5
Adaptatividade

O ambiente de grade se caracteriza por ser um ambiente volátil, no qual a todo instante usuários podem submeter tarefas sem a prévia alocação de recursos. Em ambientes dinâmicos como este, o processo de otimização de consultas convencional é inadequado, pois se baseia inteiramente na utilização de informações pré-computadas e armazenadas no catálogo do sistema. A solução para este problema encontra-se na adoção de mecanismos que permitam ao sistema responder às variações nas condições ambientais de execução da consulta de modo a garantir uma execução com custo mínimo.

Neste capítulo propomos uma estratégia de adaptação voltada para o modelo orbit que permite ao sistema adaptar o nível de paralelismo utilizado de acordo com o volume de dados existente e realizar um balanceamento de carga entre os nós que participam da execução paralela de um fragmento. O objetivo é priorizar os nós apresentando maior desempenho, de forma a minimizar o tempo total de execução.

O restante do capítulo encontra-se organizado da seguinte forma: inicialmente apresenta-se o objetivo desta estratégia e como ela pode ser combinada com a arquitetura de execução de consulta proposta no capítulo anterior. Em seguida, apresenta-se o modelo de custo que serve de base para o balanceamento de carga dos nós paralelos. O processo de adaptação em si é dividido em três etapas: monitoramento, análise e resposta. As seções seguintes cobrem, respectivamente, estes três aspectos segundo as necessidades do modelo de execução orbit.

5.1. Introdução

As grades computacionais surgiram com o intuito de ser um ambiente colaborativo, na qual dados, aplicações e recursos computacionais podem ser compartilhados entre seus usuários. Este tipo de ambiente se caracteriza por seu dinamismo, no qual a disponibilidade dos recursos computacionais pode variar

com bastante freqüência e a presença de informações estatísticas é quase inexistente. Neste tipo de ambiente o processo de otimização de consultas tradicional é ineficaz, pois se baseia somente na utilização de informações disponíveis em tempo de otimização. Uma solução para cenários como este é a utilização de técnicas que permitam o sistema se adaptarem a execução de acordo com informações que possam ser coletadas dinamicamente.

A execução de consultas no modelo orbit pode apresentar um elevado tempo de execução de acordo com volume de dados e o número de iterações a serem realizadas. Para solucionar este problema, a arquitetura de execução apresentada no capítulo anterior propõe que os recursos disponíveis no ambiente de grade sejam utilizados para execução paralela das órbitas. Neste contexto, é extremamente importante que o sistema seja capaz de adaptar o nível de paralelismo e a distribuição dos dados em função do volume dos dados disponível e a carga dos recursos.

Do ponto de vista dos recursos, a adaptatividade se faz necessária para evitar situações em que nós mais lentos recebam grande parte dos dados a serem processados e aumente o tempo total de execução. Repare que no modelo orbit este problema é ainda mais significativo, pois um atraso na execução de uma iteração acarretaria no atraso de todas as demais iterações daquele conjunto de dados, estendendo o tempo de execução da consulta.

Já do ponto de vista do volume dos dados, a adaptatividade evita a ocorrência do problema conhecido por disputa por dados (ERHARD RAHM & ROBERT MAREK, 1995). Este problema ocorre durante a execução, na medida em que tuplas terminam suas órbitas em instantes de tempo distintos, provocando uma diminuição no volume de dados. Neste cenário o nível de paralelismo deveria ser reduzido de acordo com o volume de dados disponível.

As seções seguintes apresentam as técnicas desenvolvidas neste trabalho para reagir às variações de condições de execução de consultas no modelo orbit.

5.2. Modelo de custo

No desenvolvimento de aplicações para o ambiente de grade, onde os recursos computacionais encontram-se geograficamente distribuídos, o custo de comunicação deve ser cuidadosamente ponderado, pois pode degradar

profundamente a execução. Por exemplo, o atraso na distribuição de dados do/para um nó remoto pode levá-lo a ficar sem tarefas a realizar e aumentar o tempo de execução. Por outro lado, uma excessivas troca de mensagens interrompe constantemente o processamento e pode igualmente aumentar o tempo total de execução.

