X Simpósio Brasileiro de Arquitetura de Computadores

TAMAGOSHI – Plataforma para Avaliação de Escalonamento de Tarefas em Programação em Lógica Paralela

Patrícia Kayser Vargas, Cláudio Fernando Resin Geyer
{kayser, geyer}@inf.ufrgs.br
CPGCC – UFRGS

Inês de Castro Dutra
ines@cos.ufrj.br
COPPE/Sistemas – UFRJ

Resumo

O escalonamento de tarefas é um dos pontos mais importantes de um sistema distribuído. Seu objetivo é determinar a atribuição de tarefas para elementos de processamento (nodos) e a ordem em que cada tarefa será executada de modo que algumas medidas de desempenho possam ser otimizadas. Nos sistemas de exploração de paralelismo na Programação em Lógica, o escalonamento normalmente apresenta muitas ligações com o modelo de execução, dificultando a comparação entre políticas distintas de escalonamento. Para auxiliar essa comparação, pode-se utilizar o simulador TAMAGOSHI no qual o programa em lógica é representado pela sua árvore de execução OU.

Abstract

Task scheduling is one of the most important parts of a distributed system. Its aim is to decide task attribution to processing elements and execution order of each task, in such way that some performance measures can be optimized. In parallel Logic Programming systems, scheduling is normally connected to the execution model, making the comparison between distinct scheduling policies very difficult. To help this kind of comparison, we can use the TAMAGOSHI simulator which represents a logic program through its OR execution tree.

1 Introdução

O objetivo do escalonamento de tarefas [2] é determinar a atribuição das tarefas para elementos de processamento e a ordem em que cada tarefa será executada, de modo que algumas medidas de desempenho sejam otimizadas. Permitindo a distribuição de carga de um elemento sobrecarregado para um com pouca ou nenhuma carga, o desempenho do sistema pode ser melhorado. Na exploração de paralelismo na Programação em Lógica (com Restrições), as políticas de escalonamento são importantes para garantir um bom desempenho do sistema, sendo normalmente muito relacionadas aos modelos de execução. No projeto APPELO (**A**mbientes de **P**rogramação **P**aralela em **L**ógica) existem dois modelos de paralelismo OU que utilizam políticas de escalonamento distintas: uma centralizada e outra distribuída. O TAMAGOSHI (***Task Simulator Program and OR-Scheduler Interface***) é uma ferramenta que simula a execução de um programa em lógica através da execução de árvores de execução OU [4] simplificadas permitindo a comparação entre essas e outras políticas.

O restante do texto apresenta-se organizado do seguinte modo: na seção 2 apresenta-se os modelos pclp(FD) e PloSys; na seção 3 o TAMAGOSHI é descrito; na seção 4 descreve-se os resultados da análise; e finalmente, na última seção, as conclusões e trabalhos futuros.

2 Escalonamento de Tarefas nos Modelos PloSys e pclp(FD)

O PloSys [5] e o pclp(FD) [7] modelam a exploração de paralelismo OU multi-seqüencial em máquinas de memória distribuída. Um sistema OU multi-seqüencial [4] é formado por n trabalhadores, cada um dos quais possui uma máquina abstrata completa capaz de resolver de forma seqüencial um programa em lógica. Desta forma, considera-se que um **trabalhador** é um elemento de processamento que possui uma **máquina abstrata** formada por um conjunto

de pilhas de execução que permite a resolução de um programa em lógica. Nas pilhas da máquina abstrata existem estruturas de dados denominadas **pontos de escolha** (PE) que armazenam a indicação da próxima alternativa (cláusula) possível na resolução de um predicado. A exportação de uma tarefa implica a exportação de PE e de **contexto de execução**, isto é, do conteúdo das **pilhas** da máquina abstrata no qual o PE está inserido.

A política de escalonamento é centralizada no PloSys e distribuída no pclp(FD). O controle da execução no PloSys é efetuado pelo escalonador principal, enquanto no pclp(FD) a responsabilidade é dividida entre os escalonadores, havendo um especialmente encarregado da detecção de terminação. Ambos os modelos utilizam o mecanismo de cópia de pilhas para a exportação de contexto, porém o pclp(FD) utiliza cópia incremental, que é uma otimização que permite copiar uma quantidade menor de dados a partir da detecção de porções comuns de pilhas entre trabalhadores. Para facilitar a compreensão dos algoritmos de escalonamento, apresenta-se a seguir as políticas que os compõem [6]:

1. **Transferência**: determina quando um trabalhador está em estado adequado para participar de uma transferência de tarefa, sendo usual empregar faixas de valores das unidades de carga (thresholds) para indicar estados. Ambos os modelos adotam os seguintes estados:
   - **ocioso** (*idle*): não possui tarefas, permanecendo neste estado até a importação;
   - **ocupado** (*busy*)*: não existem tarefas suficiente para exportar;
   - **sobrecarregado** (*overloaded*): possui tarefas excedentes que podem ser exportadas.

