UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

MARCELO VEIGA NEVES

Modelagem e Dimensionamento do Custo de Migração de Processos em Programas MPI

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Nicolas Bruno Maillard
Orientador

Porto Alegre, julho de 2009

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Neves, Marcelo Veiga

Modelagem e Dimensionamento do Custo de Migração de Processos em Programas MPI / Marcelo Veiga Neves. – Porto Alegre: PPGC da UFRGS, 2009.

75 p.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2009. Orientador: Nicolas Bruno Maillard.

1. Migração de processos. 2. MPI. 3. Modelagem de custo. 4. Escalonamento dinâmico de processos. 5. Processamento paralelo. I. Maillard, Nicolas Bruno. II. Título.

"I love deadlines.
I like the whooshing sound they make as they fly by."
— DOUGLAS ADAMS

AGRADECIMENTOS

Agradeço a todas as pessoas que direta ou indiretamente contribuíram para a realização deste trabalho. Ao meu orientador. Aos colegas da UFRGS e da Digitel. À minha namorada linda! À minha família: minha mãe, minhas irmãs e minha sobrinha. Muito Obrigado.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ARP	Address Resolution Protocol
GPPD	Grupo de Processamento Paralelo e Distribuído
C/R	*Checkpoint/Restart*
HPL	*High Performance Linpack*
MPI	*Message Passing Interface*
MV	Máquina Virtual
NFS	Network File System
NPB	*NAS Parallel Benchmark*
OMPI	Open MPI
SMP	*Symmetric multiprocessing*
WWS	*Writable Working Set*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A migração de processos é importante em programas MPI por vários motivos, tais como permitir re-escalonamento de processos, balanceamento de cargas e tolerância a falhas. Independentemente do tipo do uso da migração, conhecer o custo imposto pela realização desta operação é um problema pertinente. Quando utiliza-se migração para tentar diminuir o tempo de execução de uma aplicação paralela, este custo passa a ser um ponto crítico. Existem algumas soluções para migração de processos em programas MPI disponíveis atualmente. No entanto, ainda não existe um estudo que quantifique o custo destas migrações.

Nesse contexto, este trabalho apresenta um estudo para modelar e dimensionar o custo de migração de processos em programas MPI. Primeiramente, o trabalho identificou, analisou, avaliou e, quando necessário, adaptou as principais soluções disponíveis atualmente para migrar processos MPI. Com base nessas soluções, foram criados modelos de custo que poderão ser utilizado para estimar dinamicamente os custos de migração e auxiliar na tomada de decisão em algoritmos de escalonamento.

Os modelos criados foram utilizados para estimar os custos de migração em aplicações paralelas e o resultado foi comparado com os custos de migração reais. Nesta comparação, os valores previsto ficaram bastante próximos dos valores observados no experimento, demonstrando a qualidade das previsões dos modelos propostos.

Palavras-chave: Migração de processos, MPI, modelagem de custo, escalonamento dinâmico de processos, processamento paralelo.

ABSTRACT

Process migration is essential for MPI programs for different reasons, such as processes rescheduling, load balancing and fault tolerance. Knowing well the cost necessary for this operation is a pertinent problem, regardless of the type of migration use. Whenever migration is used for improving the performance of parallel applications, its cost becomes a deciding point. Nowadays, there are some solutions to process migration available for MPI programs. However, there is not a study that can quantify the migration cost and its impact on the execution of MPI programs.

In this context, this work presents a study for modeling and dimensioning the process migration cost in MPI programs. First, we identified, analyzed, evaluated and, when needed, adapted the main solutions which are presently available to migrate MPI processes. Based in these solutions, we defined cost models. These models can be used to dynamically estimate the migration costs and to guide scheduling decisions.

These models were used to predict the migration cost in parallel applications and the result was compared to observed migration costs. In this comparison, the predicted values were very similar to those observed in the experiment. This work still shows an evaluation about the impact of a migration in the execution of real parallel applications in order to verifying the viability of applying this approach to improve the performance.

Keywords: process migration, MPI, cost modeling, dynamic process scheduling, parallel processing.

1 INTRODUÇÃO

O processamento paralelo e distribuído é frequentemente utilizado para resolver problemas que demandam alto poder computacional. A paralelização de aplicações permite que se obtenha os dados de resultados mais rapidamente e uma melhor utilização dos recursos computacionais. Existem diversas arquiteturas que permitem executar programas paralelos e distribuídos. Um exemplo são os agregados de computadores (NAVAUX; DE ROSE, 2003) que, por oferecerem uma boa relação custo-desempenho, se tornaram bastante populares (BAKER; BUYYA, 1999).

Atualmente existem várias abordagens e ferramentas para a programação paralela e distribuída. Neste contexto, MPI (*Message Passing Interface*) destaca-se como um padrão de fato para comunicação em agregados (GROPP et al., 1996). Apesar disso, a norma MPI não prevê **migração de processos**, funcionalidade desejável em aplicações desse tipo por várias razões. Dentre essas razões, pode-se relacionar:

- Balanceamento de cargas: utilizando migração de processos é possível transferir aqueles que estejam executando em nós sobrecarregados para nós ociosos ou menos carregados. Especialmente em sistemas distribuídos que apresentam uma utilização não homogênea dos recursos, isto é, alguns nós permanecem ociosos enquanto outros tornam-se sobrecarregados, o balanceamento de cargas pode distribuir de forma eficiente os processos entre os nós disponíveis. Assim, é possível utilizar melhor os recursos computacionais e, consequentemente, obter ganho de desempenho em aplicações.

- Exploração da localidade de recursos: utilizando migração é possível transferir processos para o outro lado de um canal de comunicação e, desta forma, transformar uma conexão remota em local. Normalmente, o acesso local a recursos é mais eficiente que o remoto. Em aplicações paralelas, por exemplo, pode-se utilizar migração para agrupar processos que trocam muitas informações em uma mesma máquina e, assim, reduzir o volume de dados trafegados pela rede. A localidade pode ser explorada em vários níveis como, por exemplo, quando se agrupam agregados interligados por redes de diferentes capacidades de transmissão.

- Tolerância a falhas: utilizando migração é possível realocar processos de nós que apresentam falha parcial ou desligamento administrativo, com o objetivo de evitar a interrupção da execução dos mesmos. Por exemplo, existem aplicações que podem executar por dias ou talvez semanas. Neste contexto, em caso de falha ou desligamento de uma máquina, todo o processamento realizado até aquele momento é perdido. No caso de um desligamento para fins administrativos – como troca de

um dispositivo de *hardware*, por exemplo – o processo pode ser temporariamente transferido para outra máquina, até que a sua máquina de origem seja religada.

Independentemente do tipo do uso da migração, uma questão é pertinente: qual o custo para migrar um processo em um programa MPI? O custo de migração de processo tem um impacto direto no desempenho da aplicação. Por exemplo, no caso do balanceamento de cargas, o impacto no desempenho da aplicação não deve exceder o ganho obtido com o balanceamento. Da mesma forma, não faz sentido agrupar processos para reduzir o volume de dados trafegados se o ganho de desempenho não for maior que custo das migrações.

Existem algumas soluções para migração de processos em programas MPI disponíveis atualmente (WANG et al., 2008; WALTERS; CHAUDHARY, 2007; HUANG et al., 2006). No entanto, ainda não existe um estudo que quantifique o custo de migrações e o impacto destas na execução de aplicações MPI. A maioria dos trabalhos disponíveis na literatura focam na otimização do balanceamento de cargas ou escalonamento e não avaliam os custos e impactos de uma migração de processos propriamente dita. Além disso, os trabalhos que envolvem migração não consideram as características das aplicações MPI na modelagem do custo de migração de processos.

O trabalho de Halchor-Balter et al. (HARCHOL-BALTER; DOWNEY, 1997) propõe um modelo de balanceamento de cargas que leva em consideração o custo de migração e a duração da execução do processo. Nesse modelo, o custo de migração é obtido diretamente pelo cálculo do tempo de transferência da memória do processo através da rede. Apesar disso, este trabalho concluiu que há benefícios na migração de processo quando a execução possui uma longa duração. O trabalho de Mello et al. (MELLO; SENGER, 2004) propõe um modelo de balanceamento de cargas que permite detectar dinamicamente o melhor ponto para iniciar a migração. Para isso, o trabalho leva em consideração o custo de migração, mas negligencia na simplicidade da modelagem deste custo. Já o trabalho de Cong Du et al.(DU; SUN; WU, 2007) apresenta um modelo bastante detalhado para o custo de uma migração de processos baseado em *Checkpoint/Restart*, mas mantém o mesmo específico para este tipo de mecanismo.

1.1 Objetivos do Trabalho

O principal objetivo deste trabalho é criar um modelo que permita dimensionar o custo de migração de um determinado processo MPI. Através deste modelo e de execuções experimentais, pretende-se responder a pergunta de quanto custa para migrar um processo MPI com os mecanismos de migração disponíveis. O modelo proposto também poderá ser utilizado para calcular dinamicamente os custos de migração e auxiliar na tomada de decisão em algoritmos de escalonamento.

A maior parte das iniciativas para adicionar suporte a migração de processos ao MPI apresentam restrições técnicas que limitam sua utilização prática ou são projetadas para alguma implementação MPI pouco utilizada, ficando restritas ao uso acadêmico. Desta forma, um dos objetivos deste trabalho é identificar e avaliar mecanismos de migração funcionais. Para alcançar este objetivo, foram estudadas as principais abordagem disponíveis atualmente para migração de processos. Deste estudo, foram selecionadas algumas soluções para serem avaliadas e, quando necessário, adaptadas de modo a eliminar alguma restrição técnica que impedisse a sua utilização.

Outro objetivo do trabalho é avaliar o impacto da realização de migrações na execução de aplicações MPI. Esta avaliação é importante para verificar a viabilidade de utilizar

migração de processos para obter ganho de desempenho, levando em consideração as soluções de migração disponíveis. Para alcançar este objetivo, foram realizados testes desempenho com aplicações paralelas e *micro-benchmarks*. A partir destes testes é possível avaliar de forma isolada o sobrecusto das ferramentas de migração e o impacto de uma migração no tempo total de execução das aplicações.

1.2 Contextualização

Este trabalho está inserido em uma linha de pesquisa do grupo GPPD (Grupo de Processamento Paralelo e Distribuído), que trata das questões de *escalonamento de processos* em programas MPI. A norma MPI por si só não especifica escalonamento de processos, ficando a cargo do programador realizá-lo, preenchendo a ordem do arquivo de nós da aplicação[1].

Quanto à classificação, o escalonamento pode ser dividido em dois grandes grupos: estático ou dinâmico (CASAVANT; KUHL, 1988). No escalonamento estático, as decisões sobre a distribuição dos processos são tomadas antes da execução da aplicação e colocadas em prática no lançamento. Neste caso, para que o escalonamento tenha um bom resultado, é necessário conhecer previamente as características do ambiente e do comportamento da execução da aplicação. No escalonamento dinâmico, por outro lado, as decisões são tomadas em tempo de execução, isto é, os processos podem ser redistribuídos durante a execução da aplicação. Se for preciso alterar a localização dos processos para um nó distinto, faz-se necessário a utilização de algum mecanismo que consiga migrar o processos com um custo aceitável.

