base de dados. Como proposta a este problema, aplicamos o que foi sugerido em nosso trabalho(33), complementar a esta dissertação.

Nosso objetivo com o trabalho (33) foi mostrar o comportamento do e-value quando a ferramenta BLAST é executada com fragmentos da base de dados. Para isso foram analisados diversos resultados com uso variado de parâmetros dos programas BLAST (em especial, parâmetros z e y). Comenta-se também eventuais alterações no código BLAST, com uso direto das funções

de estatísticas de alinhamento, de forma intrusiva.

Foi notado que o uso do parâmetro z da ferramenta BLAST (19) possibilita e-values próximos do desejado, principalmente quando valores efetivos da base de dados são utilizados. Já o uso do parâmetro Y é outra boa opção, mas exige que sejam feitos cálculos para o espaço de busca efetivo antes que a comparação ocorra. Ferramentas, como mpiBLAST(11), possibilitam que os cálculos para uso do parâmetro Y sejam bem precisos, contudo exigem em contrapartida muito tempo de processamento para isso.

Obter e-values próximos dos valores de todo o banco de dados quanto se utiliza fragmentos não é uma tarefa trivial. Enquanto alguns pesquisadores exigem idênticos e-value, semelhantes aos originais, outros toleram pequenas diferenças e valores aproximados. Muitos pesquisadores dizem que a diferença obtida com o parâmetro z não é significativa, podendo variar dentro de uma taxa de 5% do valor original sem nenhuma perda de precisão (32).

Com os testes apresentados em (33), sugerimos que a melhor opção para manter a precisão do e-value em bases de dados fragmentadas é a utilização do parâmetro z com o tamanho efetivo da base de dados. A dificuldade para isso seria a obtenção do tamanho efetivo da base de dados. Contudo, aproveitamos a disponibilidade de todos os fragmentos em uma máquina e a possibilidade de executar o BLAST passando por parâmetro todos estes fragmentos em um único processo(19). Dessa forma, ao receber a primeira seqüência de consulta, somente esta será processada com todos os fragmentos. A partir do relatório de saída obtido, o tamanho efetivo da base é utilizado nos parâmetros das próximas execuções. Foi verificado que, quanto menor o fragmento, maior é precisão do *e-value*.

Outra discussão a respeito das estatísticas de alinhamento é o procedimento de incorporar precisão no *e-value* enquanto a base de dados está sendo atualizada durante o processamento. Pois, em situações como essas, o ideal é o sistema incorporar a atualização da base de dados aproveitando todo o processamento já ocorrido, e proceder com o restante da comparação contra a base de dados atualizada. Contudo, o problema aqui está no ato de proceder com o restante da comparação, pois com a base de dados alterada o valor que define o *e-value* é dependente do banco de dados alterado.

5.1.5 Resultados

No intuito de avaliar e comparar o desempenho da estratégia proposta, implementamos a estratégia Replicada e Fragmentada, sugerida em (9) e executamos também a estratégia mpiBLAST(11). No caso do mpiBLAST[1] variamos a quantidade de fragmentos para avaliar possíveis melhoras, no caso testamos 24 e 96 fragmentos. Executamos todas estas estratégias no intuito de observar o desempenho para base de dados maior que a memória principal, para isso utilizamos a base *nt*. Utilizamos o mesmo agrupamento de computadores, descrito no Apêndice 1, com o mesmo grupo de 1.000 seqüências de consulta como exemplo. Para a estratégia Sob Demanda, utilizamos alguns parâmetros que obtiveram bons resultados, como 24 fragmentos e 32 seqüências por tarefa.

Além dessas execuções, analisamos também o tempo serial da ferramenta BLAST para as mesmas seqüências de consulta. Contudo, nesse último caso, dada a dificuldade em manter uma máquina por tanto tempo em processamento, somamos o tempo total de execução BLAST em cada máquina de trabalho quando utilizada a estratégia Replicada. O resultado dessas execuções pode ser visto na Figura 5.9, que mostra a diferença de porcentagem em tempo de execução relacionado à estratégia Serial.