Desta forma, o que se almeja é definir um modelo de execução paralelo no qual todos os nós alocados apresentem uma mesma alocação máxima de recursos. Idealmente, uma execução maximal é aquela em que os nós estão sempre realizando tarefas úteis à solução da consulta. Em particular, em um ambiente de execução paralela semelhante ao *memória compartilhada*, como o presente, deve-se minimizar as situações de espera por tuplas em um nó remoto.

Entretanto, o modelo de execução orbit guarda em sua natureza uma restrição à garantia de tal execução maximal. O leitor deve lembrar-se que no modelo orbit tuplas retro-alimentam a execução, a cada iteração, uma vez que tenham sido processadas. Dessa maneira, o modelo de custo deve considerar que a garantia de execução contínua em um nó depende da continuidade do ciclo, compreendendo transmissão de blocos de tuplas pela rede e sua avaliação pelo nó remoto.

Parâmetro	Descrição
N	Número de nós disponível
Mc_i	Custo de enviar uma mensagem do nó central ao nó i (MB/ms)
Nt_i	Throughput do nó i (ms/tupla)
B_i	Tamanho do bloco de comunicação recomendado para o nó i
Q	Volume de dados disponível (nr. tuplas)
Q_i	Volume de dados alocado a um nó (nr. tuplas)
Ic	Custo de inicialização
It	Número de iterações

Tabela 4. Parâmetros de custo

Frente a tal contexto, o modelo de custo empregado concentra-se nos seguintes parâmetros de execução: custo de comunicação entre o nó central e cada um dos nós remotos de execução, a vazão (*throughput*) de cada nó, considerado em tempo necessário para avaliar uma tupla, e no volume de dados disponível.

Queremos dessa forma definir um modelo de custo que tomando em considerações os parâmetros acima, minimize uma função de custo que leve a uma execução minimal da consulta. Tal modelo indicará os nós a serem utilizados e a quantidade de tuplas a lhes serem alocadas. A tabela 4 apresenta os parâmetros utilizados pelo modelo de custo.

Na implementação de um modelo de execução Orbit, em que o volume de dados é restrito e o processamento dos nós remotos deve ser contínuo, a influência do custo de comunicação e do número de tuplas enviadas ao nó remoto (B_i) é ainda maior. Para garantir que a execução em um nó seja contínua e que o tempo de comunicação não predomine sobre o tempo de execução, o modelo de custo considera que uma situação ideal é aquela em que enquanto um bloco de dados é avaliado por um fragmento remoto, um segundo realiza a órbita ficando disponível no nó remoto quando este requisitar mais dados. Para determinar B_i neste contexto adotamos a seguinte equação:

$$B_i Nt_i \geq 2Mc_i \Delta \Rightarrow B_i \geq \left\lceil \frac{2Mc_i}{Nt_i} \right\rceil \Delta \quad (1)$$

Na equação (1), determina-se que dado um nó remoto *i* com vazão Nt_i, precisa-se de um bloco de tuplas de tamanho B_i para que o tempo de envio de um outro bloco de tuplas B_j qualquer (não presente na equação) possa circular pela órbita e re-alimentar o próprio nó *i* no próximo ciclo. Fica claro pela equação (1) que, para obter tal resultado, o tempo de execução do bloco enviado ao nó ($B_{i*} Nt_i$) não deve ser inferior ao tempo de transferência de um bloco pelo ciclo ($2Mc_i$).

Adiciona-se à equação um parâmetro de ajuste Δ para tratar os casos em que o tempo de processamento é próximo ou inferior ao tempo de transmissão. Nesses cenários, o tamanho de bloco se reduz a uma tupla o que pode gerar uma troca de mensagens excessiva entre o nó central e o nó remoto. Sendo assim, deve-se estimar um tamanho *k* de bloco mínimo desejado, fazendo $\Delta=\{1,k\}$. Onde $\Delta=k$ quando $Nt_i \cong Mc_i$ e $\Delta=1$ em todos outros cenários.

