Suporte a Tolerância a Falhas no Ambiente de Programação DPC++[1]

Maurício Lima Pilla[2] **Marcos Ennes Barreto**[3]
Rafael R. dos Santos[4] **Gerson G. H. Cavalheiro**[5]
Philippe O. A. Navaux[6]

Instituto de Informática
Universidade Federal do Rio Grande do Sul
Caixa Postal 15064, Porto Alegre, RS

Resumo

Este trabalho apresenta o suporte a Tolerância a Falhas para o ambiente de programação DPC++. O mecanismo que está sendo implementado atualmente é constituído por um algoritmo de criação e recuperação de *checkpoints*, o qual permite a um programa realizar recuperação automática de falhas de um objeto distribuído, aumentando a confiabilidade da aplicação. As aplicações DPC++ são geradas através do pré-compilador DPC++, o qual possibilita que aplicações utilizem o mecanismo de tolerância a falhas de modo transparente ao usuário.

Abstract

This paper presents Fault Tolerance support for DPC++ programming environment. The mechanism, that has been implemented, is constituted by an algorithm of creation and recovery of checkpoints, which allows a program to recover automatically from distributed object faults, increasing the reliability of applications. The DPC++ programs are generated by the precompiler, making possible the use of the fault tolerance mechanism in a transparent way to the user.

[1] Projeto financiado pelo CNPq e FINEP através do Programa PAD
[2] Bolsista IC-CNPq, bacharelando, Instituto de Informática, UFRGS, email:pilla@inf.ufrgs.br
[3] Mestrando, CPGCC, UFRGS, email:barreto@inf.ufrgs.br
[4] Doutorando, CPGCC, UFRGS, email rrsantos@inf.ufrgs.br
[5] Doutorando INPG-IMAG-França, email: Gerson.Cavalheiro@imag.fr
[6] Professor Doutor (Grenoble, 1979), Instituto de Informática, UFRGS, email.navaux@inf.ufrgs.br

1 Introdução

Em aplicações científicas e comerciais, a computação de um programa que foi interrompido por uma falha tem de ser feita desde o seu início novamente. Como resultado, as aplicações são terminadas apenas se o tempo entre falhas é menor que o tempo de execução do programa [ZOM96]. Foi demonstrado em outros estudos que o tempo médio de execução de um programa na presença de falhas cresce exponencialmente com o tamanho do programa.

Observando-se esta realidade, pode-se notar que a inclusão de um mecanismo de tolerância a falhas que permita a recuperação da computação já realizada, em caso de falha, é desejável em sistemas distribuídos. O processamento distribuído, com nodos possuindo memória e processadores próprios, interligados por uma rede de conexão, gera situações onde há redundância e esta pode ser utilizada para aumentar a confiabilidade dos sistemas distribuídos. Por exemplo, os processos que estejam executando em um nodo que falhe podem ser recuperados em outro nodo, desde que haja informação suficiente para tanto.

Uma abordagem possível é o uso de técnicas de *recuperação de erros por retrocesso* [SIN94]. *Checkpoints* (ou pontos de recuperação) são obtidos através da gravação do estado de processos em meio estável, normalmente em disco rígido. Do estado de um processo constam o valor das variáveis, seu ambiente, informações de controle, valor dos registradores, pilha, etc. Com a utilização de *checkpoints*, o tempo médio de execução de um programa passa a crescer linearmente com o tamanho do programa [ZOM96].

Como os processos interagem em ambientes distribuídos através de troca de mensagens, e a rede de comunicação possui um atraso considerável em relação a acessos à memória, o estado dos canais de comunicação passa a ser um importante fator a ser considerado na introdução de mecanismos de tolerância a falhas em sistemas distribuídos. A criação de um *checkpoint* global implica em haver uma sincronização global de processos, de modo que este *checkpoint* reflita um estado consistente em um determinado instante de tempo. Esta sincronização dos nodos componentes do sistema distribuído é extremamente complexa e cara. Uma solução possível, descrita em Singhal [SIN94], seria o estabelecimento de *checkpoints* assíncronos, não considerando a troca de mensagens entre processos, porém este método não garante que o estado das comunicações seja refletido corretamente em caso de *rollback* (recuperação).

