Mecanismos de *Checkpoint/Restart* para Migração de Processos MPI

Marcelo Veiga Neves, Tiarajú Asmuz Diverio, Nicolas Maillard

Laboratório de Sistemas de Computação - LSC
Instituto de Informática - Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil
{mvneves, diverio, nmaillard}@inf.ufrgs.br

Resumo

Nos últimos anos, MPI (Message Passing Interface) tornou-se um padrão para comunicação em clusters. No entanto, a norma MPI não prevê checkpoint ou migração de processos para permitir balanceamento de cargas, tolerância a falhas, etc. Existem diversas iniciativas que acrescentam esta funcionalidade ao MPI. No entanto, a maior parte delas apresenta restrições que limitam sua utilização prática. Neste contexto, este artigo faz um levantamento dos principais mecanismos de checkpoint/restart disponíveis para migração de processos MPI e analisa as possibilidades de utilização prática.

1. Introdução

O processamento paralelo e distribuído é freqüentemente utilizado para resolver problemas que demandam grande poder computacional. Através da paralelização de aplicações, pode-se obter aumentos significativos de desempenho, realizando-se operações mais rapidamente ou processando-se maiores volumes de dados. Existem diversas arquiteturas que permitem executar programas paralelos e distribuídos. Um exemplo são os *clusters* de computadores, que oferecem uma boa relação custo-desempenho e se tornaram bastante populares.

Ao longo do tempo, surgiram várias abordagens e ferramentas para a programação paralela e distribuída, sendo que, nos últimos anos, o uso de MPI (*Message Passing Interface*) tornou-se um padrão para comunicação em *clusters*. No entanto, a norma MPI não prevê *checkpoint* ou migração de processos para permitir balanceamento de cargas, tolerância a falhas, etc.

Migração de processos, no contexto de MPI, também pode ser utilizada para aumentar o desempenho de aplicações através da redução de comunicação. O objetivo é agrupar processos que trocam muitas informações em uma mesma máquina e, desta forma, reduzir o volume de dados trafegados pela rede.

Existem diversas iniciativas que acrescentam a funcionalidade de *checkpoint* ao MPI [14, 7, 3, 4]. No entanto, a maior parte delas apresenta restrições técnicas que limitam sua utilização prática. Neste contexto, o presente trabalho tem como objetivo fazer um levantamento dos mecanismos *checkpoint* disponíveis para migração de processos MPI e analisar as possibilidades de utilização prática.

O texto está organizado da seguinte maneira: primeiramente, é apresentada a problemática da migração de processos em aplicações MPI (seção 2); na seqüencia, são feitas algumas considerações sobre mecanismos de *checkpoint/restart* e as principais ferramentas disponíveis são apresentadas (seção 3); a segunda parte do artigo escolhe o suporte a *checkpoint* da implementação LAM/MPI para ser descrito em maiores detalhes (seção 4); por fim, são feitas algumas considerações finais e sugestões para trabalhos futuros.

2. Migração de Processos MPI

A migração de processos em uma aplicação MPI é mais difícil que em processos individuais, porque processos MPI são mais fortemente acoplados [5]. Se um processo MPI for migrado enquanto a aplicação estiver em execução, todos os outros processos precisam ser informados de sua nova localização. Além disso, se existirem mensagens pendentes durante a migração, há a possibilidade da aplicação falhar.

A transferência de um processo de uma máquina origem para uma máquina destino implica no salvamento do contexto da execução, alteração da localização do processo e reinicializarão da execução. Entende-se por contexto de execução, as informações relativas ao processo, tais como espaço de endereçamento, conteúdo de registradores, descritores de arquivos, etc. Este procedimento é possível com o auxílio de bibliotecas de *checkpoint/restart*. A seção a se-

guir faz algumas considerações sobre *checkpoint/restart* e apresenta algumas desses bibliotecas.

3. *Checkpoint/Restart*

Em uma aplicação paralela MPI, os processos cooperam através de trocas de mensagens. A execução destes processos pode ser vista como uma seqüência de ordenada de eventos como, por exemplo, o envio e recebimento de mensagens. Cada evento muda o estado de seu processo [14]. Desta forma, o conjunto formado pelos estados locais de todos os processos e seus canais de comunicação representam o estado global da aplicação [13].