Neste contexto, foram desenvolvidas duas bibliotecas no grupo GPPD, uma para tratar o escalonamento estático de programas MPI (SILVA et al., 2005) e outra para o dinâmico (CERA et al., 2006). A primeira solução propõe executar a aplicação uma vez, obter informações sobre seu comportamento e, então, executá-la novamente utilizando estas informações para guiar o escalonamento. No entanto, esta abordagem impõe o sobrecusto de uma execução prévia e restringe o escopo de aplicações que podem ser escalonadas àquelas que mantém o mesmo comportamento após execuções sucessivas. A segunda solução atua sobre a norma 2 de MPI e escolhe o destino de processos que são lançados de forma dinâmica através de novas diretivas dessa interface. Neste caso, é possível criar novos processos durante a execução, mas ainda não é possível migrar processos já existentes.

Ainda nessa linha de pesquisa, está sendo desenvolvido o modelo MigBSP (RIGHI et al., 2008). Este modelo visa permitir o re-escalonamento de processos em tempo de execução em aplicações do tipo BSP (*Bulk Synchronous Parallel*). Uma aplicação BSP é dividida em um ou mais superpassos, cada qual contendo fases de computação e comunicação seguidas por uma barreira de sincronização. Uma vez que a barreira espera pelo processo mais lento, a ideia de MigBSP é ajustar a localização dos processos, utilizando migração de processos, com o intuito de reduzir o tempo dos superpassos e obter um ganho de desempenho na execução da aplicação. Um fator crítico nesse modelo é conhecer, de antemão, o custo de migração de um dado processo.

Assim, o presente trabalho irá contribuir com os demais trabalhos do grupo fornecendo um estudo sobre os mecanismos de migração disponíveis, um modelo para dimensionar o custo das migrações e uma avaliação do impacto das migrações em aplicações

[1]Normalmente, implementações MPI fazem escalonamento Round-Robin sobre a lista de nós do arquivo de máquinas da aplicação.

paralelas. Especialmente no caso de MigBSP, será possível a utilização do modelo proposto para calcular dinamicamente o custo de migração e auxiliar na tomada de decisão do escalonamento.

1.3 Organização do Texto

O texto deste trabalho está organizado nos capítulos apresentados a seguir.

- O Capítulo 2 introduz os conceitos e problemas referentes à migração de processos em programas MPI e estuda as principais soluções de migração disponíveis para este tipo de programa. As soluções de migração são analisadas e, como resultado, obtém duas abordagens principais para serem estudadas em detalhes: a migração baseada em *checkpoint/restart* e a migração baseada em máquinas virtuais.

- O Capítulo 3 estuda a utilização de *checkpoint/restart* para migrar processos MPI. Este capítulo apresenta os conceitos envolvendo a migração de processo deste tipo e escolhe uma solução para ser estudada em mais detalhes. A solução escolhida é o suporte a *checkpoint/restart* de Open MPI (HURSEY et al., 2007), que foi adaptado para permitir migração de processos.

- O Capítulo 4 estuda a virtualização de recursos computacionais e a utilização de migração de máquinas virtuais para migrar processos MPI. Este capítulo apresenta alguns conceitos de virtualização e sua influência no desempenho do ambiente virtualizado. O sistema de virtualização Xen (CLARK et al., 2005) foi escolhido para ser estudado em detalhes e pôde ser utilizado para prover migração de processos.

- O Capítulo 5 propõe um modelo de custo de migração de processos para aplicações MPI. Primeiramente, define-se um modelo geral para descrever a execução de uma aplicação paralela com migração de processo, levando em consideração as características de MPI. A partir deste modelo, são propostos modelos de custo de migração para cada uma das classes de soluções analisadas.

- O Capítulo 6 descreve a metodologia utilizada na realização dos experimentos que avaliam o desempenho das solução de migração estudadas, estimam os parâmetros e verificam a corretude da modelagem de custo de migração apresentada no Capítulo 5. A metodologia apresentada envolve desde a descrição do ambiente de execução até os métodos estatísticos utilizados na análise dos resultados.

- O Capítulo 7 realiza a estimava dos parâmetros e avalia os modelos propostos. Através da estimativa de parâmetros, obtém-se um modelo preenchido que pode ser utilizado para calcular, de antemão, o custo de migrar um processo em uma dada aplicação. Ainda neste capítulo, são realizados testes de desempenho utilizando aplicações paralelas e *micro-benchmarks* para avaliar o sobrecusto das soluções de migração e o impacto das migrações na execução de aplicações.

- O Capítulo 8 apresenta as considerações finais e revê as principais contribuições do trabalho. Além disso, são apresentadas sugestões para trabalhos futuros.

2 CONCEITOS DE MIGRAÇÃO DE PROCESSOS

O objetivo deste capítulo é introduzir os conceitos e problemas referentes à migração de processos em programas MPI e estudar as principais soluções de migração disponíveis para este tipo de programa. O estudo das soluções de migração envolve a identificação das principais abordagens disponíveis para migração de processos. Para cada uma dessas abordagens, são analisadas suas principais soluções representantes. Os critérios empregados nessa análise são a existência de limitações/restrições de utilização, possibilidade de utilização em programas MPI e a existência de trabalhos científicos nessa área.

O texto está organizado da seguinte forma. Primeiramente, são introduzidos os conceitos de processo (Seção 2.1) e migração de processos (Seção 2.2). Na sequência (Seção 2.3), são descritos os problemas envolvendo a migração de processos em programas MPI. A Seção 2.4 apresenta as abordagens que podem ser utilizadas para solucionar estes problemas e também estuda os principais mecanismos de migração disponíveis atualmente. Por fim, a Seção 2.5 faz uma conclusão do capítulo e discute as abordagens e mecanismos de migração estudados.

2.1 Processos

Processo é um conceito chave em sistemas operacionais (TANENBAUM, 1992). Um processo é basicamente um programa em execução. Associado a cada processo, existe um espaço de endereçamento, geralmente dividido em código (*text*), dados (*data*) e pilha (*stack*); um conjunto de registradores, incluindo o contador de programa, o ponteiro para pilha e outros registradores de *hardware* disponíveis; e todas as outras informações necessárias à execução o programa. Algumas dessas informações são relacionadas aos recursos usados pelos processos, tais como descritores de arquivos, conexões de rede, estado de sinais pendentes, estado de terminais, etc.

Em um dado instante, o estado de um processo pode ser representado por dois componentes: o estado inicial do processo e as modificações resultantes da execução até aquele ponto (SMITH, 1988). O estado inicial pode ser recuperado a partir da imagem do programa executável. Isto é, o mesmo que iniciar a execução de um programa em um sistema operacional. Já as informações relativas às modificações ocorridas durante a execução podem estar distribuídas em diferentes pontos, desde dados no espaço de endereçamento do processo até dados não acessíveis diretamente, mantidos em tabelas internas ao sistema operacional. O estado completo de um processo em execução é muitas vezes referenciados como contexto de execução.

2.2 Migração de Processos

Migração de processos é o ato de transferir um processo de uma máquina origem para uma máquina destino durante a sua execução (MILOJICIC et al., 2000). O procedimento de migração envolve a transferência de um conjunto significativo de informações que representam o estado do processo (contexto de execução descrito na seção anterior), para que este possa ter sua execução continuada corretamente na máquina destino.

A Figura 2.1 apresenta o fluxo de execução de uma migração de processo. No caso ilustrado, o processo migrante não começa a executar na máquina destino até que todas as informações de seu estado sejam transferidas (isto é, o processo interrompe sua execução no ponto A do nó de origem e retorna somente no ponto B do nó destino). No entanto, é possível que o processo migrante comece a executar na máquina destino quase imediatamente, através da transferência de informações sob demanda. Neste caso, primeiro são enviadas informações mais importantes à continuação da execução no nó destino e, após, são enviadas as restantes conforme necessário. Da mesma forma, é possível que o processo continue executando no nó de origem até que a maior parte das informações tenham sido transferidas, a fim de minimizar o período de indisponibilidade (período de tempo entre os pontos A e B). Isto é o que se costuma chamar de *live migration*.

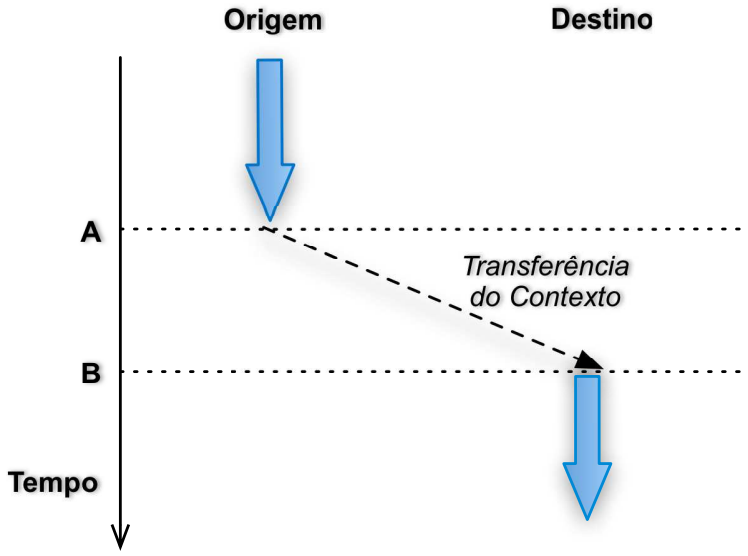


Figura 2.1 Fluxo de execução de uma migração de processo.

2.3 Migração de Processos em Programas Paralelos MPI

Message Passing Interface (MPI) é um padrão de fato para comunicação de dados em computação paralela (GROPP et al., 1996). MPI foi definido para prover uma interface única para o desenvolvimento de programas baseados em passagem de mensagens. Esse padrão possui diversas implementações, como por exemplo LAM/MPI (LAM/MPI, 2008), MPICH (MPICH2, 2008) e Open MPI (OPENMPI, 2008).

Uma aplicação desenvolvida no padrão MPI é constituída por um ou mais processos que podem ser mapeados para diferentes processadores. A comunicação entre os processos é realizada através de funções de envio e recebimento de mensagens. Quando os

processos comunicantes estão localizados em máquinas distintas, a transmissão das mensagens é realizada via interface de rede. Na maioria das implementações de MPI, essas mensagens são enviadas através de *sockets* TCP/IP.

A migração de processos em uma aplicação MPI é mais difícil que a de processos individuais, porque processos MPI são mais fortemente acoplados (CAO; LI; GUO, 2005). Se um processo MPI for migrado enquanto a aplicação estiver em execução, todos os demais precisam ser informados de sua nova localização. Além disso, se existirem mensagens pendentes durante a migração, há a possibilidade da aplicação falhar.

Além disso, a migração de processos comunicantes envolve um problema adicional, que é a migração de conexões de rede. Processos que se comunicam através do conjunto de protocolos TCP/IP (TANENBAUM, 2003) são distinguidos por um endereço de IP e uma porta, os quais são informações locais a uma máquina. Da mesma forma, comunicações que utilizam o protocolo TCP necessitam manter uma conexão que é identificada pela 4-tupla tradicional (IP de origem, porta de origem, IP destino, porta destino), que também são informações que dizem respeito à localização das máquinas. Desde que a migração de um processo envolve a modificação da sua localização física, este tipo de informação precisa ser tratada corretamente. Além disso, normalmente as informações de comunicação têm expiração por inatividade (*timeout*), o que impõe a necessidade de um tempo de indisponibilidade não muito elevado durante a migração.