Outro método para a criação de *checkpoints* consiste em manter um conjunto consistente de *checkpoints* [SIN94]. Cada processo tomaria seu *checkpoint* após cada envio de mensagem. Desta forma, o conjunto dos *checkpoints* mais recentes está sempre consistente. Porém, este método pode resultar em mensagens órfãs e, conseqüentemente, em um estado inconsistente do sistema. Além disto, este esquema necessita que a operação de envio ou recebimento de uma mensagem e a criação de *checkpoints* constituam uma operação atômica, algo difícil de ser implementado.

Portanto, fica clara a necessidade de um método de criação de *checkpoints* distribuídos que garanta a consistência do estado global do sistema, gerando o menor *overhead* possível para tanto.

O presente artigo aborda na seção 2 *a linguagem de programação DPC++ e o modelo distribuído*. A seção 3 consiste do *modelo de checkpoints* desenvolvido para o DPC++. A seção 4 apresenta uma *conclusão* sobre o artigo.

2 A Linguagem DPC++ e o Modelo Distribuído

Processamento Distribuído em C++ (DPC++) é uma linguagem para programação distribuída orientada a objetos baseada em C++. É uma linguagem de propósito geral, que dispõe de recursos que visam facilitar a programação de grandes sistemas distribuídos.

As características de orientação a objetos de DPC++ são as herdadas do C++. Inclusive a sintaxe dos programas é a mesma. Porém, em DPC++, foi introduzido um novo tipo de classe: a classe dos objetos distribuídos.

A simplicidade da escrita de aplicações distribuídas é provida pelo pré-processador DPC++. O usuário não precisa preocupar-se com detalhes de comunicação entre objetos distribuídos, pois o pré-processador encarrega-se de gerar o código que será submetido ao compilador C++.

Os conceitos básicos de orientação a objetos são aplicados ao modelo DPC++, onde existem objetos que encapsulam todas as suas propriedades: dados e métodos. A execução de programas é realizada através da invocação de métodos dos objetos, através do envio de mensagens. Ainda, no modelo DPC++ é explorada a concorrência de execução entre objetos, utilizando-os em sistemas distribuídos.

A linguagem DPC++ utiliza como modelo base de objetos distribuídos a execução da função destes em uma rede de processadores homogêneos, onde cada nodo pode suportar n objetos distribuídos executando, sendo este número limitado apenas pela capacidade de memória localmente disponível. Um escalonador é responsável por compartilhar o uso do processador entre os objetos.

O *Diretório* é um objeto distribuído especializado que possui tabelas com informações sobre todos os demais objetos distribuídos, tais como nodo em que está alocado, endereço de comunicação, identificador do processo e tipo do objeto. O Diretório provê serviços ligados à manutenção, como criação de novos objetos distribuídos, pesquisa sobre informações destes objetos e verificação de falha e restauração de objetos distribuídos.

Em Cavalheiro [CAV93], são encontradas outras considerações e restrições da linguagem DPC++.

3 Modelo de Checkpoints

O modelo de criação e recuperação de *checkpoints* desenvolvido para o ambiente de programação DPC++ é baseado na criação de *checkpoints* após cada troca de mensagens entre objetos distribuídos, com um protocolo para garantir que os *checkpoints* foram criados e que não há perda de mensagens e em um mecanismo de *timeout*, o qual detecta a ocorrência de erros.

Considera-se que o meio de transmissão é confiável o suficiente para garantir que todas as mensagens enviadas pelo nodo origem são corretamente recebidas pelo nodo destino [MID90]. O problema de confiabilidade do meio de transmissão não é considerado inicialmente. Falhas do objeto Diretório também não são consideradas neste momento, sendo que um outro mecanismo de tolerância a falhas está sendo desenvolvido para estes casos.

A criação de *checkpoints* foi implementada através de uma versão modificada da *libckpt* [PLA95], na qual foi adicionada a capacidade de criação de múltiplos *checkpoints* para um mesmo usuário, através do acréscimo de uma extensão que diferencia os mesmos.