Um estado global é dito consistente, quando ocorre em um período de execução correta e livre de falhas. Em um estado global consistente, se o estado de um processo indica que uma mensagem foi recebida, o estado do processo que enviou esta mensagem deve, necessariamente, indicar que a mensagem foi enviada [6]. Desta forma, um *checkpoint* global consistente é o conjunto dos *checkpoints* locais, um para cada processo, que forma um estado global consistente. Qualquer *checkpoint* global consistente pode ser usado para reiniciar a aplicação.

3.1. Implementações de *Checkpoint/Restart*

As implementações de *checkpoint/Restart* podem ser classificadas em três classes [12], de acordo com o nível de implementação: implementadas na própria aplicação, através de uma biblioteca ligada à aplicação ou no núcleo do sistema operacional. *Checkpoint* implementado na aplicação tende a prover maior eficiência, uma vez que o programador da aplicação possui o conhecimento de quais as estruturas de dados precisam ser salvas, e quais podem ser descartadas. No entanto, esta abordagem possui algumas desvantagens, como o custo de implementação de uma solução especifica para cada aplicação. Outra desvantagem são as limitações de onde o *checkpoint* é realizado, normalmente, o *checkpoint* só pode ser feito no final de um período de processamento.

Implementações de *checkpoint* em nível de aplicação pode resolver alguns desses problemas. O uso de bibliotecas pode eliminar a necessidade de modificações no código da aplicação. Além disso, implementação em nível de biblioteca, tipicamente, utilizam tratamento de sinais para disparar o procedimento de *checkpoint*. Isto pode eliminar as restrições de onde o *checkpoint* pode ser aplicado. No entanto, a utilização de *checkpoint* implementado como biblioteca impõem restrições de quais chamadas de sistema podem ser usadas, já que esta abordagem é implementada em nível de usuário (*user-level*). Comunicação inter-processos, *sockets*, descritores de arquivos e outras funcionalidades, que exigem acesso à informações internas ao sistema operacional, são normalmente perdidas.

Mecanismos de *checkpoint* implementados em nível de sistemas necessitam ser compilados dentro do núcleo do sistema operacional ou como módulos carregáveis (*loadable kernel module*). Esta abordagem diminui as restrições no escopo de aplicações que podem realizar *checkpoint* e, posteriormente, serem recuperadas corretamente uma vez que muito mais estruturas de dados são acessíveis de dentro do núcleo do sistema operacional. Além disso, implementações em nível de sistema tipicamente podem ser realizado a qualquer instante. Devido à dificuldade de implementação de *checkpoint/restart* em nível de sistema, existem, atualmente, implementações para poucos sistemas operacionais.

- Libckpt [11] é uma implementação em nível de biblioteca e, logo, possui todas as limitações típicas de bibliotecas de *checkpoint/restart*. No entanto, libckpt provê algumas otimizações que reduzem o tamanho dos arquivos de contexto. Além disso, esta biblioteca suporta *checkpoint* incremental, que salva somente as paginas de memória que foram salvas desde o ultimo salvamento.

- Condor [10] é outro sistema que provê serviços de *checkpoint* de processos. O modulo de *checkpoint* de Condor é transparente e totalmente implementado em nível de usuário. A utilização de Condor requer que a aplicação seja ligada à uma biblioteca especial, mas nenhuma modificação do código é necessária.

- BLCR (*Berkeley Lab's Checkpoint/Restart*) [9] é uma implementação robusta de *checkpoint/restart* a nível de sistema. BLCR é implementado como um modulo do *kernel* Linux e uma biblioteca de nível de usuário. A biblioteca permite que aplicações e outras bibliotecas possam interagir com *checkpoint/restart*. Esta biblioteca permite, por exemplo, o registro de funções de *callback* definidas pelo usuário, que são executadas cada vez que um *checkpoint* ou recuperação for solicitado.

- CRAK (*Checkpoint/Restart As a Kernel Module*) é outra implementação de *checkpoint/restart* em nível de sistema. CRAK foi projetado para migração de processos e suporta *checkpoint* de *sockets* TCP. Além disso, a utilização de CRAK não exige nenhuma modificação na aplicação.

3.2. *Checkpoint/Restart* para MPI

Existem várias bibliotecas e ferramentas que permitem *checkpoint/restart* em aplicações MPI.