2.4 Abordagens para Migração de Processos MPI

Ao longo do tempo, as pesquisas nesta área levaram ao desenvolvimento de várias soluções para a migração de processos. No entanto, devido aos problemas descritos na seção anterior, poucas delas podem ser aplicadas diretamente em programas MPI. Apesar disso, algumas soluções para migração de processos em MPI foram propostas e algumas delas em trabalhos bastante recentes (WANG et al., 2008, 2007; WALTERS; CHAUDHARY, 2007; CAO; LI; GUO, 2005). Neste contexto, o restante desta seção faz um levantamento das principais soluções de migração disponível atualmente e verifica quais delas podem ser utilizadas para programas MPI. É importante ressaltar que, neste trabalho, foram consideradas apenas soluções publicamente disponíveis, de modo que pudessem ser verificadas e/ou validadas na prática.

De modo geral, os sistemas para migração de processos disponíveis hoje podem ser classificados em umas das três categorias principais: (i) baseadas em *checkpoint/restart*; (ii) baseadas em sistemas operacionais *Single System Image* para agregados; e (ii) baseadas em máquinas virtuais. A seguir, essas categorias são discutidas e seus principais representantes são apresentados.

2.4.1 Migração Baseada em *Checkpoint/Restart*

A técnica de *checkpointing* consiste na criação de um arquivo de descrição de um processo em execução, o qual pode ser utilizado para reconstruir o processo em um momento futuro (*restart*). Este arquivo contém uma imagem do estado de execução do processo em um dado instante de tempo (contexto de execução descrito na Seção 2.1). Isto possibilita que o processo possa continuar sua execução a partir do ponto onde o *checkpoint* foi realizado. Devido a esse comportamento, essa técnica é comumente chamada de *checkpoint/restart* (C/R).

Existem algumas soluções que utilizam C/R para implementar migração de processos (ROMAN, 2002). Neste caso, o *checkpoint* pode ser realizado em conjunto com

um sistema de arquivos compartilhado ou um *socket* aberto para transferir o contexto de execução para um nó remoto e, então, o processo pode ser imediatamente reconstruído (continuação da execução) no nó destino. Em resumo, migração de processos baseada em C/R divide-se em três fases:

1. Criação do *checkpoint* do processo em execução no nó de origem;
2. Transferência dos dados do *checkpoint* para o nó destino;
3. Reconstrução (*restart*) do processo no nó destino, usando os dados do *checkpoint*.

Existem algumas implementações de MPI com suporte a C/R. Na maioria dos casos, a funcionalidade de C/R é utilizada para tolerar falhas nos nós que executam a aplicação. No entanto, em muitos casos, é possível utilizar essa técnica para prover migração de processos. A seguir são apresentadas as principais implementações MPI com suporte a C/R e as suas relações com a funcionalidade de migração de processos.

2.4.1.1 MPICH-V

MPICH-V (*MPI implementation for Volatile resources*) (MPICH-V, 2008) é um projeto, iniciado na Universidade de Paris-Sud, que visa integrar algoritmos e protocolos de tolerância a falhas em uma implementação MPI baseada no MPICH. O mesmo é implementado como um *framework* e uma de suas principais funcionalidades é permitir, em caso de falhas, a transferência da execução de uma aplicação MPI completa de um agregado para outro.

Uma abstração fundamental utilizada por MPICH para implementar o padrão MPI é a noção de dispositivos (*devices*) (COTI et al., 2006). Os dispositivos implementam as rotinas básicas de comunicação para um *hardware* específico ou um novo protocolo de comunicação. No caso de MPICH-V, um *framework* genérico é utilizado para implementar um dispositivo, chamado *ch_v*, que pode suportar diferentes protocolos de tolerância a falhas.

MPICH-V é composto pelo dispositivo *ch_v* e por um conjunto de componentes, entre os quais estão: os *daemons* de comunicação, responsáveis por gerenciar a comunicação entre os nós; o *dispatcher*, responsável pelo lançamento dos processos MPI e pela detecção de falhas; e o servidor de *checkpoint*, responsável por criar e coletar os arquivos de contexto de todos os processos MPI locais. MPICH-V pode trabalhar com três mecanismo de C/R: Condor (*Condor Standalone Checkpointing Library*) (LITZKOW et al., 1997), libckpt (PLANK et al., 1995) e BLCR (*Berkeley Linux Checkpoint/Restart*) (DUELL; HARGROVE; ROMAN, 2002).

MPICH-V utiliza algoritmos de *checkpoint* coordenado e não coordenado para resolver o problema do estado global consistente (que é explicado em maiores detalhes no Capítulo 3). As implementações MPICH-VCL e MPICH-PCL realizam uma coordenação entre os processos de forma a esvaziar os canais de comunicação antes de iniciar o salvamento do contexto de execução. Já as implementações MPICH-V1 e MPICH-V2 utilizam um protocolo não coordenado, que armazena em filas as mensagens recebidas por cada processo desde o último *checkpoint*.

Existem diversos trabalhos (BOSILCA et al., 2002; BOUTEILLER et al., 2003), que avaliam o desempenho de MPICH-V para operações de C/R em aplicações MPI e, também, utilizam esse *framework* para comparar diferentes algoritmos de tolerância a falhas. No trabalho de Bosilca et al. (BOSILCA et al., 2002), MPICH-V foi utilizado para tolerar falhas em nós de um agregado durante a execução de aplicações MPI, através da

re-execução a partir dos dados de *checkpoint*. A re-execução após uma falha é realizada em um novo conjunto de nós, o que pode ser caracterizado como uma forma de migração de processos. Apesar disso, não foram encontrados artigos explorando as funcionalidades de tolerância a falhas de MPICH-V para realizar migração de processos propriamente dita.

2.4.1.2 LAM/MPI+BLCR

LAM/MPI (LAM/MPI, 2008) é uma implementação do padrão MPI, inicialmente desenvolvida pelo Ohio Supercomputing Center, mantida pela Universidade de Notre Dame. LAM/MPI disponibiliza, em sua implementação, um módulo para C/R de aplicações MPI. Este mesmo módulo pode ser usado para transferir a execução de uma aplicação completa de um agregado para outro. Além disso, com algumas modificações na implementação é possível alterar o mapeamento de processos específicos, o que caracteriza uma forma de migração de processos (CAO; LI; GUO, 2005; SILVA MARTINS JR.; GONÇALVES, 2005).

LAM/MPI possui uma arquitetura de duas camadas: a camada LAM e camada MPI. A camada LAM provê um ambiente de execução (*Run Time Environment* - RTE), sobre o qual a camada MPI executa. LAM utiliza um 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 agregado, no início de uma seção, formando um agregado lógico. Já a camada MPI fornece uma infraestrutura e interface para troca de mensagens entre os processos do agregado lógico. Também existe um *framework* (*System Services Interface* - SSI) que possibilita a criação de módulos de funcionalidades que podem atuar em ambas as camadas e serem selecionados em tempo de execução.

O suporte a C/R de LAM/MPI é implementado através de um módulo SSI chamado CR. O mecanismo de salvamento de contexto utilizado é BLCR (DUELL; HARGROVE; ROMAN, 2002). Uma vez solicitado um *checkpoint*, um protocolo coordenado é iniciado a fim de esvaziar os canais de comunicação, produzir um estado global consistente e salvar o contexto dos processos MPI. Após o salvamento do contexto, é possível interromper a aplicação e transferi-la inteiramente para outro conjunto de nós.

Recentemente, alguns autores propuseram soluções para migração de processos MPI baseando-se no suporte à C/R de LAM/MPI. O trabalho de Cao et al. (CAO; LI; GUO, 2005) propõe um esquema de migração que utiliza uma ferramenta para analisar os arquivos de *checkpoint* gerados pela BLCR e alterar as informações de localização do processo. Wang et al. (WANG et al., 2007) propõe uma solução que adiciona um estado de pausa ao suporte à C/R de LAM/MPI. Neste caso, não é necessário reiniciar todos os processos, apenas os processos migrados precisam ser reiniciados, os demais permanecem pausados esperando por um sinal de continue. A transferência dos arquivos de contexto dos processos migrados é realizada via NFS. Já o trabalho de Walters et al. (WALTERS; CHAUDHARY, 2007) propõe modificar as informações relativas à localização dos processos antes de ser efetuado o *checkpoint* e salvar o arquivo de contexto diretamente no computador destino, sem a necessidade de utilizar NFS.

2.4.1.3 OpenMPI+BLCR

Open MPI (*Open Source High Performance Computing*) (OPENMPI, 2008) é a combinação de vários projetos MPI já existente, com o objetivo de disponibilizar uma única implementação MPI integrando funcionalidades destes projetos. Entre as funcionalida-

des prevista estão C/R e migração de processos, apesar desta última ainda não ter sido implementada.

A arquitetura de Open MPI consiste em três camadas de abstração que, combinadas, provêm todas as suas funcionalidades. A camada mais superior é a OMPI (*Open MPI layer*), que provê a interface MPI para as aplicações. Logo abaixo, encontra-se a camada ORTE (*Open Run-Time Environment*), que provê um ambiente de execução paralelo independente das capacidades do sistema. Por fim, a camada OPAL (*Open Portable Access Layer*) abstrai as peculiaridades específicas do sistema a fim de aumentar a portabilidade das camadas superiores. Abaixo da camada OPAL, está o sistema operacional e os outros serviços que executam no nó local.

O suporte a C/R de Open MPI é implementado através de módulos ligados às diferentes camada de Open MPI, chamado módulo CR, que estende a solução de mesmo propósito de LAM/MPI. Da mesma forma que acontece em LAM/MPI, Open MPI utiliza um protocolo coordenado para obter um estado global consistente na realização do salvamento do contexto de execução. O mecanismo de salvamento de contexto utilizado é BLCR. Uma das principais diferenças em relação à implementação de LAM/MPI é o fato das informações de localização dos processos serem armazenados fora do arquivo de contexto, o que permite o remapeamento de processos após o *checkpoint*.

O trabalho de Hursey et al. (HURSEY et al., 2007), apresenta testes iniciais de desempenho da solução de C/R de Open MPI. Apesar de ter sido baseado no módulo de C/R de LAM/MPI, não foram encontrados trabalhos semelhantes ao descritos na Seção 2.4.1.2 explorando o C/R de Open MPI para realizar migração de processos.

2.4.2 Migração Baseada em *Single System Image*

Single System Image (SSI) é a propriedade de um sistema que esconde a heterogeneidade e a natureza distribuída dos recursos disponíveis. Neste tipo de sistema, os recursos distribuídos são apresentados aos usuários e aplicações como se formassem um recurso computacional único (BUYYA; CORTES; JIN, 2001). Do ponto de vista das aplicações, um sistema operacional para agregado SSI provê a mesma interface de um sistema operacional tradicional para um máquina SMP (multiprocessador simétrico), com algumas funcionalidades adicionais como, por exemplo, migração de processos.

Estes sistemas são atrativos principalmente pela facilidade de programação e uso (LOTTIAUX et al., 2005). No entanto, os sistemas SSI assumem um ambiente de agregado fortemente acoplado, o que pode não ser possível em alguns casos.