O algoritmo responsável pela consistência do estado global resultante da criação dos *checkpoints* e pela eventual recuperação de objetos distribuídos que falham está descrito em Pilla [PIL97].

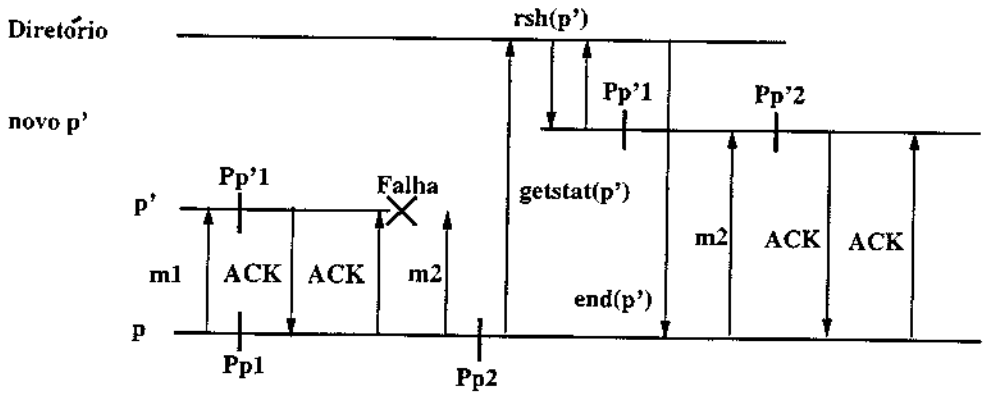

Figura 1 Fluxo de mensagens entre dois objetos distribuídos

A figura 1 ilustra o fluxo de mensagens entre dois objetos, p e p'. Inicialmente, o objeto distribuído p envia uma mensagem *m1* ao objeto distribuído p' e ambos criam seus *checkpoints*. Depois, o objeto p' envia uma mensagem de confirmação (ACK) ao objeto p, que então envia uma mensagem de confirmação para p' e prossegue normalmente.

Após enviar a mensagem de confirmação para p, p' falha. Quando p tenta enviar uma mensagem *m2* para p', cria seu *checkpoint* e aguarda pela confirmação de p'. Como p' não responde em um determinado *timeout*, p envia uma mensagem ao Diretório, pedindo o *status* do objeto p'. O Diretório descobre que p' falhou e dispara um novo objeto da mesma classe (*rsh(p')*). Este novo objeto utiliza as informações armazenadas no último estado armazenado do objeto que falhou para restaurar seu contexto e continuar a execução.

O Diretório devolve para p o novo endereço de p'. O objeto p envia novamente a mensagem, sem, contudo, criar um novo *checkpoint*, e espera pela confirmação. p' envia a confirmação e espera pela confirmação de p. Depois disto, ambos os objetos prosseguem normalmente.

O algoritmo de criação de *checkpoints* apresentado por Koo [KOO87] exige um número maior de mensagens, pois quando um processo deseja criar um *checkpoint* envia mensagens a todos os processos de quem recebeu mensagens desde a criação de seu último *checkpoint*. Estes processos, por sua vez, executam o mesmo procedimento, fazendo com que o número de mensagens seja bastante grande em função do número de processos (objetos). O outro problema associado à criação de *checkpoints* deste modo é que um grande número de processos criam *checkpoints* ao mesmo tempo, o que pode criar uma carga muito elevada para o sistema em certos momentos.

Atualmente, estão sendo desenvolvidas as bibliotecas que permitirão a incoporação do mecanismo de tolerância a falhas proposto ao pré-processador DPC++, através de um segundo protótipo do mecanismo. Com isto, será possível desenvolver aplicações distribuídas em DPC++ incluindo de forma transparente ao programador o mecanismo de tolerância a falhas.

4 Conclusão

O mecanismo apresentado permitirá que programadores possuam uma ferramenta de fácil programação com uma confiabilidade maior, decorrente do uso do mecanismo de tolerância a falhas. Como foi mostrado em Pilla [PIL97], o mecanismo suporta falhas de um objeto distribuído por vez.