- CoCheck [14] é uma ferramenta que provê *checkpoint* para aplicações MPI e PVM. No entanto, esta ferramenta foi projetada para trabalhar com tuMPI, uma implementação acadêmica da norma MPI que não é mais suportada por seus desenvolvedores.
- CLIP [7] é uma ferramenta de *checkpoint* para aplicações MPI e Intel NX. Esta ferramenta utiliza a biblioteca de libckpt para prover salvamento de contexto. No entanto, CLIP foi projetada para o multicomputador Intel Paragon e também não é mais suportada por seus desenvolvedores.
- Starfish [3] provê detecção de falhas e recuperação de aplicações MPI-2. No entanto, Starfish suporta somente aplicações desenvolvidas na linguagem de programação OCaml, o que limita a sua utilização prática.
- MPICH-V [4] é uma extensão da implementação MPICH para prover tolerância à falhas. MPICH-V pode trabalhar com diferentes protocolos de obtenção de estado global consistente e utiliza a biblioteca Condor para realizar o salvamento de contexto.

A maior parte das ferramentas citadas aqui são projetadas para implementações MPI pouco utilizadas, ficando restritas ao uso acadêmico. LAM/MPI (*Local Área Multicomputer MPI*) [1], por outro lado, é uma implementação da norma MPI que já possui um grande número de usuários e apresenta um bom desempenho, quando comparada com outras implementações[2]. Além disso, LAM/MPI possui um suporte a *checkpoint/restart* que utiliza o pacote BLCR.

Alguns trabalhos, disponíveis na literatura, indicam que o suporte a *checkpoint* da LAM/MPI é funcional e já foi utilizado, com sucesso, para realizar migração de processos MPI [13, 5, 8]. Além disso, testes realizados no Laboratório de Tecnologia em Cluster (LabTeC) da UFRGS, confirmaram a funcionalidade desta ferramenta. A seção a seguir descreve o mecanismo de *checkpoint/restart* da implementação LAM/MPI.

4. Checkpoint/Restart na LAM/MPI

LAM/MPI possui uma arquitetura de duas camadas: a camada LAM e camada MPI. A camada LAM provê um *framework* (SSI) e um ambiente de execução (RTE), sobre o qual a camada MPI executa. LAM utiliza uma pequeno *daemon* (lamd) de nível de usuário para o controle de processos, comunicação e redirecionamento de mensagens entre processos. Esse *daemon* é carregado em cada nó do *cluster*, no início de uma seção, formando um *cluster* lógico. Já a camada MPI fornece uma interface e uma infraestrutura para troca de mensagens entre os processos do *cluster* lógico.

A interface com o usuário é feita por aplicativos de linha de comando. Estes aplicativos permitem, entre outras ações, a inicialização da LAM (lamboot), compilação de programas (mpicc, mpif77, etc.) e carga e execução de aplicações (mpirun).

Durante a carga dos processos da aplicação, utilizando mpirun, é possível selecionar o módulo MPI, que implementa as primitivas de comunicação, o módulo RPI (*Request Progression Interface*) que implementa a camada básica de comunicação dependente de dispositivo e o módulo CR, responsável pela funcionalidade de *checkpoint* e recuperação.

Até o momento, o módulo RPI de responsável por aceitar requisições de *checkpoint* e coordenar a obtenção de estado global consistente, possui implementações para TCP (crtcp) e Myrinet (gm). Suporte InfiniBand e outras interconexões de auto desempenho estão sendo estudados [5]. Já o módulo CR, atualmente, possui uma implementação única, que utiliza BLCR.

O processo de *checkpoint* implementado na LAM/MPI, inicia com uma solicitação, que pode ser realizada pelo usuário ou pela própria aplicação. Quando mpirun recebe uma solicitação de *checkpoint*, propaga-a para todos os processos da aplicação. Cada processo, por sua vez, negocia com os outros processos para atingirem um estado global consistente. Uma vez alcançado um estado consistente, cada processo solicita o salvamento de seu contexto e, então, continua sua execução normal. Neste ponto, mpirun se prepara para uma possível recuperação e indica que está pronto para o *checkpoint*. Então, o contexto de mpirun é salvo em arquivo.