A seguir são apresentadas as soluções para agregado SSI com suporte a migração de processos disponíveis atualmente.

2.4.2.1 openMosix

openMosix (OPENMOSIX, 2006) é um sistema SSI baseado em MOSIX, um projeto de pesquisa iniciado na Universidade Hebraica de Jerusalém. openMosix permite a criação de agregados SSI sobre o *kernel* do Linux. Desta forma, é possível fazer múltiplos uniprocessadores e SMPs, rodando o mesmo *kernel*, cooperarem para executar um trabalho. Além disso, openMosix possui um mecanismo de balanceamento de cargas projetado para responder as variações *on-line* na utilização dos recursos entre diferentes nós. Isto é alcançado através da migração de processos de um nó para outro de forma preemptiva e transparente.

openMosix possui duas partes principais: um mecanismo de migração de processos preemptivo (*Preemptive Process Migration* - PPM) e um conjunto de algoritmos para

compartilhamento adaptativo de recursos. Ambas as partes são implementadas em nível de *kernel*, usando um módulo carregável. Desta forma, a interface do *kernel* não é modificada e openMosix é transparente para o nível de aplicação (BAR; RECHENBURG, 2003).

O PPM pode migrar qualquer processo, a qualquer momento, para qualquer nó disponível. Usualmente, as migrações são baseadas em informações providas por um dos algoritmos de compartilhamento de recursos. No entanto, os usuários podem ignorar qualquer sistema de decisão automática e migrar manualmente seus processos, através de uma interface de programação disponibilizada no sistema de arquivos /proc. Associado a cada processo, existe um único *home node*, que é o nó onde o processo foi criado. Após ser migrado, o processo continua possuindo uma forte dependência com o *home node* e as chamadas de sistema (como por exemplo, comunicações de rede e acesso a arquivos), que não poderiam ser processadas localmente após a migração, são realizadas remotamente através de um canal de comunicação com o *home node*. Uma desvantagem desta abordagem é o sobrecusto extra na execução destas chamadas de sistema.

Existem poucos trabalhos na literatura que descrevem a utilização de openMosix com MPI. No trabalho de Argentini (ARGENTINI, 2002), verificou-se que o uso de MPICH com openMosix não afeta o desempenho global da biblioteca. No entanto, não encontraram-se trabalhos que avaliem o sobrecusto de migrações em aplicações MPI. Apesar disso, acredita-se que MPI tenha seu desempenho degradado quando os processos migrados tenham que fazer chamadas de sistemas remotamente para comunicações de rede. Além disso, o projeto openMosix foi oficialmente fechado em março de 2008, após o inicio deste trabalho.

2.4.2.2 OpenSSI

OpenSSI (OPENSSI, 2006) é um sistema SSI baseado no projeto *Non-Stop Cluster for UnixWare*. O sistema OpenSSI provê uma plataforma capaz de integrar diversas tecnologias de código fonte aberto para agregados SSI. OpenSSI permite balanceamento dinâmico baseado na carga de CPU dos nós do agregado utilizando, para isso, migração de processos.

O mecanismo de migração utilizado em OpenSSI é derivado de Mosix. Quando ocorre uma migração em OpenSSI, todo o processo é enviado para o nó destino. No entanto, utiliza-se um mecanismo de acesso a recursos remotos para recursos que o sistema não consiga migrar. Este mecanismo é principalmente utilizado para IPC e algumas chamadas de sistema. O sistema OpenSSI também permite a migração de grupos de *threads*.

As principais características de OpenSSI são disponibilidade, escalabilidade e gerenciabilidade. Apesar de não ser desenvolvido explicitamente para alto desempenho, já foi reportado (KHAN, 2006) que OpenSSI trabalha corretamente com muitas ferramentas de alto desempenho, inclusive MPI em sua implementação MPICH. No entanto, não foram encontrados trabalhos descrevendo experiências de migração de processos em aplicações MPI.

2.4.2.3 Kerrighed

Kerrighed (KERRIGHED, 2006) é outro sistema SSI para agregados. O foco principal de Kerrighed está em aplicações de alto desempenho, agregados de alta disponibilidade, gerenciamento eficiente de recursos e facilidade de uso. Kerrighed é implementado como uma extensão do *kernel* Linux (um conjunto de módulos e um pequeno *patch* para o *kernel*). Kerrighed oferece um escalonador de processos global e configurável. A sua

política de escalonamento padrão realiza balanceamento dinâmico baseado na carga de CPU dos nós do agregado. Para isso, utiliza-se um esquema de migração preemptiva de processos. Quando um nó está pouco carregado, o sistema detecta do desbalanceamento e migra um processo de nó sobrecarregado para o nó menos carregado. Kerrighed também disponibiliza uma ferramenta para escrita de novas políticas definidas pelo usuário.

A migração de Kerrighed é baseada em vários mecanismos, tais como *ghosting*, *containers*, *streams* migráveis e um sistema de arquivos distribuído. O mecanismo *ghosting* é usado para extrair as informações de estado do processo e armazenar em um dado dispositivo. Este dispositivo pode ser um arquivo em disco (*checkpointing*), uma conexão rede (migração de processo ou criação remota de processo) ou uma região de memória (duplicação de processo). O mecanismo de *container* é usado para compartilhar dados entre os nós de forma coerente. Este mecanismo é utilizado para implementar compartilhamento de memória e um sistema de arquivos distribuído, chamado KerFS. O mecanismo de *stream* migrável é usado para tratamento eficiente de migração de processos comunicantes. Processos utilizando *pipes* e *sockets* podem ser migrados sem penalidade na latência ou largura de banda após a migração.

Existem alguns trabalhos na literatura que indicam a possibilidade de utilização de Kerrighed com MPI. O trabalho de Morin et al. (MORIN et al., 2003) apresenta Kerrighed como um sistema operacional SSI para computação de alto desempenho. Ainda neste trabalho, demonstra-se a execução com sucesso de programas com múltiplas *threads* e aplicações MPI. Lottiaux et al. (LOTTIAUX et al., 2005) apresentaram um estudo comparativo entre openMosix, OpenSSI e Kerrighed, onde diversos critérios foram analisados. Nesta comparação, Kerrighed apresentou o melhor desempenho em migrações de processos tanto na latência quanto largura de banda. Além disso, é possível utilizar o escalonador do Kerrighed para gerenciar a distribuição dos processos MPI, conforme descrito na página Web do projeto (KERRIGHED, 2006). No entanto, não foram encontrados trabalhos descrevendo experiências de migração de processos propriamente dita em aplicações MPI.

2.4.3 Migração Baseada em Máquinas Virtuais

A virtualização de recursos computacionais consiste em abstrair o *hardware* subjacente através da criação de uma interface de máquina virtual, que representa recursos virtualizados, tais como CPU, memória física, dispositivos de armazenamento, etc. Todas as aplicações, incluindo o próprio sistema operacional executam sobre esta abstração de máquina virtual. Desde que várias máquinas virtuais podem existir sobre um mesmo recurso, a utilização de virtualização permite a execução de diversas instâncias de sistemas operacionais sobre uma mesma arquitetura de *hardware* de forma transparente e isolada.

Uma vantagem da utilização de virtualização, particularmente importante no contexto deste trabalho, é a capacidade de migração de máquinas virtuais (CLARK et al., 2005; NAGARAJAN et al., 2007). A migração do sistema operacional inteiro, com toda a imagem da memória, evita os problemas clássicos que envolvem a migração em nível de processo, tais como a dependência residual e o re-estabelecimento de conexões (MILOJICIC et al., 2000). No que diz respeito à migração de processos MPI, as conexões de rede entre os nós são um ponto crítico, fazendo com que seja necessário protocolos complexos para obtenção de um estado global consistente e esvaziamento dos canais de comunicação antes de efetuar a migração de um processo de um nó físico para outro (conforme discutido na Seção 2.4.1). No caso de migração de máquinas virtuais, processos MPI podem ser migrados de forma transparente, já que toda a pilha de protocolos é migrada junto com

o sistema operacional para um nó destino.

A seguir são apresentados os sistemas de virtualização disponíveis atualmente que possuem suporte a migração.

2.4.3.1 Xen

Xen (BARHAM et al., 2003) é um sistema de virtualização, com suporte a diversas arquiteturas, que permite a criação e execução de múltiplas máquinas virtuais, simultaneamente, sobre um mesmo *hardware*. Xen foi originalmente desenvolvido como um projeto de pesquisa na Universidade de Cambridge e atualmente é mantido pela empresa Citrix System Inc., com código-fonte aberto. Entre as principais características de Xen, podem-se citar o suporte a migração de máquinas virtuais sem interromper a execução das mesmas, também conhecido como *live migration*.

A arquitetura de Xen é composta por dois elementos principais: (i) um monitor de máquinas virtuais (MMV), chamado de *hypervisor*; (ii) e as máquinas virtuais que são controladas pelo monitor (chamadas de domínios Xen). O MMV abstrai a camada de *hardware* e provê acesso para os diferentes domínios. Um domínio especial é aquele chamado Domain0, o qual é capaz de acessar diretamente a interface de controle do MMV. Através desse acesso, é possível criar e gerenciar outros domínios Xen, chamados DomainU.

A migração de Xen ocorre de forma iterativa. Inicialmente é feita uma reserva de recursos no computador destino. Caso sejam atendidos alguns requisitos mínimos de compatibilidade, inicia-se a transferência da máquina virtual para um novo destino. Na primeira iteração, são transferidas todas as páginas de memória e, nos demais turnos, as páginas modificadas são retransmitidas. Além disso, Xen implementa um algoritmo para limitar o número de turnos e melhorar o desempenho da migração.

Existem alguns trabalhos explorando o uso de Xen para migrar processos em aplicações MPI. O trabalho de Boufleur et al. (BOUFLEUR; KOSLOVSKI; CHARÃO, 2006) apresenta resultados experimentais sobre o custo de utilização de Xen para a execução de aplicações paralelas e distribuídas. Neste artigo, foram realizadas migrações de processos em uma aplicação MPI residente em diferentes máquinas virtuais. Explorando a união de Xen e MPI, o trabalho desenvolvido por Youseff et al. (YOUSEFF et al., 2006) realizou uma análise completa da utilização de Xen em ambiente de alto desempenho.

2.4.3.2 KVM

KVM (*Kernel-based Virtual Machine*) (KVM, 2009) é uma solução completa para virtualização em *hardware* x86 com extensões de virtualização nativa (tais como as tecnologias Intel VT e AMD-V). Usando KVM é possível criar e executar múltiplas máquinas virtuais como se fossem processos normais do Linux. Além disso KVM possui suporte à migração de máquinas virtuais sem interromper execução das mesmas (*live migration*).

A implementação do suporte a virtualização no *kernel* do Linux consiste em dois módulos carregáveis: `kvm.ko`, que provê a infraestrutura de virtualização; e um específico para a família do processador, `kvm-intel.ko` ou `kvm-amd.ko`. A execução da máquina virtual é realizada sobre uma versão modificada do sistema QEMU (QEMU, 2009) – um emulador de processador genérico – e a interface dessa máquina virtual com o kernel é realizada através de um dispositivo de caractere, `/dev/kvm`, que permite a execução da máquina virtual em nível de usuário.