Aprofundando-se na análise de B_i, chega-se a conclusão de que seu tamanho defini o perfil de execução de um nó remoto. Por exemplo, caso a um nó sejam

alocadas $Q_i \geq 2B_i$ tuplas, podem-se construir dois blocos de tamanho $\geq B_i$, garantindo uma execução maximal contínua àquele nó.

Frente a tal conclusão, pode-se vislumbrar fornecer um volume de dados sempre superior à $2B_i$ para cada nó. Entretanto, essa decisão não considera o aspecto global da execução, em particular a diferença de desempenho de cada nó e a quantidade total de tuplas disponíveis. Dessa forma, faz-se necessário definir três funções de custo, segundo a quantidade de tuplas Q_i alocadas a um dado nó. Apresentam-se, a seguir, as três funções de custo:

Transparência Total → este é o cenário de execução ideal, pois enquanto o fragmento processa um bloco de tuplas, um segundo bloco completa sua órbita, garantindo que a execução não seja interrompida por falta de dados. Apenas os custos de transmissão inicial e final influenciam no resultado. Neste caso $Q_i \geq 2B_i$ e o custo da execução é calculado como:

$$C(Q_i) = It(Q_i Nt_i) + 2Mc_i \quad (2.1)$$

Transparência parcial → corresponde ao cenário em que $B_i < Q_i < 2B_i$. A função de custo neste caso pode ser modelada como:

$$C(Q_i) = It(Q_i Nt_i + (2Mc_i - (Qi - Bi)Nti)) + 2Mc_i \rightarrow C(Qi) = It(2BiNti) + 2Mci \quad (2.2)$$

Este resultado mostra que independentemente do número de tuplas entre B_i e Q_i o custo da execução é sempre constante. Entretanto, um aspecto especial a ser considerado neste cenário é a eficiência da execução, que se reduz à medida que a diferença $(Q_i - B_i)$ diminui. A eficiência é definida como:

$$Ef = (1 - \frac{Q_i - Bi}{B_i}) \Rightarrow 0 < Ef < \frac{Bi - 1}{Bi}$$

Essa medida pode ser associada a função de custo 2.2 e oferecer um critério de desempate para as distribuições neste cenário e, sendo assim, a função de custo deste cenário é expressa por:

$$C(Qi) = It(2BiNti) + 2Mci + Ef \quad (2.3)$$

Sem transparência → neste caso $Q_i < 2B_i$ e a execução acontece de forma seqüencial. O custo da execução é definido por:

$$C(Q_i) = It\left(Q_i Nt_i + \left(2Mc_i - \left(\frac{Qi}{2}\right)Nti\right)\right) + 2Mc_i$$
$$\rightarrow C(Q_i) = It\left(2Mci + \left(\frac{Qi}{2}\right)Nti\right) + 2Mci \quad (2.3)$$

Finalmente, podemos definir a função de custo global a ser minimizada como:

$$f(Q,n) = Min_{i=1,k}\,(Max_{j=1,n} C(Q_{i,j})) \quad (3)$$

onde *i* representa as *k* possíveis maneiras de se distribuir *Q* tuplas por *n* nós. A equação (3) explora todos as possíveis distribuições de 'Q' tuplas sobre os nós disponíveis. Para cada distribuição $Q_{i,j,}$ a função de custo relevante (2.1 ou 2.2 ou 2.3) é utilizada, em função da relação entre Q_i e B_i, para o cálculo do custo daquela configuração. Como será visto na seções 5.3.3.1, o algoritmo G^2N implementa uma versão otimizada da equação (3), no qual o número de cenários analisados é significativamente menor.

