Tolerância a falhas em um ambiente de execução paralela

Helena de Lima Braga, Gerson G. H. Cavalheiro

Universidade do Vale do Rio dos Sinos
Avenida Unisinos, 950 – São Leopoldo - RS
helena_lb@uol.com.br, gersonc@exatas.unisinos.br

Introdução

Atualmente as mais diversas áreas exigem alto poder computacional. Exemplos desta demanda são problemas em áreas como meteorologia, medicina, biociências e mesmo muitos problemas comerciais. Estas tarefas não podem ser realizadas utilizando-se os computadores tradicionais disponíveis nos dias de hoje por falta de poder computacional dos mesmos. Esta situação tem motivado o crescimento acelerado de máquinas paralelas e de técnicas que viabilizem o processamento de alto desempenho [LEM 02]. Neste contexto tem sido cada vez mais comum a exploração de aglomerados de computadores (*clusters*) e arquiteturas multiprocessadoras com memória compartilhada (*Symmetric Multi-processors*). No entanto, o entendimento destas arquiteturas não é trivial, tendo sido criado para isto uma série de ambientes de execução e técnicas de programação para auxiliar o programador nesta tarefa [CAV 03].
É de se observar que muitas aplicações mesmo dispondo de múltiplos processadores podem permanecer processando durante horas, dias, ou até mesmo, semanas para que resultados possam ser obtidos. Diversos fatores podem vir a interromper o processamento repentinamente, como, por exemplo, falta de energia. É de grande interesse que o processamento realizado não seja perdido ao longo da execução do programa. Para isso, faz-se necessário um mecanismo de tolerância a falhas nestes ambientes de execução, para garantir que o recomeço da aplicação seja viabilizado com a menor perda possível de processamento.
O objetivo deste artigo é propor um mecanismo de tolerância a falhas em um ambiente multithread utilizando técnicas de checkpoints [DIE 99, DIE 99b, PLA 95, WAN 95]. O ambiente a ser utilizado chama-se Anahy e será explicado com mais detalhes na próxima seção.

Ambiente Anahy

A implementação de Anahy torna viável a exploração do processamento de alto desempenho sobre arquiteturas do tipo aglomerado de computadores, onde cada nó pode vir a ser um multiprocessador com memória compartilhada [CAV 03]. Anahy tem um modelo de programação que tem por objetivo retirar do programador qualquer preocupação relacionada ao mapeamento da concorrência da aplicação com o paralelismo da arquitetura real em cima da qual o programa será executado [REA 03]. O programador tem a visão de uma arquitetura virtual multiprocessada dotada de memória compartilhada. Essa arquitetura pode ser visto na Figura 1. A arquitetura real é composta por um conjunto de nodos de processamento, dotados de memória local e de

unidades de processamento (CPU's). A arquitetura virtual é composta por um conjunto de processadores virtuais (PVs) alocados sobre os nodos e por uma memória compartilhada pelos PVs. Cada PV conta ainda com um espaço de memória próprio, utilizado para armazenar dados locais à atividade que está sendo executada. Enquanto um PV estiver executando a tarefa que lhe foi atribuída, nenhuma outra sinalização será tratada por ele. A comunicação entre os PVs se dá através de memória compartilhada, acessada pelas instruções introduzidas pela arquitetura virtual [CAV 03].

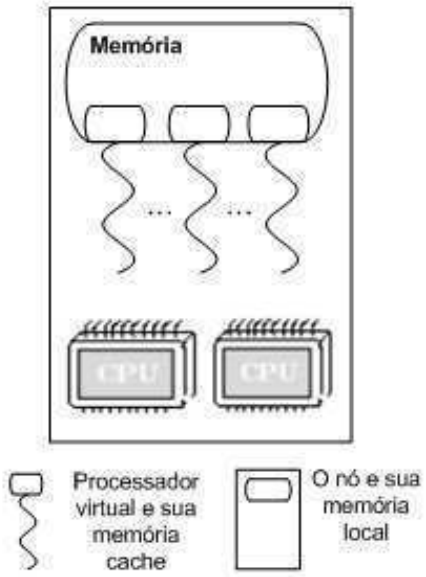

Figura 1: Modelo lógico da arquitetura de suporte à Anahy

O ambiente Anahy é estruturado em três camadas: Interface Aplicativa Anahy (API Anahy), Núcleo Executivo e Maquina Abstrata [REA 03, REA 02].
Interface Aplicativa Interface para desenvolvimento de programas. Permite a descrição da concorrência de uma aplicação sob a forma de atividades concorrentes, denominadas tarefas, capazes de se comunicar via parâmetros de entrada e retorno de resultados.
Núcleo Executivo Visa retirar do programador a preocupação de explorar o paralelismo real da arquitetura. Implementa o mecanismo de escalonamento de tarefas, sendo dotado de recursos de balanceamento de carga.
Máquina Abstrata Utiliza recursos oferecidos por threads POSIX e MPI. É construído um agregado virtual, onde o papel do processador é exercido pelas *threads* e o compartilhamento de dados é obtido através da biblioteca de comunicação. Estas unidades processadoras representadas pelas threads são também chamadas de processadores virtuais.
O Núcleo Executivo de Anahy gerencia quatro listas de tarefas: a primeira contém as tarefas prontas (aptas a serem executadas), a segunda, as tarefas terminadas, cujos resultados ainda não foram solicitados (a operação de *join* sobre estas tarefas ainda não foi realizada). A terceira e a quarta lista contém tarefas bloqueadas e desbloqueadas, respectivamente.