O suporte a migração de KVM é implementado dentro de QEMU. A migração inicia com a criação de uma instância de QEMU, no nó destino, que irá receber as páginas de

memória da máquina virtual de forma iterativa. O algoritmo de transferência é semelhante ao de Xen, com a diferença que as páginas são enviadas uma por vez. Embora KVM exija que o *hardware* subjacente possua suporte a virtualização nativa, é possível migrar máquinas virtuais entre nós com tecnologias diferentes. Por exemplo, é possível migrar uma máquina virtual que executa sobre uma plataforma AMD para um destino de plataforma Intel.

KVM ainda está em desenvolvimento. A versão inicial de KVM foi liberada em 2006 e a funcionalidade de *live migration* só foi disponibilizada após o início deste trabalho. O trabalho de Deshane et al. (DESHANE et al., 2008) apresenta uma comparação de desempenho entre KVM e Xen, na qual ambos apresentam um desempenho semelhante. No entanto, não foram encontrados trabalhos avaliando a migração de máquinas virtuais ou relatando a utilização de KVM para executar programas MPI.

2.4.4 Outras Abordagens

Além das soluções apresentadas até o momento, existem outras abordagem que, apesar de não se encaixarem diretamente nas categorias anteriores, são bastante representativas para o estado da arte. Um exemplo dessas soluções é Adaptive MPI (HUANG; LAWLOR; KALÉ, 2003), que utiliza uma abordagem de migração de *thread* para prover o efeito de migração de processos. Esta solução é apresentada a seguir.

2.4.4.1 Adaptive MPI

Adaptive MPI (AMPI) (HUANG; LAWLOR; KALÉ, 2003) é uma implementação do padrão MPI que utiliza o *framework* orientado a objetos Charm++ para oferecer balanceamento de carga dinâmico em aplicações. Esse *framework* utiliza o conceito de processadores virtuais (VP). Em Charm++, cada VP é implementado como um objeto paralelo migrável e a comunicação entre os objetos é realizada através de chamadas remotas de métodos. AMPI faz uso desta abordagem para implementar processos MPI como *threads* de usuário embutidas em objetos paralelos de Charm++. Desta forma, cada processo MPI executa sobre um VP e pode ser migrado de um VP para outro.

A abordagem de migração de *threads* é vantajosa, já que tende a ser menos custosa que a migração de processos. Além disso, para migrar *threads* em nível de usuário não é necessário suporte do sistema operacional. No entanto, existe a necessidade de garantir que qualquer referência à memória continuará válida após a migração para um nó destino.

Na biblioteca AMPI, existem duas formas de se manter válidas as referências após uma migração. A primeira delas envolve a alocação de um mesmo intervalo de endereçamento virtual em cada processador. O tamanho desta área alocada é dependente do número de processadores, podendo tornar inviável a solução quando a quantidade deles for muito grande. Utilizado este método é possível mover os dados da pilha (*stack*) inteiramente para o nó destino e mapear para o mesmo endereço usando `mmap`. No caso dos dados alocados dinamicamente (*heap*), todas as chamadas de `malloc()` devem ser redefinidas por uma diretiva especial de AMPI chamada `isomalloc()`, que garante a alocação no endereço correto. Variáveis globais alocadas estaticamente (*heap*) também precisam ser tratadas para se encaixarem em um dos casos anteriores. Alternativamente, é possível definir manualmente funções para empacotamento e desempacotamento de dados migráveis.

Existem diversos trabalhos explorando o suporte de migração de processos de AMPI. No trabalho de Huang et al. (HUANG et al., 2006), por exemplo, foi realizado um estudo do desempenho de AMPI executando aplicações MPI, inclusive com migração de pro-

cessos para realizar balanceamento de cargas. Apesar de AMPI apresentar uma solução funcional para migração de processos MPI, existe a necessidade do programador modificar o código fonte da aplicação para torná-la migrável, o que pode dificultar a migração de processos em aplicações legadas.

2.5 Conclusão

No decorrer desse capítulo foram introduzidos os conceitos e problemas envolvendo migração de processos em aplicações MPI. Também, foram apresentadas as principais soluções de migração disponíveis atualmente. Essas soluções foram classificadas em três categorias principais de acordo com a abordagem utilizada: baseadas em C/R, baseadas em SSI e baseadas em MVs.

As soluções baseadas em C/R são normalmente desenvolvidas para tolerância a falhas, mas podem ser adaptadas para prover migração de processos. MPICH-V possui suporte a C/R funcional mas, apesar de ser bastante utilizado em tolerância a falhas, não foram encontrados trabalhos utilizando esse suporte para migrar processos MPI. LAM/MPI também possui um suporte a C/R funcional e já foi utilizado para realizar migração de processos (WANG et al., 2008, 2007; WALTERS; CHAUDHARY, 2007; CAO; LI; GUO, 2005). No entanto, foi anunciado recentemente que os esforços de desenvolvimento foram transferidos para Open MPI. O suporte a C/R de Open MPI, por sua vez, foi recentemente lançado como uma evolução da solução de LAM/MPI.

As soluções de agregado SSI normalmente incluem migração de processo como ferramenta para prover a ilusão de uma máquina única. No entanto, os sistemas SSI possuem o inconveniente de assumirem um ambiente de agregado fortemente acoplado. Além disso, foram encontrados poucos trabalhos avaliando esse tipo de solução na execução de aplicações MPI. Entre as soluções estudadas, openMosix apresentou o problema da dependência residual após a migração de processos que realizam comunicações. OpenSSI e Kerrighed são as soluções mais promissoras, mas não foram encontrados trabalhos aplicando-os na migração de processos MPI especificamente.

As soluções de virtualização historicamente impõem um sobrecusto. No entanto, trabalhos recentes (DESHANE et al., 2008; BOUFLEUR; KOSLOVSKI; CHARÃO, 2006; YOUSEFF et al., 2006) revelam as ferramentas Xen e KVM como soluções de virtualização com desempenho bastante próximo ao sistema nativo. Além disso, a utilização de migração de máquinas virtuais evita os problemas clássicos da migração, como a dependência residual e o re-estabelicimeno de conexões. Xen mostra-se como a solução mais madura e viável para migração de processos MPI, sendo utilizado em diversos trabalhos.

Também foi apresentada a solução AMPI, que não pôde ser classificada diretamente nas categorias anteriores, utilizando a abordagem de migração de *threads* para simular a migração de processos MPI. AMPI é uma solução funcional e é atualmente utilizada para prover balanceamento de cargas em programas MPI (HUANG et al., 2006; HUANG; LAWLOR; KALÉ, 2003). No entanto, AMPI impõe a necessidade de modificar o código fonte das aplicações para torná-las migráveis. Assim, AMPI difere das outras soluções estudadas que permitem a migração de processos de forma transparente à aplicação.

Neste contexto, entre as abordagens estudadas, as que se mostraram mais adequadas para migração de processos MPI, foram as soluções baseadas em C/R e baseada em MVs. Estas abordagens também são as que apresentam trabalhos mais recentes. Desde o início deste trabalho, foram lançados o suporte a C/R de Open MPI e o suporte a *live migration* de KVM. Nos próximos capítulos estas abordagens são estudas em mais detalhes e as so-

luções de migração são analisadas. O Capítulo 3 estuda a utilização de C/R para migração de processos MPI, descrevendo como Open MPI pode ser utilizada para migrar processo MPI. O Capítulo 4 estuda a migração de máquinas virtuais e descreve a utilização de Xen para migrar processos MPI.

3 *CHECKPOINT/RESTART* PARA MIGRAÇÃO DE PROCESSOS MPI

O objeto de estudo principal deste capítulo é a utilização de C/R para migrar processos MPI. Primeiramente, são apresentadas as abordagens para salvamento de contexto, as quais são utilizadas por soluções de migração, e os conceitos de *checkpoint* distribuído, utilizado para garantir que a aplicação MPI continuará sua execução de forma consistente após uma migração. Após, são estudadas as soluções de C/R que podem ser utilizadas para migrar processos. A solução escolhida para ser estudada em mais detalhes foi a de Open MPI. O suporte a C/R de Open MPI é baseado na implementação de LAM/MPI, que já foi utilizada com sucesso para migrar processos MPI. Além disso, foi anunciado recentemente que os esforços de desenvolvimento de LAM/MPI foram transferidos para Open MPI, o que justifica a escolha desta solução ao invés da primeira.

Conforme discutido no Capítulo 2, tanto o suporte a C/R de LAM/MPI quando de Open MPI foram projetados inicialmente para tolerância a falhas. Desta forma, é necessário adaptar estas soluções para permitir a realização de migração de processo. Este capítulo descreve as modificações realizadas para viabilizar a migração.

O texto deste capítulo está organizado da seguinte forma. A Seção 3.1 apresenta uma discussão sobre os mecanismos para salvamento de contexto. A Seção 3.2 apresenta o conceito de *checkpoint* distribuído. A Seção 3.3 apresenta o suporte a C/R da LAM/MPI e a forma como este pode ser utilizado para prover migração. O conhecimento da implementação de C/R em LAM/MPI, e suas limitações, contribui para o entendimento da solução proposta em Open MPI. A Seção 3.4 apresenta o suporte a C/R de Open MPI e a forma como este é utilizado, neste trabalho, para realizar migração de processos. Por fim, a Seção 3.5 faz um resumo dos principais pontos discutidos no decorrer de capítulo.

3.1 Mecanismo de *Checkpoint/Restart*

O armazenamento do contexto de execução é, por si só, um domínio de pesquisa. Entre as soluções de migração de processo baseados em C/R, observa-se que a funcionalidade de C/R é, em todos os casos, um módulo à parte do projeto. Além disso, alguns destes projetos utilizam mecanismos de terceiros, prontos e já validados. Desta forma, antes de estudar a fundo as soluções para migração de processos, é importante uma discussão sobre alguns destes mecanismos de C/R em que estas soluções se baseiam.

As implementações de C/R podem ser classificadas em três classes (ROMAN, 2002), de acordo com o nível de implementação: implementadas na própria aplicação, através de uma biblioteca ligada à aplicação ou no núcleo do sistema operacional.

Mecanismos de C/R implementados em nível de aplicação tendem a prover maior

eficiência, uma vez que o programador da aplicação possui o conhecimento de quais as estruturas de dados que precisam ser salvas, e quais podem ser descartadas. No entanto, esta abordagem possui algumas desvantagens, como o custo de implementação de uma solução específica 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.

Mecanismos de C/R implementados em nível de biblioteca podem resolver alguns desses problemas. O uso de bibliotecas pode eliminar a necessidade de modificações no código da aplicação. Além disso, implementações 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 a informações internas ao sistema operacional, são normalmente perdidas.

Mecanismos de C/R implementados em nível de sistema 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 realizar *checkpoint* a qualquer instante.

A seguir são apresentadas algumas das implementações de C/R mais difundidas:

- **Libckpt** (PLANK et al., 1995) é uma implementação em nível de biblioteca e, portanto, possui todas as limitações típicas deste tipo de solução. 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 páginas de memória que foram modificadas desde o último salvamento.

- **Condor** (LITZKOW et al., 1997) é outro sistema que provê serviços de *checkpoint* de processos. O módulo de *checkpoint* de Condor é transparente e totalmente implementado em nível de usuário. A utilização de Condor requer que a aplicação seja ligada a uma biblioteca especial, mas nenhuma modificação do código é necessária.