No processo de recuperação, mpirun inicia a sua execução a partir de seu contexto recuperado e solicita que todos os processos também sejam reiniciados a partir de seus respectivos arquivos de contexto. Cada processo, após serem reiniciados, enviam suas novas informações de processo para mpirun. mpirun, por sua vez, constrói a tabela global, que contém as informações sobre todos os processos da aplicação MPI, e propaga para todos os processos. Os processos, após receberem estas informações, reconstroem os canais de comunicação com os outros processos. Após esse procedimento, a aplicação continua sua execução normal.

5. Considerações Finais

O presente artigo apresentou um levantamento de mecanismos *checkpoint/restart* para migração de processos MPI. Existem diversas bibliotecas para salvamento de contexto de processos. No entanto, o *checkpoint* em aplicações MPI envolve algumas complicações, como a obtenção de um estado global consistente, que exigem soluções específicas.

Este artigo considerou, baseado em trabalhos encontrados na literatura e testes realizados no LabTeC, a implementação de *checkpoint* da LAM/MPI como sendo a solução mais utilizável na prática. A maior parte das outras soluções apresentam restrições que limitam a utilização.

Este trabalho ainda está no início, sendo que, como próxima atividade, está previsto a realização de testes de migração de processos MPI. Futuramente, pretende-se utilizar geração automática de grafos de fluxo de dados e migração de processos para realizar mapeamento de programas MPI em tempo de execução.

Referências

[1] LAM/MPI Parallel Computing, 2006. Disponivel em http://www.lam-mpi.org. Acessado em junho de 2006.

[2] MPI Performance on Coral, 2006. Disponivel em http://www.icase.edu/coral/mpi/MPIonCoral.html. Acessado em junho de 2006.

[3] A. Agbaria and R. Friedman. Starfish: Fault-tolerant Dynamic MPI Programs on Clusters of Workstations. In *Proceedings of 8th International Symposium on High Performance Distributed Computing (HPDC-8'99)*, page 31. IEEE Computer Society Press, Aug 1999.

[4] G. Bosilca, A. Bouteiller, F. Cappello, S. Djilali, G. Fedak, C. Germain, T. Herault, P. Lemarinier, O. Lodygensky, F. Magniette, V. Neri, and A. Selikhov. MPICH-V: Toward a Scalable Fault Tolerant MPI for Volatile Nodes. In *Proceedings of IEEE Supercomputing 2002*, Nov 2002.

[5] J. Cao, Y. Li, and M. Guo. Process Migration for MPI Applications based on Coordinated Checkpoint. In *Proceedings of 11th International Conference on Parallel and Distributed Systems*, pages 306–312. IEEE Computer Society Press, Jul 2005.

[6] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, Feb 1985.

[7] Y. Chen, J. S. Plank, and K. Li. CLIP: A Checkpoint Tool for Message Passing Parallel Programs. In *Proceedings of Supercomputer'97*, Nov 1997.

[8] A. da Silva Martins Jr. and R. A. L. Gonçalves. Extensões na LAM/MPI para Automatizar o Checkpoint e Tolerar Falhas em Cluster de Computadores. In *Proceedings of Workshop em Sistemas Computacionais de Alto Desempenho (WSCAD'2005)*, Out 2005.

[9] J. Duell, P. Hargrove, and E. Roman. The Design and Implementation of Berkeley Lab's Linux Checkpoint/Restart, 2002.

[10] M. Litzkow, T. Tannenbaum, J. Basney, and M. Livny. Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System. In *Technical Report CS-TR-1997-1346. University of Wisconsin.*, Apr 1997.

[11] J. S. Plank, M. Beck, G. Kingsley, and K. Li. **Libckpt**: Transparent checkpointing under Unix. In *Usenix Winter Technical Conference*, pages 213–223, January 1995.

[12] E. Roman. A Survey of Checkpoint/Restart Implementations, 2006. Disponivel em http://ftg.lbl.gov/CheckpointRestart/checkpointSurvey-020724b.pdf. Acessado em junho de 2006.

[13] S. Sankaran, J. M. Squyres, B. Barrett, A. Lumsdaine, J. Duell, P. Hargrove, and E. Roman. The LAM/MPI checkpoint/restart framework: System-initiated checkpointing. *International Journal of High Performance Computing Applications*, 19(4):479–493, Winter 2005.

[14] G. Stellner. CoCheck: Checkpointing and Process Migration for MPI. In *Proceedings of the International Parallel Processing Symposium*, pages 526–531, Honolulu, HI, Apr 1996. IEEE Computer Society Press.