   Assim, um trabalhador permanece sobrecarregado até a diminuir sua carga pela execução local ou pela exportação, e mantém-se ocioso até receber uma tarefa por importação.
2. **Seleção**: determina qual tarefa deve ser transferida, que no caso desses modelos é sempre o PE mais antigo do trabalhador exportador, considerando a heurística de que tarefas de mais alta granulosidade encontram-se próximas da raiz da árvore de execução [4] e o fato de que, quanto mais velho for um PE, menor será o contexto a ser exportado.
3. **Informação**: decide quando coletar informações sobre o sistema. A propagação de informação no PloSys ocorre apenas na mudança de estado: toda transição é informada ao escalonador. No pclp(FD), cada trabalhador possui uma lista de ociosos que é atualizada em dois momentos: (a) ao tornar-se ocioso, o trabalhador avisa outros trabalhadores; (b) na exportação, a lista de ociosos é enviada com a tarefa.
4. **Localização**: determina o trabalhador mais adequado para receber uma tarefa. No PloSys, o escalonador mantém um estado aproximado dos trabalhadores, escolhendo para a troca de tarefas um par de trabalhadores formado por: (a) exportador: o ativo de maior carga; (b) importador: um inativo. No pclp(FD), o exportador localiza o importador através da consulta aos trabalhadores da sua lista de ociosos.

3 TAMAGOSHI – Task Simulator Program and OR-Scheduler Interface

O protótipo TAMAGOSHI foi proposto com o objetivo de realizar uma execução simplificada de um sistema OU multi-seqüencial: os trabalhadores foram substituídos por executores de programas sintéticos e o controle da execução fica a cargo de escalonadores completos. Em uma fase inicial do protótipo pclp(FD), o TAMAGOSHI simulava o protocolo de escalonamento com uma máquina abstrata com gerador de cargas aleatórias, não utilizando um padrão de execução de aplicações reais nem permitindo a repetição de execuções. Por isso, o protótipo TAMAGOSHI foi alterado para simular programas sintéticos, que obedecem ao padrão de execução de programas em lógica, através de árvores OU simplificadas,

* O PloSys denomina esse estado de *quiet*, possuindo a mesma semântica.

conforme será explicado a seguir. A versão do protótipo TAMAGOSHI empregada nesse trabalho implementa também o algoritmo de escalonamento do PloSys. Além disso, ele possui uma interface gráfica que permite observar o funcionamento do protocolo de escalonamento.

Para a definição de um programa sintético do TAMAGOSHI utiliza-se a noção de árvore de execução OU. Considere o exemplo da figura 1 que efetua uma consulta em uma base de conhecimento. A partir da execução da consulta `?- thing(X, blue, liquid)` forma-se uma árvore de execução como a representada. Note-se que apesar de existir mais de uma alternativa (cláusula) para o predicado `state`, não é gerado um PE pois em um sistema real, o compilador evitaria a criação desta estrutura, uma vez que existe apenas uma opção válida.

```
color(sky,blue).
color(sea,blue).
color(grass, green).

state(sky,gaseous).
state(sea,liquid).
state(grass,solid).

thing(T, C, S) :-
    color(T, C),
    state(T, S).

?-thing(X,blue,liquid).
```

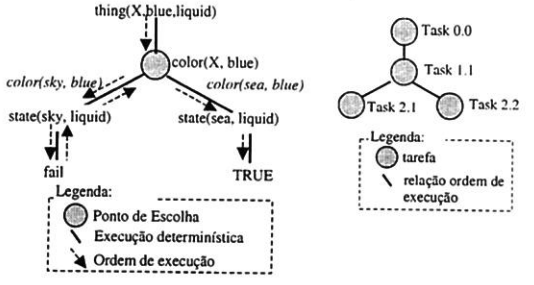

Figura 1 – Exemplo de programa, árvore de execução OU e árvore sintética.

O TAMAGOSHI simula a execução de programas a partir de códigos sintéticos formados por um conjunto de tarefas similares a PEs Prolog representando o padrão da árvore de execução. A figura 1 apresenta a árvore correspondente ao programa sintético do exemplo. Cada tarefa possui um tempo de execução determinística antes da criação da primeira tarefa e a lista de suas sub-tarefas. A figura abaixo apresenta as principais classes que compõem o TAMAGOSHI utilizando a linguagem UML (*Unified Modeling Language*).

Os métodos das classes abstratas *Scheduler* e *Engine* fornecem uma interface padronizada para a descrição de diferentes políticas de escalonamento. O simulador permite a utilização de diferentes códigos sintéticos (formado por tarefas) e realiza a coleta das estatísticas. A implementação foi efetuada na linguagem Java utilizando *threads*, o que permite a simulação de execução de paralelismo OU em uma máquina não paralela. Uma instância da classe *Scheduler* deve basicamente definir a política de localização e de informação, pois a definição da tarefa a ser exportada (seleção) e dos estados do trabalhador (transferência) é feita automaticamente.