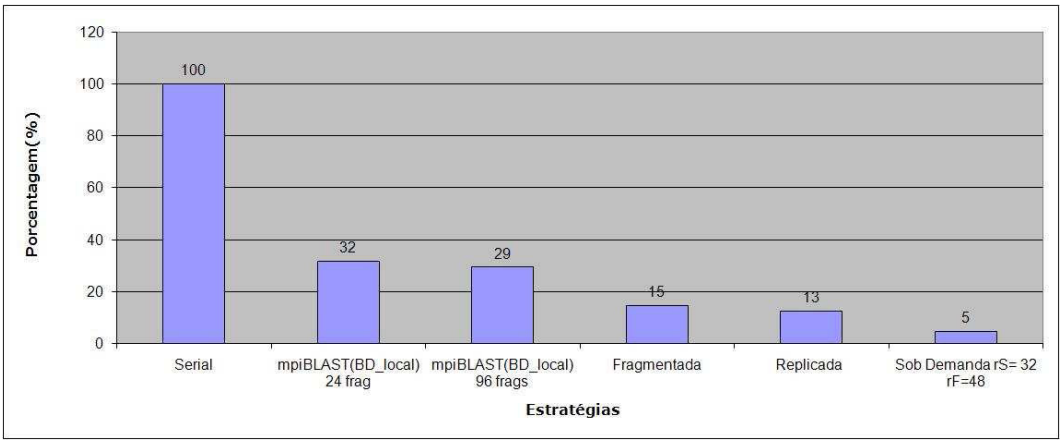

Figura 5.9: *Apresenta o resultado da execução de várias estratégias e o desempenho da Estratégia Sob Demanda, mostrando a porcentagem em tempo de processamento em relação à estratégia Serial. Em todas as estratégias foram utilizadas as mesmas seqüências de consulta e o mesmo banco nt.*

[1]Com o objetivo de acelerar a execução do mpiBLAST mantivemos todos os fragmentos da base de dados alocados localmente em cada estação de trabalho.

5.2
Estratégia Corretiva

Na Estratégia Corretiva, o processamento é dividido em tarefas de acordo com o número de estações de trabalho, de forma com que cada estação receba uma tarefa. Quando uma máquina terminar o processamento da tarefa que lhe foi designada, pode ainda existir máquinas processando outras tarefas, então a máquina ociosa poderá contribuir recebendo o pedaço de uma tarefa que se prolonga em outra máquina. O objetivo aqui é corrigir a alocação inicial de tarefas, envolvendo todas as tarefas para completar o BLAST, feita no início do processamento.

Analogamente, a execução paralela completa pode ser assim descrita:

T(1 : n)(1 : m),

sendo n o número de seqüências e m o número de fragmentos a serem utilizados. O primeiro passo da Estratégia Corretiva é dividir a tarefa a fim de que cada pedaço seja enviado para uma estação de trabalho. Logo, cada estação receberá a seguinte tarefa:

Expressão 3

T(1 : n)(xi : xi+rF)

sendo xi o primeiro fragmento a ser designado para a estação, e rF o intervalo de fragmentos utilizados. Ao somarmos todas as tarefas submetidas a cada estação de trabalho, uma única tarefa pode ser expressa:

Sejam: $\Delta m = \lfloor m/rF \rfloor$

Consideremos sem perda de generalidade, que o resto da divisão m/rf é igual a zero.

Expressão 4

$$T(1 : n)(1 : m) \equiv \sum_{i=0}^{\Delta m-1} T(1 : n)(i * rF + 1 : (i + 1) * rF)$$

Após uma estação de trabalho ter processado a tarefa recebida, se tornando ociosa, a máquina gerente toma conhecimento disto e aciona o módulo de realocação. Esse é responsável por verificar qual é a estação mais lenta e qual a quantidade da tarefa deve ser realocada.

Podem existir duas formas de realocar uma tarefa. Na primeira, o módulo de realocação decide somente que algumas seqüências serão processadas por um único fragmento na máquina ociosa. Na segunda forma, um fragmento inteiro será processado por todas as seqüências na máquina ociosa.