5.3. Estratégia de Adaptação

No âmbito de sistemas paralelos dois tipos de adaptação podem ser observados: local e global. A adaptação local é baseada apenas nas informações de um nó de execução e o objetivo é minimizar apenas seu custo. Já a adaptação global é realizada a partir de todas as informações disponíveis no sistema e visa minimizar o custo total da execução. No contexto deste trabalho é apresentada uma estratégia de adaptação global que visa minimizar o tempo total de execução,

adaptando o nível de paralelismo utilizado e a distribuição dos dados segundo o desempenho de cada nó.

A estratégia de adaptação desenvolvida é guiada pelo modelo de custo apresentado na seção anterior, essa abordagem incorre no custo da captura informações durante a execução, mas, por outro lado, se comparadas a estratégias heurísticas (L.AMSALEG, M.J..FRANKLIN et al., 1996), possuem uma maior precisão. A figura 10 ilustra como essa estratégia pode ser integrada à arquitetura de execução de consulta apresentada no capítulo 4. A partir das informações de desempenho coletadas dos fragmentos e do volume de dados disponível, indicado pelo conceito da sobra – seção 5.3.2 –, o algoritmo de escalonamento G^2N (*Grid Greedy Node*) determina a melhor distribuição de dados a ser utilizada.

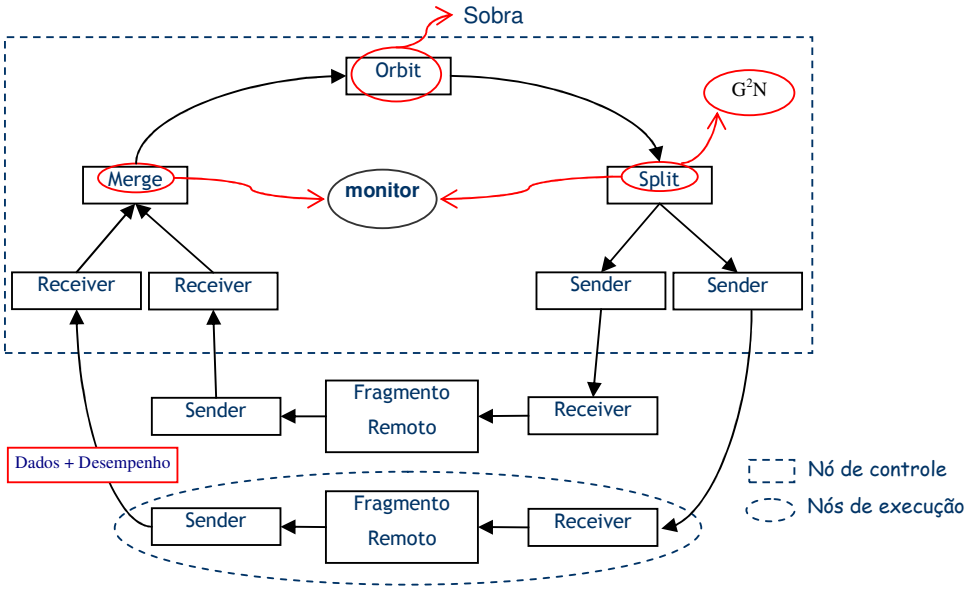

Figura 10. Pontos de adaptação

O processo de adaptação, conforme proposto em (ANASTASIOS GOUNARIS, NORMAN W.PATON et al., 2004), pode ser dividido em três fases: monitoramento, análise e resposta. Na fase de monitoramento são capturadas informações sobre o andamento da execução como, por exemplo, informações sobre o ambiente e estatísticas dos dados (i.e. seletividade). Em uma segunda etapa, estas informações são analisadas com o objetivo de avaliar se as condições previstas inicialmente se mantiveram e, sendo assim, a execução da consulta será realizada com o custo esperado. Se alguma alteração foi detectada o sistema entra em uma terceira fase que define quais as alterações devem ser realizadas para

corrigir a execução. As subseções a seguir detalham o funcionamento de cada fase no contexto deste trabalho.