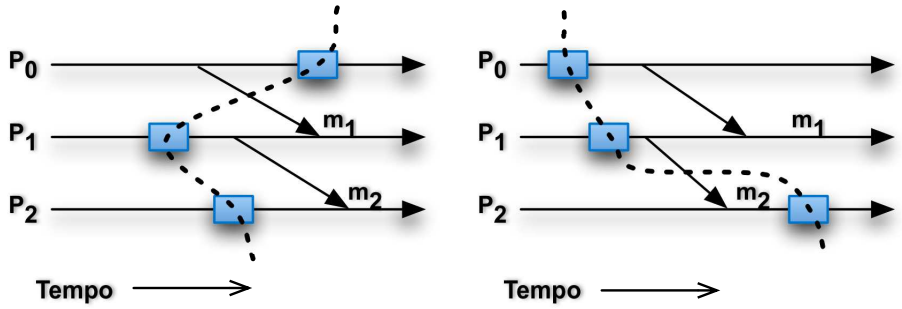
- **BLCR** (*Berkeley Lab's Checkpoint/Restart*) (DUELL; HARGROVE; ROMAN, 2002) é uma implementação robusta de C/R em nível de sistema. BLCR é implementado como um módulo do *kernel* Linux e uma biblioteca de nível de usuário. A biblioteca permite que aplicações e outras bibliotecas possam interagir com o módulo de C/R. 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*) (ZHONG; NIEH, 2001) 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* Distribuído e Migração de Processos

Conforme mencionado na Seção 2.3, a migração de processos em uma aplicação MPI é mais difícil que em processos individuais devido ao forte acoplamento de MPI. Em uma aplicação paralela MPI, os processos cooperam através de trocas de mensagens. A execução destes processos pode ser vista como uma sequência ordenada de eventos como, por exemplo, o envio e recebimento de mensagens. Cada evento muda o estado de seu processo. Desta forma, o conjunto formado pelos estados locais de todos os processos e seus canais de comunicação representa o estado global da aplicação (SANKARAN et al., 2005).

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 (CHANDY; LAMPORT, 1985). A Figura 3.1 contém dois exemplos de estado global em uma aplicação formada por três processos (P_0, P_1 e P_2) trocando mensagens (m_1 e m_2): (a) mostra um exemplo de estado global consistente onde a mensagem m_1 é salva como enviada pelo processo P_0, embora ainda não seja recebida pelo processo P_1, já que o envio ocorre cronologicamente antes do recebimento; e (b) mostra um exemplo de estado global inconsistente onde a mensagem m_1 é registrada como recebida pelo processo P_1, mas ainda não enviada pelo processo P_0. 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 de forma correta.



(a) (b)

Figura 3.1 Obtenção de estado global consistente.

Os mecanismos de C/R para MPI utilizam diferentes abordagens para alcançar um estado global consistente. As principais são *checkpoint* coordenado e não coordenado. No ***checkpoint* não coordenado**, o salvamento do contexto de execução é realizado de forma independente por cada processo. Neste caso, é necessário armazenar as mensagens recebidas desde o último *checkpoint* para serem consultadas na ocasião de um *restart*. O estado global consistente é, portanto, conhecido somente no *restart*. No caso de ***checkpoint* coordenado** existe uma interação entre os processos de forma a garantir um estado global consistente antes de iniciar o salvamento do contexto. Normalmente, mensagens especiais são trocadas entre os processo para forçar o esvaziamento dos canais de comunicação. Conforme apresentado no Capítulo 2, tanto LAM/MPI quando Open MPI utilizam a abordagem de *checkpoint* coordenado.

3.3 Migração de Processos em LAM/MPI

LAM/MPI não oferece nativamente suporte a migração, mas o seu suporte a C/R pode ser utilizado para migrar processos. A Seção 2.4.1.2 apresentou a arquitetura de LAM/MPI organizada em duas camadas: a camada LAM e a camada MPI. A Figura 3.2 ilustra esta organização e a sua relação com o *framework* SSI, o qual permite a criação de diferentes tipos de componentes ou módulos para prover funcionalidades/serviços únicos. Cada tipo de componente pode ter uma ou mais instâncias, que podem ser selecionadas em tempo de execução. Um exemplo de componente SSI é o RPI, que implementa a comunicação ponto-a-ponto dependente de dispositivo. O módulo RPI pode ter diferentes instâncias, tais como TCP, gm, memória compartilhada, etc.

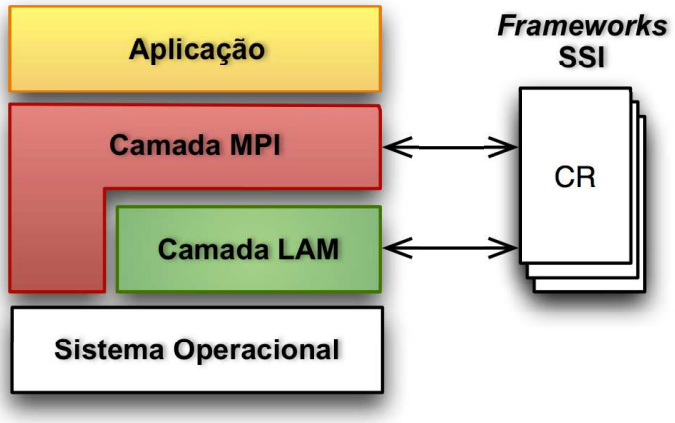


Figura 3.2 Arquitetura de LAM/MPI.

O suporte a C/R de LAM/MPI é implementado como um módulo SSI, chamado CR. Atualmente, o módulo CR possui uma implementação única, que utiliza BLCR. Para habilitar o suporte a C/R é necessário selecionar o módulo CR no comando `mpirun`, que é disponibilizado por LAM/MPI para realizar o lançamento da aplicação.

O procedimento de C/R inicia com uma solicitação realizada pelo usuário. O ponto de entrada das solicitações de C/R é `mpirun`, Os passos do procedimento de *checkpoint* são listados a seguir.

1. `mpirun` recebe uma solicitação de *checkpoint* e propaga-a para todos os processos da aplicação.
2. Cada processo, por sua vez, negocia com os outros processos para atingirem um estado global consistente.
3. Uma vez alcançado um estado consistente, cada processo solicita o salvamento de seu contexto.
4. O mecanismo de C/R, neste caso BLCR, salva o contexto de cada processo.
5. Processos MPI continuam sua execução normal.
6. Neste ponto, `mpirun` se prepara para uma possível recuperação e indica que está pronto para salvar seu contexto.

7. Então, o contexto de `mpirun` é salvo por BLCR.

A partir dos arquivos de contexto dos processos MPI e de `mpirun` é possível recuperar a execução da aplicação (*restart*). Os passos do procedimento de recuperação são listados a seguir.

1. `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.

2. Cada processo, após ser reiniciado, envia suas novas informações de processo para `mpirun`.

3. `mpirun`, por sua vez, constrói a tabela global, que contém as informações sobre todos os processos da aplicação, e propaga-a para todos os processos.

4. Os processos, após receberem estas informações, reconstroem os canais de comunicação com os outros processos.

5. Após esse procedimento, a aplicação continua sua execução normal.

A migração de processos pode ser alcançada através da modificação da localização física dos processos na ocasião da recuperação. Conforme apresentado no capítulo anterior, existem alguns trabalhos na literatura propondo soluções de migração utilizando o suporte a C/R de LAM/MPI (CAO; LI; GUO, 2005; WANG et al., 2007; WALTERS; CHAUDHARY, 2007). Neste trabalho, optou-se por reproduzir a solução proposta por Cao et al. (CAO; LI; GUO, 2005) que modifica os arquivos gerados por BLCR.

A **migração de processos** utilizando esta abordagem consiste em três fases. Primeiramente é realizado o *checkpoint* da aplicação, seguido da transferência dos arquivos de contexto dos processos migrantes para seus nós destinos. Na sequência, modificam-se as informações de localização nos arquivos de contexto e, então, continua-se a execução da aplicação a partir destes arquivos modificados. Cada processo em LAM/MPI possui uma estrutura `struct _gps` que contém as informações de localização (por exemplo, identificação do processo e do nó). Estas informações são utilizadas por LAM/MPI para reiniciar os canais de comunicação após uma operação de C/R. Desta forma, a modificação do arquivo de contexto consiste em localizar a estrutura `_gps` e alterar as informações para indicar o novo nó. Também é necessário modificar a tabela de mapeamento presente no arquivo de contexto de `mpirun`. Com essas modificações é possível continuar a execução da aplicação corretamente em um conjunto diferente de nós, caracterizando uma migração de processos.

Esta abordagem foi implementada e verificada na prática no agregado do grupo GPPD, sendo realizada migração de processos com sucesso. Apesar de funcional, esta solução apresenta um custo elevado devido a necessidade de processar os arquivos de contexto, conforme descrito no trabalho de Cao et al. (CAO; LI; GUO, 2005). Além disso, os esforços de desenvolvimento de LAM/MPI foram migrados para Open MPI, inclusive com uma implementação de C/R para esta distribuição.

3.4 Migração de Processo em Open MPI

Assim como LAM/MPI, Open MPI não possui suporte nativo a migração, mas seu suporte a C/R pode ser utilizado para prover esta funcionalidade. A Seção 2.4.1.3 apresentou a arquitetura de Open MPI em três camadas de abstração: OMPI, ORTE e OPAL. A Figura 3.3 ilustra essas camadas e a sua relação com os *frameworks* MCA (*Modular Component Architecture*), que permite organizar as funcionalidades de Open MPI em componentes. Cada camada da arquitetura possui *frameworks* MCA para definir grupos de funcionalidades (comunicação ponto-a-ponto, por exemplo). Cada *framework* pode possuir diferentes implementações na forma de componentes/módulos (suporte a TCP/IP, por exemplo).

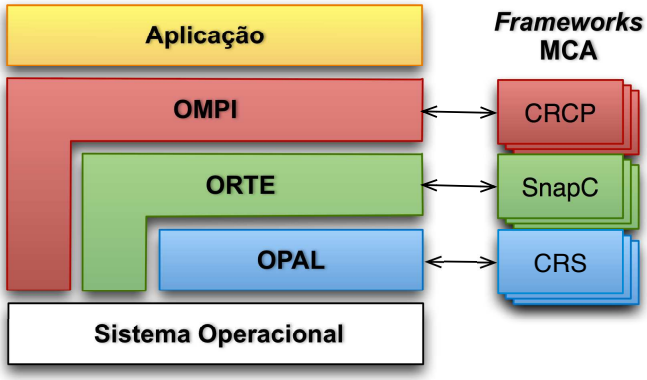


Figura 3.3 Arquitetura de Open MPI.