Checkpoints

Um checkpoint é uma “foto” do estado dos processos. Esta imagem armazena informações suficientes para restaurar estes processos em caso de colapso [USM 96]. Em uma aplicação multithread, através da utilização de checkpoints é possível obter uma imagem pontual do estado de execução das threads em um intervalo de tempo pré-determinado ou na ocorrência de uma operação específica. Com essa imagem do processamento armazenada, o tempo de perda em caso de pane é limitado ao último checkpoint realizado, que será restaurado e dará seguimento ao processamento da aplicação [USM 96].

No momento em que um checkpoint for ser obtido é importante garantir que as relações causais existentes entre as operações que compõem uma thread sejam respeitadas. No ambiente em questão, as tarefas executadas nos processadores virtuais têm a característica de serem pequenas. Assim , em caso de colapso a tarefa será recomeçada desde o princípio, evitando desta forma qualquer possível problema em relação às dependências existentes entre operações da tarefa.

Como exemplos de bibliotecas que realizam checkpoints, pode-se citar *libckpt* [PLA 95], *libckp* [WAN 95], *libtckpt* [DIE 99, DIE 99b] entre outras. A primeira biblioteca citada, *libckpt* foi modelada para realizar checkpoints de forma transparente ao usuário em sistemas monoprocessados. Esta biblioteca realiza um checkpoint da aplicação a cada 10 minutos. Em caso de colapso a aplicação deve ser reiniciada setando o flag *recover* para que o checkpoint seja restaurado e a aplicação reiniciada. O fato desta biblioteca não suportar aplicações multithread impossibilita a sua utilização em conjunto com o ambiente Anahy. A ferramenta *libckp*, também possui características interessantes como a divisão do programa em estado consistente e estado volátil. Estes dois conceitos são considerados como sendo a chave para tornar a utilização de checkpoints um mecanismo poderoso. É feita uma divisão sobre o que é importante para a continuação da aplicação em caso de falha e o que é considerado característica do ambiente e, que desta forma, não estará salva no checkpoint. No entanto, *libckp* também não fornece suporte a aplicações multithread, e por esse motivo não será utilizada juntamente com Anahy.

A última biblioteca citada, *libtckpt*, possui características mais próximas da realidade do ambiente Anahy. Diferente das demais, *libtckpt* implementa o mecanismo de checkpoints em aplicações com múltiplas threads. Esta biblioteca divide o estado de um programa multithread em área privada e área compartilhada. A área privada de uma thread inclui o PC (*Program Counter*), o ponteiro para a pilha e os registradores. Já a área compartilhada inclui tudo que é comum a todas as *threads* no processo, como o espaço de endereçamento, por exemplo. A biblioteca chkpt salva estas duas áreas no momento em que um checkpoint é realizado.

Tolerância a falhas em Anahy

A necessidade do ambiente Anahy em relação aos mecanismos de tolerância a falhas citados anteriormente, é mais restrita. No Anahy as tarefas são organizadas em filas, bastando assim garantir a integridade destas para que a aplicação possa ser recuperada. O consumo destas filas descreve o estado da aplicação, informando as tarefas já realizadas, prontas para execução, bloqueadas e não-bloqueadas. É de suma importância

manter o estado destas filas, já que elas serão responsáveis pela recuperação do estado do sistema em caso de falha. Não há necessidade de utilização de métodos mais onerosos de checkpoints visto que as tarefas Anahy são sempre pequenas. Isso é um detalhe importante visto que a recuperação em caso de colapso será de toda a tarefa. Esta decisão não impactará no desempenho do mecanismo proposto visto que, caso faça-se necessário restaurar um checkpoint, a posição salva terá como recuperar a fila de tarefas prontas para serem executadas e as demais filas do sistema. E desta forma, a perda em função do colapso será das tarefas que estavam em execução no momento da pane.

Conclusão

O número de aplicações que demandam muito tempo de execução é bastante grande e Anahy é uma ferramenta que tem como objetivo suportar esta classe de aplicações. O mecanismo de tolerância a falhas proposto neste trabalho é invisível ao usuário, ou seja, não requer qualquer intervenção do programador para que esteja ativo. Além disso, possui baixo custo computacional para a sua manutenção e, mesmo sendo específico para o ambiente Anahy, não limita a capacidade de descrição de aplicações.

Referências

[CAV 03] CAVALHEIRO, G. G. H. REAL, L. C. V. DALL'AGNOL, E. C. Uma Biblioteca de Processos Leves para a Implementação de Aplicações Altamente Paralelas. WSCAD 2003, SBC. São Paulo, 2003.

[DIE 99] DIETER, W. R. LUMPP, J. E. A User-level Checkpointing Library for POSIX Threads Programs. Department of Electrical Engineering, University of Kentucky, 1999.

[DIE 99b] DIETER, W. R. LUMPP, J. E. User-level Checkpointing of POSIX Threads. Department of Electral Engineering, University of Kentucky, 1999.

[LEM 02] LEMKE, N. Aplicações de Alto Desempenho Trivialmente Paralelizáveis. ERAD: Escola Regional de Alto Desempenho, São Leopoldo, 2002.

[PLA 95] PLANK, J. S. et al. Libckpt: Transparent Checkpointng under Unix. Department of Computer Science, University of Tennessee, 1995.

[REA 02] REAL, L. C. V. et al. Construção de um Ambiente de Programação para o Processamento de Alto Desempenho. ERAD: Escola Regional de Alto Desempenho. São Leopoldo, 2002.

[REA 03] REAL, L. C. V. CAVALHEIRO, G. G. H. Avaliação de Desempenho de um Ambiente de Programação Paralelo. ERAD: Escola Regional de Alto Desempenho, Santa Maria, 2003.

[USM 96] USMANI, K. H. A Linux Checkpoint and Rollback Facility. Department of Computer Science, College of William and Mary, 1996.

[WAN 95] WANG, Y. et al. Checkpointing and Its Applications. IEEE FCTS: Fault Tolerance Computing Symposium, 1995.