5.3.1. Monitoramento

A fase de monitoramento desempenha um papel importante no processo de adaptação, pois a ausência de informações precisas pode induzir o sistema a tomar decisões que degradem ainda mais a execução. Ao mesmo tempo em que as informações devem ser precisas, sua captura não deve representar uma fração substancial do tempo de processamento esperado, pois também inviabilizaria a adaptação.

Diferentes abordagens podem ser encontradas na literatura para se monitorar a execução de consultas (NAVIN KABRA & DAVID J.DEWITT, 1998) (P.J.HAAS & JOSEPH M.HELLERSTEIN, 1999) (F.FABRET, 2000). A escolha da melhor técnica está intimamente relacionada ao nível de adaptatividade que se deseja realizar, pois definirá as informações a serem capturadas.

Neste trabalho propõe-se o monitoramento de, e conseqüente adaptação frente à, três aspectos da execução do modelo orbit de forma a minimizar o tempo de execução: distribuição de dados entre os nós que participam da execução paralela de um fragmento do plano; variação de desempenho de cada nó e volume de dados disponíveis. Correspondentemente, três informações devem ser capturadas durante a execução: o volume de dados disponível, o desempenho de cada nó e a taxa de transmissão entre o nó central e os nós remotos.

O volume de dados disponível pode ser facilmente obtido através do operador *Orbit,* uma vez que ele utiliza esta informação para detectar o final da execução (ver seção 4.2.1). Esta informação é disponibilizada no quadro de comunicação sob a chave *data_volume* e assim pode ser utilizada por qualquer operador do plano.

Para calcular o desempenho de cada nó utiliza-se uma estratégia semelhante à (NAVIN KABRA & DAVID J.DEWITT, 1998), na qual um operador de controle é adicionado ao plano para capturar as informações necessárias. Na estratégia de monitoramento desenvolvida os operadores *Instance2Block* e *Block2Instance* são estendidos de forma a computar o *throughput* (tuplas/ms) de cada nó de execução. As informações capturadas são enviadas ao nó central

através dos blocos de comunicação e disponibilizadas para os operadores da máquina no quadro de comunicação.

No modelo de execução proposto no capítulo anterior, a execução no nó remoto é realizada de forma assíncrona e, os operadores *Instance2Block* e *Block2Instance* são responsáveis por implementar tal característica. Desta forma, estes operadores podem facilmente calcular o throughput, pois possuem o tempo gasto no calculo de cada tupla e ainda são capazes de identificar quando um nó está sem dados para ser processado. O cálculo do *throughput* tem início no operador *Block2Instance* que ao receber um bloco de dados registra o número de tuplas disponíveis para processamento em um *timer*(relógio) que é acessível pelo quadro de comunicação. Ao consumir dados do fragmento remoto o operador *Instance2Block* registra o início e fim de processamento de cada tupla junto ao *timer*(relógio) que acumula o tempo gasto até o momento atual. Ao final do processamento de um bloco, operações de *reset*(reinicialização) são realizadas. Esta operação retorna o tempo total acumulado e inicia o *Timer*(relógio) para o cálculo do próximo bloco. A 0 ilustra este processo.

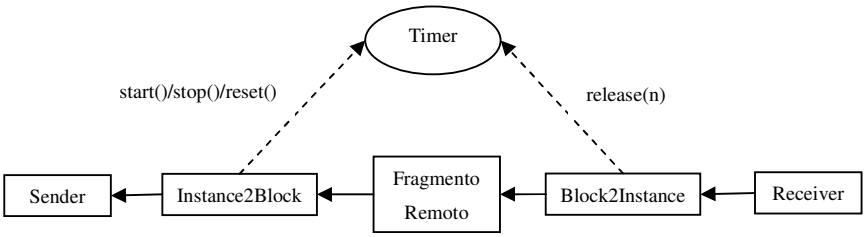

Figura 11. Cálculo do *throughput* em um nó de execução

Por fim, o cálculo do desempenho de cada nó é realizado através do throughput médio das taxas em uma janela. Pois a utilização do throughput acumulado poderia não refletir o desempenho atual do nó e a utilização de uma única entrada poderia levar o sistema a se adaptar a pequenas oscilações.