Cabe lembrar que no caso da estratégia aqui proposta, e por conta da alocação da base de dados replicada com fragmentos primários, não existe o tráfego de seqüências ou fragmentos no processo de realocação, pois esses já estão disponíveis em todas as máquinas. Existem apenas mensagens de metadados sendo enviados, informando e definindo a tarefa a ser executada.

A fim de facilitar o entendimento desta estratégia, mostramos na Figura 5.10 o seu funcionamento e procedimentos adotados. Note que o tratamento para montar os resultados é o mesmo explicado para a Estratégia Sob Demanda.

Nas próximas seções vamos apresentar as propostas dessa estratégia referente aos requisitos apresentados no Capítulo 2. Com relação ao tratamento de robustez e estatísticas de alinhamento, a estratégia segue as mesmas idéias das seções, 5.1.3 e 5.1.4

5.2.1 Balanceamento de carga

A estratégia aqui comentada parte de pressuposto que somente quando o desbalanceamento for identificado é que as atividades de realocação serão executadas. No intuito de sempre manter o sistema balanceado, qualquer máquina considerada ociosa poderá contribuir com a execução de uma tarefa. A Figura 5.11 apresenta o balanceamento de carga obtido com essa estratégia, utilizando 8 máquinas de trabalho e comparando 1.000 seqüências de consulta contra a base de dados nt.

Um papel importante nesta estratégia é o módulo de realocação, pois além de decidir qual estação será ajudada, também vai definir a quantidade

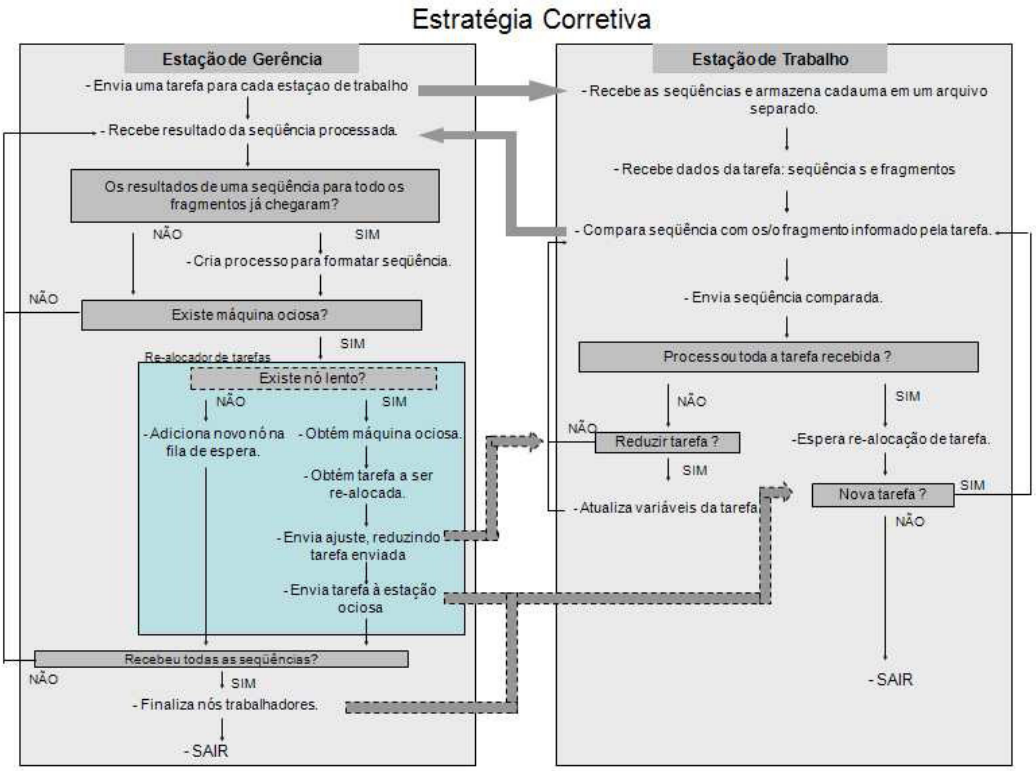

Figura 5.10: *Fluxograma de funcionamento da estratégia Corretiva*

de tarefa a ser realocada (o nível de granularidade da execução). Várias são as possibilidades de realocação, desde poucas seqüências a serem comparadas com um fragmento até todo um fragmento a ser processado com todas as seqüências de consulta.