Uma característica interessante do modelo adotado é que objetos que não tenham falhado não necessitam serem recuperados para um estado anterior, o que poupa o sistema deste *overhead*. Esta característica é alcançada através da utilização das particularidades dos objetos distribuídos DPC++, os quais somente se comunicam com outros objetos nos momentos de invocação de métodos e de resposta dos mesmos. Outros modelos de criação de *checkpoints* não garantem que apenas o processo que falha necessite fazer o *rollback*, sendo que em alguns casos pode ocorrer o chamado *efeito dominó*. No *efeito dominó*, um processo que falha força outro processo a voltar ao seu último *checkpoint*, devido ao estado inconsistente das comunicações. Este processo, por sua vez, faz com que outro objeto retorne ao seu último *checkpoint*, e assim sucessivamente.

Outra característica importante é a inexistência de um coordenador centralizado para a criação dos *checkpoints*, sendo que cada par de objetos distribuídos que se comunicam criam seus *checkpoints* sem afetar os demais objetos.

A implementação deste mecanismo será complementada com outros mecanismos de tolerância a falhas, tais como um mecanismo para o objeto Diretório.

Referências

[BEL93] BELMONTE, Valdir Rossi, WEBER, Raul Fernando. *Gerindo Tolerância a Falhas em Sistemas Distribuídos.* São José dos Campos: V Simpósio de Computadores Tolerantes a Falhas, *anais...*, outubro, 1993.

[CAV93] CAVALHEIRO, Gerson G. H., NAVAUX, P. O. A.. *DPC++: Uma Linguagem para Processamento Distribuído.* Florianópolis: V SBAC-PAD, *anais...*, outubro, 1993.

[CAV94] CAVALHEIRO, Gerson G. H., SANTOS, Rafael R., NAVAUX, Philippe O. A.. *Análise de Desempenho de um Protótipo da Linguagem DPC++.* Caxambu : XXI SEMISH, *anais...*, 1994.

[JAL94] JALOTE, Pankaj. *Fault Tolerance in Distributed Systems.* P T R Pretince Hall. 1994.

[KAN97] KANELLAKIS, Paris C.; SHVARTSMAN, Alex A. *Fault-Tolerant Parallel Computation.* Kluwer Academic Publishers. 1997.

[KOO87] KOO, Richard, TOUEG, Sam. *Checkpointing and Rollback-Recovery for Distributed Systems.* IEEE Transactions on Software Engineering, vol. SE-13, no.1, January, 1987.

[MID90] MIDKIFF, S. F. e VAIDYANATHAN, P. *Performance evalutation of communication protocols for distributed processing. Computer Communications*, 13:(5) junho 1990.

[PIL97] PILLA, M. L.; BARRETO, M. E.; SANTOS, R. R.; CAVALHEIRO, G. G. H.; NAVAUX, P. O. A. *Mecanismo de Tolerância a Falhas para a Linguagem de Programação DPC++*. IX SBAC-PAD. Campos do Jordão. SP:SBC, 1997.

[PLA95] PLANK, James S. et alli. *Libckpt: Transparent Checkpointing under Unix*. USENIX Winter 95 Technical Conference. 1995.

[SAN93] SANTOS, R. R.; CAVALHEIRO, G. G. H.; NAVAUX, P. O. A. *Mecanismo de Transporte para Comunicação entre Objetos Distribuídos*. Simpósio Nacional de Redes de Computadores e suas Aplicações. Porto Alegre. SUCESU-RS: 1993.

[SIN94] SINGHAL, Mukesh; SHIVARATRI, Niranjan G. *Advanced Concepts in Operating Systems*. McGraw-Hill. 1994.

[YAU92] YAU, S. S., JIA, X., BAE, D. H.. *Software Design Methods for Distributed Computing Systems*. Computer Communications, 15(4):213-224, May, 1992.

[ZOM96] ZOMAYA, Albert Y.H. *Parallel and Distributed Computing Handbook*. McGraw Hill. 1996.