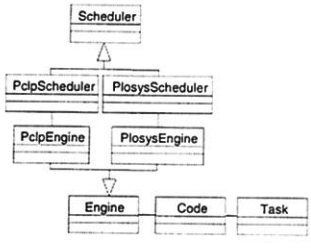

Figura 2 – Algumas classes do TAMAGOSHI.

4 Análise dos Resultados

Foram comparados o número total de mensagens utilizadas, o tempo total de execução e custos adicionais para efetuar o escalonamento. Apresenta-se aqui os resultados de forma qualitativa, uma vez que o resultado dessa comparação não é tão relevante quanto a possibilidade de efetuá-la em um ambiente de execução uniforme fornecido pelo TAMAGOSHI. Através da observação do comportamento do TAMAGOSHI, notou-se que ambos os escalonadores conseguem manter os trabalhadores ocupados a maior parte do

tempo. Há também a verificação de que, dependendo do tamanho da aplicação a ser executada, a inclusão de muitos trabalhadores não auxilia a execução, uma vez que vários trabalhadores ficam desocupados, degradando um pouco o desempenho do sistema devido ao aumento no número de mensagens. Como esperado, o escalonador centralizado utiliza um número menor de mensagens do que o distribuído. Para minimizar o número de mensagens adicionais, o pclp(FD) trabalha com uma incerteza quanto as informações de estado dos trabalhadores, através do uso da lista de ociosos, talvez por isso esse aumento no número de mensagens não seja expressivo. Notou-se que no pclp(FD) a lista de ociosos tende a se tornar desatualizada no decorrer do processamento: um trabalhador pode considerar que um ou mais escalonadores ocupados estão ociosos. Porém, em nenhuma das simulações, ocorreu a existência de trabalhadores sobrecarregados desconhecendo a existência de ociosos.

5 Conclusão

Em Prolog, vários sistemas para exploração de paralelismo têm sido propostos e implementados, mas no contexto de máquinas distribuídas ainda existem poucos trabalhos. O custo de exportação em ambientes de memória distribuída é bastante elevado, por isso o escalonador tem um papel essencial. Também existem poucos sistemas com escalonamento distribuído, devido a eficiência da gerência centralizada para um número pequeno de processos e a facilidade de implementação. No entanto, deve-se avaliar a importância do escalonamento distribuído para obtenção de escalabilidade e de tolerância a falhas. A políticas de escalonamento do PloSys e do pclp(FD) possuem duas vantagens principais: a simplicidade do protocolo, que facilita a implementação e entendimento, e a independência de topologia, que também auxilia a sua utilização.

Com relação à simulação do paralelismo, a abordagem apresentada não é totalmente original, existindo trabalhos semelhantes [3] [1]. Porém, diferente de outros trabalhos, o TAMAGOSHI permite simular em uma única máquina uma política de escalonamento com n trabalhadores, com vantagens didáticas e facilidades para a depuração de protocolos. Além disso, é uma plataforma independente de arquitetura. A existência das classes abstratas permite que diferentes políticas escalonamento sejam descritas sem a preocupação com o modelo de execução real da máquina abstrata.

6 Referências Bibliográficas

[1] COSTA, C.A. da. **Uma Proposta de Escalonamento Distribuído para Exploração do Paralelismo na Programação em Lógica**. CPGCC/UFRGS, 1998. Dissertação.

[2] EL-REWINI, H.; LEWIS, T.G.; ALI, H.H. **Task Scheduling in Parallel and Distributed Systems**. Prentice Hall, 1994. 290p.

[3] KANNAT, S.E., et al. A Platform to Study Load Balancing Functions for Parallel Logic Systems. **Proceedings...** INTERNATIONAL WORKSHOP ON PARALLEL PROCESSING, 1., 1994, Bangalore, India.

[4] KERGOMMEAUX, J. C.; CODOGNET, P. **Parallel Logic Systems**. Grenoble, France: Institut IMAG, 1992. Technical Report.

[5] MOREL, É.; et al. Cuts and Side-effects in Distributed Memory OR-Parallel Prolog. **Parallel Computing**, v. 22, p.1883-1896, February 1997.

[6] SINGHAL, M.; SHIVARATRI, N. G. **Advanced Concepts in Operating Systems: Distributed, Database, and Multiprocessor Operating Systems**. MIT Press, 1994. 522p.

[7] VARGAS, P.K.; GEYER, C.F.R.. Introduzindo o Paralelismo OU na Programação em Lógica com Restrições. **Anais...** SBAC-PAD, 9., 1997, Campos do Jordão, SP. p.381-396.