Por exemplo, digamos que uma estação de trabalho esteja ociosa, enquanto outra que recebeu a tarefa T(1 : 100)(1 : 5), somente processou T(1 : 50)(1 : 3). Nessas condições ainda falta processar T(51 : 100)(3 : 3) + T(1 : 100)(4 : 5).

Em nossa implementação, o módulo de realocação inicialmente verifica (mantendo um controle das tarefas executadas) qual máquina esta mais lenta no processamento da tarefa recebida. Feito isso, será analisado se existe algum fragmento com o qual nenhuma seqüência foi processada. Em caso afirmativo, e considerando que o fragmento anterior a esse não esteja terminando, o fragmento que ainda não teve início será alocado para uma máquina ociosa. No exemplo anterior, o módulo de realocação iria transferir a tarefa T(1 : 100)(5 : 5) para a estação ociosa.

Caso não exista fragmento que nenhuma seqüência tenha sido com-

parado, o módulo vai realocar seqüências que ainda não foram comparadas ao fragmento que está em operação, permitindo que máquinas ociosas processem as últimas seqüências do intervalo. Ainda no exemplo anterior, considerando-se que ainda falte processar T(51 : 100)(3 : 3), a tarefa a ser alocada seria: T(99 : 100)(3 : 3). Na medida em que novas estações se tornem ociosas, poderão receber novas tarefas definidas de maneira análoga.

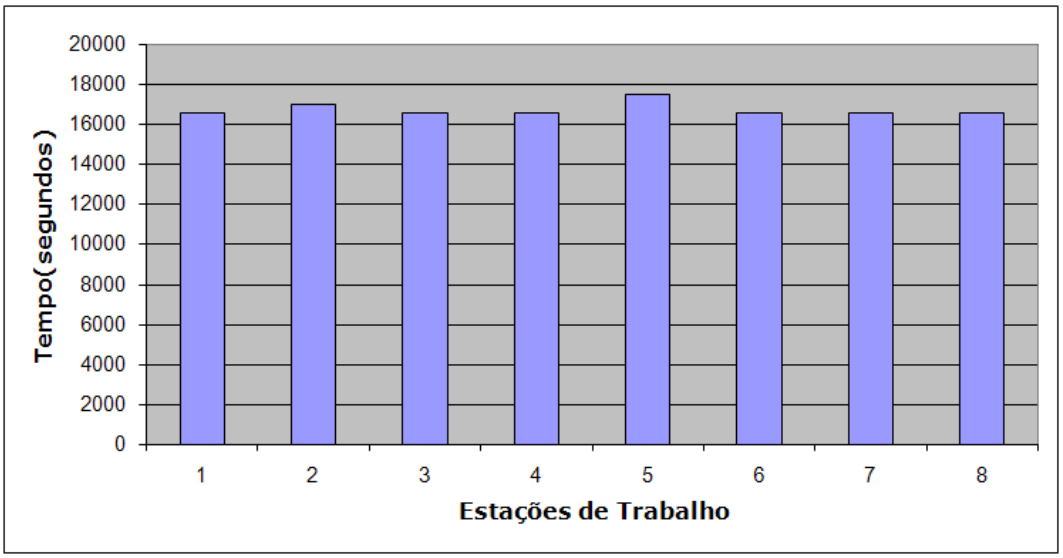

Figura 5.11: *Balanceamento de carga obtido pela Estratégia Corretiva.*

5.2.2 Processamento em bancos de dados maiores que a memória principal

A própria disposição dos dados possibilita melhorar o desempenho quando utilizado grandes bancos de dados. Mas além dessa abordagem, a estratégia aqui sugerida possibilita um avançado aproveitamento dos dados já contidos em memória primária.

Isso ocorre porque sempre que a estação de trabalho recebe uma tarefa todas as seqüências de consulta são processadas contra um fragmento e, somente após isso, as seqüências serão processadas contra outro fragmento. Dessa forma, $n-1$ seqüências vão se aproveitar dos dados ainda mantidos em memória, melhorando significativamente o tempo de execução da ferramenta.