O cálculo da taxa de comunicação não foi estudado neste trabalho e, assume-se que esta informação poderia ser obtida a partir de serviços de compõem a grade como, por exemplo, o MDS (*Metacomputing Directory Services*) do globus.

5.3.2. Análise

Na fase de análise as informações capturadas anteriormente são avaliadas de forma a identificar se a execução da consulta ocorre de acordo com o planejado inicialmente. Se algum desvio for detectado a fase de resposta é iniciada e um conjunto de ações serão realizadas para adaptar a execução às novas condições do ambiente. A decisão de iniciar a fase de resposta deve ser realizada de forma cautelosa, pois pode implicar em um elevado custo e, desta forma, não deve considerar pequenas oscilações do ambiente.

De acordo com a estratégia de adaptação utilizada, a fase de análise deve verificar a disponibilidade de dados e o desempenho de cada nó utilizado.

Na execução de consultas em orbita o nível de paralelismo deve ser alterado de acordo com o volume de dados disponível (ver equação (3)). Considerando, por exemplo, tuplas que deixem suas órbitas, reduzindo o volume de dados total disponível. Neste cenário, não é mais possível manter o tamanho dos blocos de comunicação previamente estipulados. Em cenários extremos, nós apresentando menor desempenho podem chegar a consumir todas as tuplas disponíveis e aumentar drasticamente o tempo de execução. Este problema é conhecido na literatura como contenção de dado.

Para evitar a disputa por dados entre os fragmentos e permitir adaptação quando da modificação no volume de dados, define-se o conceito da *Sobra*. Este último indica aproximadamente a quantidade de dados disponíveis no nó central necessária para se manter uma determinada configuração de distribuição de dados. A *Sobra* determina um *threshold* para o monitoramento de volume de dados e, pode ser calculada pela formula a seguir:

$$S \geq \delta \sum_{i=0}^{n} \frac{Q_i}{2} \quad (4)$$

Se observarmos cuidadosamente o modelo de custo utilizado, percebemos que enquanto metade dos dados de um nó é processada, a outra metade encontra-se disponível para distribuição no fragmento central. Desta forma, todos os nós serão atendidos na sua próxima requisição, permitindo a manutenção da configuração de distribuição de dados. Caso a quantidade de tuplas no nó central

seja menor do que *S*, uma nova configuração deve ser computada, provavelmente levando à redução do nível de paralelismo. O parâmetro δ funciona apenas como um fator de ajuste. A avaliação da *Sobra* é realizada a cada solicitação por dados ao nó central.

A avaliação de desempenho é realizada a partir da análise do custo atual de cada nó, que pode ser calculado a partir da função de custo do cenário no qual o nó se encontra. Nesta análise, utilizamos um monitor que verifica se o desempenho de um nó é maior ou menor que um certo *threshold* do previsto inicialmente. Esta avaliação não é realizada a cada requisição, pois poderia levar a um cenário de super-reatividade e, sendo assim, optou-se por realizá-la de tempos em tempos segundo um parâmetro Δ configurável no monitor. A fórmula (5) representa os limites para tomada de uma ação:

$$TempoDecorrido - \Delta \leq \sum Ci(Qi) \leq TempoDecorrido + \Delta \quad (5)$$

5.3.3. Resposta

A fase de resposta está relacionada às ações que devem ser tomadas para adaptar o restante da execução às condições atuais do ambiente. Dependendo do âmbito em que estas ações são tomadas elas podem ser classificadas em locais ou globais. As ações locais são aquelas relacionadas a um nó específico e podem envolver desde a re-estruturação do fragmento do plano de execução de um nó ou a simples troca de um operador físico para adaptar às condições de memória atual. Já as ações globais são realizadas com base nas informações de todos os nós e, por sua vez, podem envolver a adição/remoção de novos nós, atualização da distribuição dos dados (balanceamento de carga), ou até mesmo, a re-formulação de todo o plano de execução. Neste caso, de acordo com a estratégia de adaptação apresentado, deve-se adaptar a distribuição dos dados para permitir o balanceamento dos dados.

