UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
Departamento de Ciência da Computação

MAC5701 - Tópicos em Ciência da Computação

# *Checkpointing* e Recuperação por Retrocesso de Aplicações Paralelas no InteGrade.

Raphael Yokoingawa de Camargo

São Paulo, 8 de abril de 2003

# 1 Introdução

Sistemas distribuídos estão cada vez mais presentes em nossa sociedade. Um exemplo recente é o surgimento das grades computacionais que permitem o compartilhamento de recursos e conseqüentemente a possibilidade de utilização de recursos computacionais que de outro modo ficariam ociosos. Globus, Condor e InteGrade [GKG+04] são exemplos destes sistemas de grades computacionais.

Para que estas grades computacionais sejam utilizadas para pesquisa científica, é necessário que bibliotecas de programação sejam integradas a estas grades. No caso do InteGrade foi implementado parte de uma biblioteca para o modelo de programação paralela BSP [HMS+98, Val90]. Este modelo é baseado no conceito de superpassos, onde a comunicação entre processos ocorrida em superpasso só fica disponível para estes processos no superpasso seguinte. Isto é, não existe troca de informações dentro de um mesmo superpasso.

Mas apesar das grades computacionais trazerem um grande ganho de poder computacional, surgem alguns problemas, como a maior propensão a falhas. No caso do InteGrade os nós que serão utilizados nas computações são tipicamente estações de trabalho. Estas podem sofrer diversos tipos de problemas, como travamentos, *reboots* e desconexões da rede. No caso de aplicações paralelas que possuem diversos processos sendo executados em diferentes máquinas este problema é ainda maior, dado que uma falha em qualquer um dos nós pode causar a falha da aplicação inteira. Mecanismos de tolerância a falha, como a recuperação por retrocesso [EAWJ02], visam resolver este tipo de problema.

## 1.1 Recuperação por Retrocesso

A **recuperação por retrocesso baseada em *checkpoints*** é o processo de reiniciar a execução da aplicação a partir de um estado salvo durante sua execução após a ocorrência de falhas em um ou mais de seus processos. Estes estados salvos são denominados *checkpoints*, e permitem que no caso de falha, a aplicação não precise retornar a seu estado inicial.

Existem diversas abordagens para salvar o estado de uma aplicação em execução. A mais comum, denominada *system level checkpointing*, salva todos os dados dos endereços de memória utilizados pela aplicação, incluindo a pilha de execução e os dados do *heap*. Este processo pode ser executado tanto no nível do usuário quanto do *kernel*. Um exemplo de uma biblioteca deste tipo é a libckpt [PaGKL95]. A outra técnica, denominada *application-*

*level checkpointing* consiste em adicionar o código para salvar o estado da aplicação e fazer sua reinicialização na própria aplicação. Esta abordagem é a que fornece a maior portabilidade [BMPS03, SR96].

No caso de aplicações paralelas e distribuídas existe um problema extra que é a dependência entre os diversos processos que estão sendo executados simultaneamente. Esta dependência é gerada pela troca de informação entre processos. Um exemplo claro disto é no envio de uma mensagem de um processo A para um processo B. Após o processo B receber a mensagem enviada por A, passa a haver uma dependência entre ambos, pois para que o processo B tenha recebido a mensagem, é necessário que o processo A a tenha enviado.

Deste modo, não basta que cada processo simplesmente salve seu estado periodicamente de maneira independente, pois se escolhermos um conjunto de *checkpoints* contendo um *checkpoint* de cada processo, este pode não formar um estado global consistente. No pior caso, é possível que a aplicação precise retornar a seu estado inicial, mesmo tendo salvo diversos *checkpoints*.

Existem diversas maneiras de se resolver este problema. O protocolo de *checkpointing* coordenado exige que todos os processos salvem seus estados de maneira coordenada, de modo que todos os estados globais gerados sejam consistentes. Isto garante que a aplicação sempre será reiniciada a partir do último *checkpoint* salvo, além de facilitar outros aspectos importantes como a coleta de *checkpoints* salvos obsoletos.

## 2 Plano de Trabalho

- Estudar as diversas técnicas que permitem o salvar o estado de uma aplicação e sua posterior reinicialização. Entre estas técnicas estão o *checkpointing* no nível do sistema, que por sua vez pode ser implementado em modo *kernel* ou usuário, e *checkpointing* no nível da aplicação.

- Implementar um sistema de recuperação por retrocesso baseado em *checkpoints* para a biblioteca BSPLib do InteGrade. Isto inclui:

  - Adicionar à BSPLib um sistema que realiza a coordenação entre os processos de modo que um estado global consistente seja formado.

– Criação de um sistema de detecção de falhas nos processos de modo que, em caso de falha, a aplicação possa ser reiniciada a partir do último *checkpoint* global salvo.

– Implementar um pre-compilador que analisa o código fonte de uma aplicação em C e/ou C++ e o modifica de modo a automatizar o processo de salvar o estado da aplicação e sua reinicialização.

## Referências


[BMPS03] Greg Bronevetsky, Daniel Marques, Keshav Pingali, and Paul Stodghill. Automated application-level checkpointing of mpi programs. In *Proceedings of the 9th ACM SIGPLAN PPoPP*, pages 84–89, San Diego, USA, 2003.

[EAWJ02] Mootaz Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, 34(3):375–408, May 2002.

[GKG+04] Andrei Goldchleger, Fabio Kon, Alfredo Goldman, Marcelo Finger, and Germano Capistrano Bezerra. InteGrade: Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines. *Concurrency and Computation: Practice and Experience*, 2004. Accepted for publication on March/April 2004.

[HMS+98] Jonathan M. D. Hill, Bill McColl, Dan C. Stefanescu, Mark W. Goudreau, Kevin Lang, Satish B. Rao, Torsten Suel, Thanasis Tsantilas, and Rob H. Bisseling. BSPlib: The BSP programming library. *Parallel Computing*, 24(14):1947–1980, 1998.

[PaGKL95] J. S. Plank, M. Beck amd G. Kingsley, and K. Li. Libckpt: Transparent chieckpointing under unix. In *Proceedings of the USENIX Winter 1995 Technical Conference*, pages 213–323, 1995.

[SR96] Volker Strumpen and Balkrishna Ramkumar. Portable checkpointing and recovery in heterogeneous environments. Technical Report UI-ECE TR-96.6.1, University of Iowa, June 1996.

[Val90] L.G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.