5.2.3

Resultados

Também para a estratégia aqui apresentada, comparamos com as estratégias Replicada, Fragmentada sugeridas em (9), além da execução Serial e mpiBLAST(11). Utilizamos o agrupamento de computadores descrito no Apêndice 1. Veja os resultados da comparação na Figura 5.12. Também nessa figura os valores se referem à porcentagem em relação à estratégia Serial. Nosso objetivo com a figura é mostrar o comportamento das estratégias utilizando uma base de dados maior que a memória principal (*nt*), recebendo as mesmas 1.000 seqüências de consulta. A Estratégia Corretiva teve 96% de ganho se comparada à estratégia Serial.

O bom desempenho se deve principalmente à otimização na utilização dos fragmentos mantidos em memória, evitando que sejam alternados freqüentemente entre memória primária e secundária.

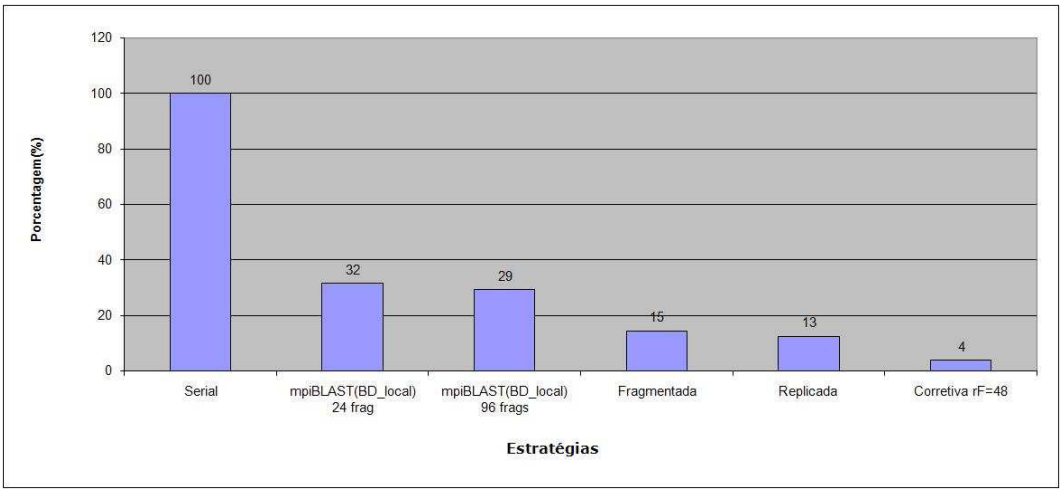

Figura 5.12: *Compara a Estratégia Corretiva em relação à varias outras, mostrando a porcentagem em tempo de processamento em relação à estratégia Serial. Em todas as estratégias foram utilizadas as mesmas seqüências de consulta e o mesmo banco nt.*

5.3 Conclusão

Neste Capítulo sugerimos duas estratégias de execução paralela, denominadas *Sob Demanda* e *Corretiva*. Ambas dividem uma tarefa em várias partes, e as submetem para as máquinas de trabalho de forma a obter melhor balanceamento possível.

As duas estratégias utilizam a mesma alocação da base de dados. Considerando que todo o banco esteja replicado e cada réplica por sua vez fragmentada, com um ou mais fragmentos ditos primários.

Ao relembrar do Capítulo 2, observamos que as estratégias aqui propostas atendem os principais requisitos: balanceamento de carga, robustez, correção dos valores estatísticos e uma alternativa à leitura de bancos de dados maiores que a memória primária.

Mostramos inicialmente que as estratégias tiveram ótimo desempenho ao compararmos com outras estratégias conhecidas, como puramente Replicada, Fragmentada e mpiBLAST.

Em relação ao balanceamento de carga, embora não tenhamos feito testes em ambiente com processos concorrentes e de diferentes configurações de máquinas, o balanceamento obtido com os desvios de fatores de similaridade e tamanho já justificam a sugestão.

Importante deixar claro que a nossa estratégia na alocação replicada da base de dados visa somente o bom desempenho durante o processamento. Caso não seja possível replicar a base em cada máquina, a confiabilidade e correção das estratégias se mantêm.