O processo de adaptação é baseado em uma extensão do algoritmo de escalonamento G^2N (*Grid Greedy Node*) que, utiliza o modelo de custo apresentado na seção 5.2 para definir o custo de uma distribuição. A partir Esta versão encontra-se completamente integrada à máquina de execução de consulta e sempre que um desvio for detectado o algoritmo é executado a fim de definir uma

nova distribuição de dados, adequando o nível de paralelismo às condições atuais da execução. Conforme veremos na próxima seção este algoritmo é capaz de escolher a melhor distribuição de dados possível, representada pela equação (3), a partir da análise de um subconjunto do espaço de soluções.

5.3.3.1. G^2N

O G^2N (*Grid Greedy Node*) (VINÍCIUS FONTES, MARCIO L DUTRA et al., 2004) (FABIO PORTO, VINÍCIUS FONTES et al., 2005) é um algoritmo de escalonamento desenvolvido no contexto do projeto CoDIMS-G. Seu objetivo é definir um subconjunto de nós a ser utilizado de forma a minimizar o tempo de execução da consulta. Este algoritmo foi desenvolvido para escalonar programas que podem ser avaliados tupla a tupla e cujo tempo de processamento unitário é aproximadamente constante. Sua única restrição é que a avaliação de uma tupla ou tarefa deve ser independente de algum nó, ou seja, possa ser executada em qualquer um dos nós disponíveis.

Seu funcionamento é baseado no histórico de execuções dos programas em cada nó. Este histórico é armazenado no Gerente de Metadados do CoDIMS-G, ver capítulo 4, e encontra-se disponível para o processo de otimização. O algoritmo é ilustrado pela figura 12.

O algoritmo recebe como entrada a lista de nós disponíveis, os respectivos tempos médios de avaliação de uma tupla (tp_1, tp_2,..., tp_n), as taxas médias de transmissão Mc_i (nós de execução ↔ nó de controle) e o total estimado de tarefas (T) a serem executadas. Para um processamento circular o total de tarefas pode ser definido como o produto entre o número de tuplas a serem processadas (NT) e o número de iterações (NI). Nos cenários em que os tempos médios de avaliação de uma tupla e as taxas de transmissão não se encontram disponíveis, tipicamente na primeira execução da aplicação, uma estimativa para estes valores pode ser utilizada. À medida que a aplicação é executada valores reais para estas variáveis são capturados e assim o sistema é capaz de se adaptar.

Inicialmente o algoritmo ordena os nós disponíveis de acordo com a taxa média de produção, e os nós mais rápidos são privilegiados (ordem não crescente). Em seguida o tempo de avaliação das tarefas é calculado para uma alocação unitária, onde o nó mais rápido realiza todo o processamento. Neste

momento, calcula-se o custo desta configuração utilizando-se uma das funções de custo discutidas na seção 5.2. A partir daí, o laço principal do programa tenta redistribuir as tarefas já alocadas para um novo nó (o próximo da lista) até que não seja mais possível reduzir o tempo total de execução.

```
G2N (throughput(tp_1, tp_2, , tp_n), number-tasks):result
nodelist:= descending order(throughput);
result:= result nodelist(1);
cost(1):= number-tasks * nodelist(1);
current-cost:=cost(1);
While
    (nodes in the list and add-new-node)
    total-cost:= current-cost;
    new-node:= next-node in nodelist;
    While (current-cost <= total-cost)
        move tuples from lowest node in result to new-node;
        Update costs of nodes and total-cost;
        If current-cost > total-cost
          If we could move at least 1 tuple to the new-node
            result:= result new-node
          else
            add-new-node:=false;
          Endif
          Stop loop;
        Endif
    endwhile
enwhile
output result;
```