O suporte a C/R de Open MPI é organizado em três *frameworks* MCA – SnapC, CRCP e CRS – ligados às diferentes camadas de Open MPI. O *framework* SnapC (*Snapshot Coordinator*) atua na camada ORTE e coordena a obtenção do *checkpoint* distribuído. Entre as atividades do SnapC estão iniciar o *checkpoint* de cada processo local, monitorar o progresso do *checkpoint* global e gerenciar os arquivos de contexto gerados. O componente inicial que implementa o *framework* SnapC utiliza uma abordagem centralizada, similar ao suporte a C/R de LAM/MPI. O *framework* CRCP (*Checkpoint/Restart Coordination Protocol*) é responsável pela obtenção do estado global consistente. Como esta tarefa envolve o estado dos canais de comunicação da aplicação, CRCP atua diretamente na camada OMPI. O *framework* CRS (*Checkpoint/Restart Service*) atua na camada OPAL e é responsável pelo C/R dos processos locais. O componente inicial implementa uma interface com a ferramenta BLCR.

O procedimento de C/R inicia com uma solicitação realizada pelo usuário. Da mesma forma como ocorre em LAM/MPI, o ponto de entrada das solicitações de C/R é `mpirun`. A Figura 3.4 ilustra os passos do procedimento de *checkpoint* em uma aplicação MPI que executa em três nós. Estes passos são listados a seguir.

1. `mpirun` recebe uma solicitação de *checkpoint* e propaga-a para todos os nós da aplicação.

2. Em cada nó, as camadas OPAL, ORTE e OMPI são informadas da solicitação e inicia-se a preparação para o *checkpoint*.

3. CRCP negocia um estado global consistente com todos os processos.

4. CRS utiliza BLCR para salvar o contexto dos processos locais.

5. SnapC centraliza todos os arquivos de contexto e as informações necessárias para a recuperação da execução (metadados) em um servidor de armazenamento persistente.

6. CRCP normaliza as operações na camada OMPI e os processos MPI continuam sua execução normal. Por fim, o usuário recebe uma referência do contexto de execução da aplicação.

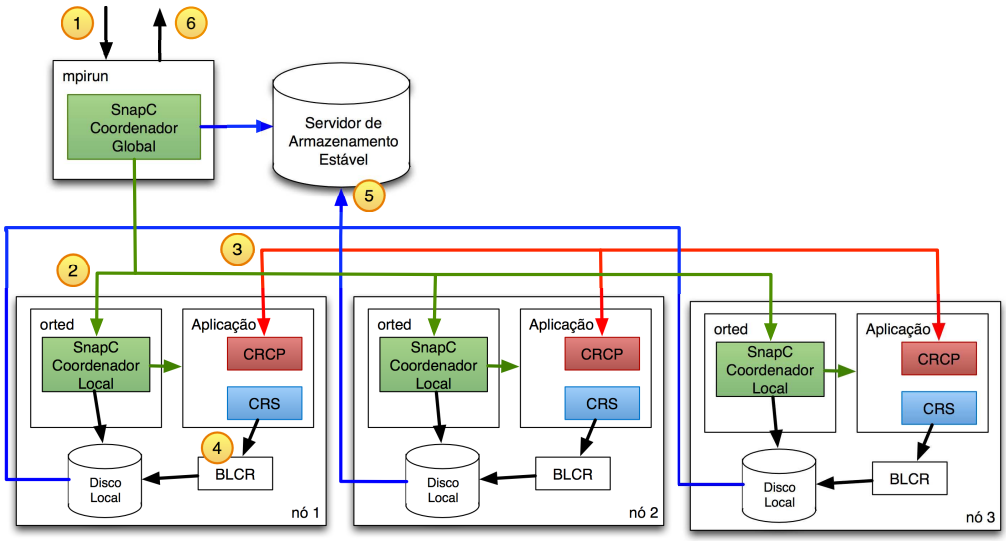


Figura 3.4 Operação de *checkpoint* em Open MPI.

A partir da referência retornada no *checkpoint*, é possível recuperar a execução da aplicação (*restart*). Os passos do procedimento de recuperação são listados a seguir.

1. `mpirun` inicia a sua execução através da referência retornada pelo *checkpoint* e utiliza os metadados para solicitar a recuperação de todos os processos da aplicação.

2. SnapC solicita a transferência dos arquivos de contexto do servidor persistente para os nós da aplicação.

3. CRS recupera a execução de cada processo a partir de seus arquivos de contexto.

4. CRCP informa aos componentes de comunicação da camada OMPI que se trata de uma operação de *restart* e os canais de comunicação entre os processos são recriados.

5. Após esse procedimento, a aplicação continua sua execução normal.

Uma das principais diferenças em relação à implementação de C/R de LAM/MPI é a possibilidade de reiniciar a aplicação em um novo conjunto de nós, sem a necessidade de modificar informações de localização nos arquivos de contexto. Desta forma, é possível obter o efeito de **migração de processos** passando um novo arquivo de máquinas para retomar a execução da aplicação com um novo mapeamento.

Outra diferença significativa é a utilização de servidores de armazenamento persistente. A implementação inicial de SnapC realiza a transferência dos arquivos de contexto para um servidor de armazenamento, conforme ilustrado no passo 5 da Figura 3.4. O objetivo deste procedimento é prover tolerância a falhas, mas o custo das transferências de arquivos torna inviável a utilização deste mecanismo como solução de migração de processos.

Neste contexto, foi criado uma novo componente para SnapC com o objetivo de reduzir o custo das operações de C/R e viabilizar a migração de processos. Nesta nova implementação, a coordenação do *checkpoint* distribuído não envolve a transferência de arquivos de contexto, apenas metadados são trafegados pela rede. Desta forma, eliminou-se o passo 5 da Figura 3.4 e reduziu-se significativamente o custo das operações de C/R. Esta solução está funcional e foi utilizada com sucesso, no agregado do grupo GPPD, para migrar processos em aplicações MPI. Uma avaliação de desempenho dessa modificação é apresentada no Capítulo 7.

3.5 Conclusão

Este capítulo explorou a utilização de C/R para migrar processos MPI. Em particular, as soluções de C/R de LAM/MPI e Open MPI foram estudadas. Ambas as soluções são funcionais e foram testadas na prática no agregado do grupo GPPD para migrar processos em aplicações MPI.

Apesar de funcional, a migração de processos em LAM/MPI apresenta um custo elevado devido à necessidade de processar os arquivos de contexto. Além disso, os esforços de desenvolvimento de LAM/MPI foram migrados para Open MPI.

Open MPI, por outro lado, permite reiniciar a aplicação com um novo mapeamento sem a necessidade de modificar os arquivos de contexto. A implementação inicial de C/R para Open MPI envolve a transferência dos arquivos de contexto para um servidor de armazenamento persistente. Como o custo deste procedimento inviabiliza a migração de processos, optou-se por modificar esta solução para permitir a migração de processos. O Capítulo 7 apresenta uma avaliação de desempenho do suporte a migração implementado.

4 MIGRAÇÃO DE MÁQUINAS VIRTUAIS

Este capítulo trata da virtualização de recursos computacional e a utilização de migração de máquinas virtuais para migrar processos em aplicações MPI. A solução escolhida para ser estudada em maiores detalhes foi Xen. Este sistema destaca-se entre os demais por implementar migração de MVs do tipo *live migration* e ser uma solução estável e amplamente difundida. Além disso, trabalhos recentes analisam o desempenho de Xen e relatam um desempenho bastante próximo ao do sistema nativo.

O texto deste capítulo está organizado da seguinte forma. Primeiramente, a Seção 4.1 apresenta alguns conceitos de virtualização e sua influência no desempenho do ambiente virtualizado. Na sequencia, a Seção 4.2 descreve o suporte a migração de MV de Xen e como este pode ser utilizado para migrar processos MPI. Por fim, a Seção 4.3 traça um resumo dos principais pontos discutidos no decorrer do capítulo.

4.1 Monitores de Máquinas Virtuais

Soluções de virtualização envolvem basicamente o uso de uma camada de *software* que cria uma abstração de máquina virtual. Em ambientes virtualizados o gerenciamento dos recursos reais e implementação da abstração de máquinas reais são realizados por um monitor de máquinas virtuais, ou MMV. Devido a esta camada de *software*, as técnicas de virtualização historicamente impõem um sobrecusto de desempenho (GOTH, 2007). Com o intuito de reverter esse panorama, pesquisas recentes conseguem reduzi-lo, como é o caso da **paravirtualização** (BARHAM et al., 2003). O restante desta seção apresenta as principais abordagens disponíveis para implementar a virtualização: virtualização completa, paravirtualização e virtualização assistida por *hardware*.

A virtualização completa (*full virtualization*) abstrai totalmente o *hardware* subjacente e oferece a ilusão de uma máquina virtual completa, sobre a qual um sistema operacional pode executar diretamente. Nenhuma modificação no sistema operacional hospedado, ou nas aplicações que executam sobre ele, é necessária. Esta abordagem pode ser vantajosa, pois permite desacoplar totalmente o *software* do *hardware* (por exemplo, um sistema operacional pode executar independente da arquiteturas de *hardware* da camada inferior). Por outro lado, a virtualização completa pode impor um elevado sobrecusto de desempenho, devido as camadas de abstração de *software*. Exemplos de MMVs que implementam a virtualização completa são Microsof Virtual Server e VMware ESX Server, ambos *softwares* proprietários.

A paravirtualização, ou virtualização assistida pelo sistema operacional, simplifica o processo de virtualização através da eliminação de funcionalidades específicas de *hardware* e instruções que são difíceis de serem virtualizadas com eficiência. Neste caso, é oferecida uma abstração de máquina virtual que é similar ao *hardware* da camada inferior,

mas não idêntica. Para poder executar corretamente sobre a máquina virtual, o sistema operacional hospedado precisa ser modificado. A principal vantagem desta abordagem é o desempenho bastante próximo ao de sistemas nativos. Exemplos de MMVs que utilizam paravirtualização são Denali (WHITAKER; SHAW; GRIBBLE, 2002) e Xen (BARHAM et al., 2003).

A virtualização assistida por *hardware* tira proveito das novas arquiteturas que incorporam funcionalidades de suporte a virtualização. A primeira geração de processadores com suporte a virtualização inclui as famílias VT (*Intel Virtualization Technology*) da Intel e AMD-V da AMD, ambas simplificam a virtualização através de um novo nível de execução, o qual permite a execução do MMV diretamente abaixo do nível normal de execução. Desta forma, o sistema operacional que executa em máquinas virtuais não precisa ser modificado. A principal desvantagem desta técnica é a necessidade de um *hardware* especial. Exemplos de MMVs que suportam esse tipo de virtualização são VMware, KVM e Xen.

Dentre os monitores MMV existentes (ADAMS; AGESEN, 2006), o sistema Xen (BARHAM et al., 2003) se destaca por permitir tanto paravirtualização quanto virtualização assistida por *hardware*. No caso da paravirtualização, é possivel conseguir um desempenho semelhante ao do sistema nativo. A seção a seguir descreve como Xen pode ser utilizado para migrar processos MPI.

4.2 Migração de Processos em Xen

Xen não oferece migração de processos propriamente dita, mas permite um efeito semelhante através da migração de MVs. A Seção 2.4.3.1 apresentou a arquitetura de Xen composta por dois elementos principais: (i) um monitor de máquinas virtuais (MMV) e (ii) as máquinas virtuais que são controladas pelo monitor (chamadas de domínios Xen). A Figura 4.1 ilustras esses elementos em um nó físico que executa duas máquinas virtuais. As máquinas virtuais de Xen são independentes do *hardware* existente e, assim, podem ser encapsuladas e posteriormente migradas para outro nó físico. Além disso, a migração de MV inteira evita os problemas clássicos de migração, conforme discutido na Seção 2.4.3.