Figura 12. Algoritmo G2N

A redistribuição das tarefas é realizada de forma unitária, a todo momento o nó com tempo de execução predominante cede uma tarefa para o novo nó a ser alocado e o tempo de execução é atualizado. Para que a nova configuração seja mais eficiente, o novo nó deve ser capaz de processar todas as tarefas atribuídas a ele até o momento, em tempo menor ou igual ao tempo de execução calculado com a configuração anterior. Como observado, a redistribuição de tuplas para um nó sendo alocado, tem como origem aqueles apresentando o tempo total igual ao tempo máximo calculado até aquele instante. Durante este processo, podem ocorrer situações em que um conjunto de nós apresente tempos iguais e, um

tratamento especial deve ser realizado, pois para reduzir o tempo de execução estes nós devem deixar de executar ao menos tarefa. Nestes casos a redistribuição das tarefas deixa de ser unitária, e o conjunto de nós restante pode ser utilizado.

A execução do algoritmo atinge o fim quando um nó não é capaz de contribuir com o processamento de ao menos uma tupla ou não existirem mais nós a serem adicionados. Podemos expressar a condição de término pela equação (6) abaixo:

$$\sum_{i=1}^{k} Ci(Qi) > \sum_{i=1}^{k} Ci(Qi) - Ntj + Ntw,_{w=k+1}^{w=k} \text{ onde } Ntj = \overset{k}{Max}_{j=1}(Ci(Qi)) \quad (6).$$

Na equação (6), *Ntj* corresponde ao custo de avaliação de uma tupla no nó 'j' de maior custo e *Ntw* corresponde a adicionar uma tupla ao nó que se está adicionando.

Para tratar situações de distribuição onde um conjunto de 'J' nós apresenta tempo estimado igual ao tempo máximo é necessário retirar 'J' tuplas, uma de cada um dos nós para que o tempo total decresça. Temos então que a condição de parada define-se como:

$$\sum_{i=1}^{k} Ci(Qi) > \sum_{i=1}^{k} Ci(Qi) - \sum_{j=1}^{n} Ntj + n \times Ntw,_{w=k+1}^{w=k}, \text{ onde } Ntj[\ ] = \overset{n}{Max}_{j=1}(Ci(Qi))$$

(7)

Na equação (7), *n* corresponde ao número de nós com custo igual ao custo máximo.

5.4. Síntese do Capítulo

Neste capítulo apresentamos a estratégia utilizada para adaptar a execução de consultas no modelo *Orbit* às constantes variações que podem ocorrer em um ambiente de grade. Esta estratégia permite ao sistema adaptar o nível de paralelismo utilizado de acordo com o volume de dados disponível, evitando que aconteça o problema conhecido como disputa por dados. Na presença deste problema, nós com desempenho inferior poderíamos obter todos os dados e aumentar o tempo total de execução. Além disso, esta estratégia permite ao sistema balancear a distribuição dos dados entre os nós que participam do processamento paralelo das órbitas. Este benefício é alcançado parte pelo algoritmo de escalonamento que, define a quantidade de dados ótima a ser enviada

a cada nó e, parte pelo design de arquitetura descrito no capítulo 4 que permite ao sistema utilizar blocos de tamanho variável. Esta política de distribuição apresenta vantagens significativas se comparadas ao mecanismo *DQ* de *Rivers*, pois o tamanho de bloco utilizado em nossa estratégia possui tamanho variável e a distribuição dos dados é realizada a partir de uma visão global do sistema, o que não acontece em *Rivers*.

Em alguns cenários o *throughput* de um nó não é proporcional à taxa de comunicação com o nó central e, em situações extremas, o tamanho de bloco recomendado B_i pode ser muito pequeno. Nestes casos a freqüente troca de mensagem pode prejudicar a execução. Para corrigir este problema inserimos o parâmetro Δ na expressão que determina o tamanho de B_i e, e seu cálculo deve ser alvo de trabalhos futuros.