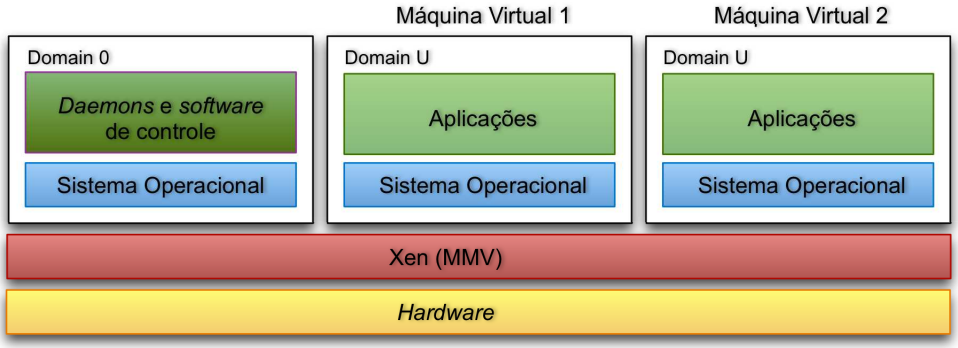


Figura 4.1 Arquitetura de Xen.

O procedimento de migração de uma máquina virtual é ilustrado pela Figura 4.2. Esta figura representa um agregado formado por dois nós, que hospedam máquinas virtuais, sendo que cada MV executa alguns processos (p_1, p_2, p_3, etc.). A migração demonstrada

transfere a execução da MV 2 para para o nó 2, que possui os recursos disponíveis para esta ação, representados pela região pontilhada. O procedimento de migração inicia com uma solicitação realizada pelo usuário. O ponto de entrada da solicitação é o *daemon* de Xen (`xend`). Os passos da migração são listados a seguir.

1. `xend` local recebe uma solicitação de migração.
2. Inicia-se a negociação de migração com o `xend` do nó destino.
3. `xend` do nó destino verifica se existem recursos disponíveis para a operação e, em caso positivo, reserva esses recursos para receber a MV.
4. Inicia-se a transferência das páginas de memória da MV.
5. Após a transferência ser completada, a MV continua a execução no nó destino e os recursos são liberados no nó de origem.

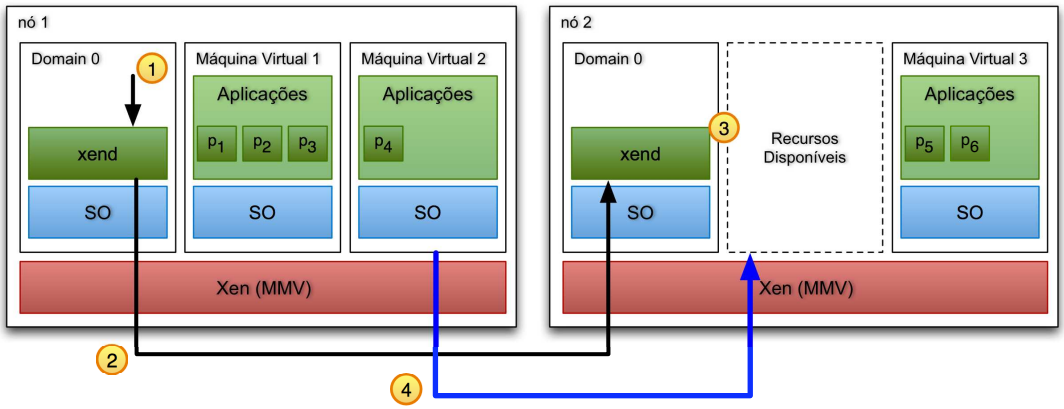


Figura 4.2 Migração de uma máquina virtual entre nós de um agregado.

Neste tipo de migração, todos os processos residentes na MV alvo são levados até o nó destino e seguem sua execução normalmente (sem uma reinicialização). Transferindo essa ideia para o contexto de migração de processo MPI, esta figura poderia representar, por exemplo, uma aplicação MPI com 6 processos, sendo que um deles (p_4) é migrado do nó 1 para o nó 2 através da migração de sua MV hospedeira (MV 2).

Como mencionado anteriormente, o mecanismo de migração implementado por Xen é do tipo *live migration* (CLARK et al., 2005). Para implementar esse tipo de migração, Xen utiliza uma a abordagem de transferência de páginas de memória chamada de pré-cópia (*pre-copy*). Essa abordagem combina as estratégias de uma fase iterativa de cópia sob demanda (*iterative push*) com uma fase curta de parada-e-cópia (*stop-and-copy*) (CLARK et al., 2005). O termo iterativo significa que a pré-cópia ocorre em turnos, sendo que no primeiro deles todas as páginas de memória são transferidas. Logo após, as páginas transferidas no turno r são aquelas que foram modificadas no turno anterior ($r-1$). No último turno, realiza-se a fase de parada-e-cópia, onde a MV interrompe sua execução no nó de origem, transmite as últimas páginas de memória restantes e continua a execução no nó destino.

Normalmente, as MVs possuem um conjunto de páginas que são modificadas frequentemente e que podem fazer com que o número de turnos seja muito grande. Assim,

é limitado o número de turnos de pré-cópia, baseado numa análise do comportamento do conjunto de páginas frequentemente modificadas (WWS - *Writable Working Set*) da MV. No caso de aplicações de alto desempenho, é de se esperar um WWS grande, devido sua natureza de computação intensiva, o que torna a velocidade de modificação superior à velocidade de transmissão das páginas. Este cenário, faz aumentar a duração da migração, alcançando o limite de turnos (CLARK et al., 2005).

A transferência das páginas de memória foi inicialmente projetada (CLARK et al., 2005) para respeitar uma taxa limite de vazão de rede. Deste modo, procura-se realizar uma migração que mantenha a execução das aplicações da MV sem uma degração de desempenho demasiada. Durante a configuração de Xen, o administrador deve especificar um limite de vazão máximo e mínimo (V_{min}, V_{max}). No primeiro turno da transferência, utiliza-se a vazão mínima e, a cada turno subsequente r, conta-se o número de páginas modificadas no turno anterior ($r-1$), e divide-se pelo tempo de duração. Desta forma, a taxa de modificação de páginas esta diretamente relacionada com o tamanho do WWS. Como resultado, obtém-se a taxa de modificação de páginas do turno anterior (*Taxa de modicação*$_{r-1}$), calculado através da Equação 4.1.

$$\textit{Taxa de modificação}_{r-1} = \frac{\textit{Número de páginas modificadas}_{r-1}}{\textit{Duração}_{r-1}} \tag{4.1}$$

A vazão limite para o próximo turno r é determinada dinamicamente, através da adição de um incremento constante K à taxa de modificação do turno anterior ($r-1$). Já no último turno (fase de parada-e-cópia), utiliza-se a vazão máxima (V_{max}) para diminuir o tempo de indisponibilidade da MV migrada. Desta forma, a Equação 4.2 permite calcular dinamicamente a vazão limite V_r utilizada em cada turno r da migração de uma MV.

$$V_r = \begin{cases} V_{min}, & r = 1 \\ V_{r-1} + (\textit{Taxa de modificação}_{r-1} * K), & 1 < r < \textit{último turno} \\ V_{max}, & \textit{último turno} \end{cases} \tag{4.2}$$

Apesar deste limite de vazão fazer parte do projeto original de *live migration* de Xen e estar documentado no artigo que apresentou o mecanismo (CLARK et al., 2005), observou-se que a implementação atual não leva em consideração esse limite e utiliza a vazão máxima disponível desde o primeiro turno. Nesta caso, a Equação 4.2 pode ser substituída por $V_r = V_{max}$.

É importante ressaltar também que o Xen não possui mecanismo para migração de sistema de arquivos. Portanto, quando uma máquina virtual é migrada é necessário que o sistema de arquivo esteja disponível no computador destino. Neste caso, normalmente utiliza-se um sistema de arquivos distribuído como é o caso do NFS. Já a migração de conexões é realizada através do envio de respostas ARP não solicitadas (*ARP reply*) (PLUMMER, 1982). Nesta técnica, todos os computadores da rede Ethernet são informados que o IP da máquina virtual migrada foi modificado para um novo local. Consequentemente, todos os processos com conexões remotas podem seguir executando normalmente e todo o mecanismo de gerenciamento de conexões é tarefa do Xen.

4.3 Conclusão

Este capítulo descreveu a utilização de virtualização e migração de máquinas virtuais para migrar processos em aplicações MPI. O sistema estudado foi Xen, que utiliza a

técnica de paravirtualização para oferecer um desempenho bastante próximo ao sistema nativo. A migração de máquinas MV oferecida por Xen é do tipo *live migration* e permite migrar processos de um nó para outro sem a necessidade de interromponder a execução da aplicação.

Utilizando esta abordagem de migração de MVs é possível obter o efeito de migração de processos. De fato, quando migra-se uma MV, todos os processos residentes nesta MV são levados até o nó destino e seguem sua execução normalmente. Esta migração ocorre com um tempo de indisponibilidade pequeno e todas as conexões de rede deles são mantidas de forma transparente usando o mecanismo de *ARP Reply*. Desta forma, evitam-se os problemas clássicos da migração a nível de processo. O Capítulo 7 apresenta uma avaliação do sobrecusto da virtualização na execução e migração de processos MPI.

5 MODELAGEM DO CUSTO DE MIGRAÇÃO

O objetivo deste capítulo é propor um modelo de custo de migração de processos para aplicações MPI. Através deste modelo e de execuções experimentais realizadas no Capítulo 7, pretende-se dimensionar o custo de migrar um processo MPI utilizando os mecanismos de migração disponíveis.

Primeiramente, define-se um modelo geral para descrever a execução de uma aplicação paralela com migração de processo levando em consideração as características de MPI. A partir deste modelo, são propostos modelos de custo de migração para cada uma das classes de soluções analisadas.

O texto do capítulo está organizado da seguinte forma. A Seção 5.1 apresenta a modelagem da execução de uma aplicação MPI que realiza migração de processo. Na sequência, são propostos modelos de custo para as duas abordagens de migração. A Seção 5.2 apresenta o modelo de custo de migração de processos baseada em C/R. Já a Seção 5.3 apresenta um modelo de custo de migração de processos baseada em migração de MVs. Por fim, a Seção 5.4 apresenta as considerações finais e um resumo sobre os modelos propostos.

5.1 Modelo de Execução de uma Aplicação MPI com Migração de Processos

Em uma aplicação MPI os processos executam uma carga de trabalho e comunicam dados através de troca de mensagens. Desta forma, o tempo total de execução da aplicação (T_{total}) pode ser dado, de maneira geral, por

$$T_{total} = T_w + T_c \tag{5.1}$$

onde T_w é o tempo de trabalho e T_c é o tempo gasto pelas comunicações bloqueantes.

Em caso de migração de processos, ao tempo total, acrescenta-se um custo de migração que é dependente do mecanismo utilizado. Este custo será representado por C_{ij} para uma migração de processo do nó de origem n_i para o nó destino n_j.

No instante que uma migração é iniciada, parte do processamento já foi realizado no nó de origem e o restante será realizado no nó destino. Neste caso, $\frac{\gamma W}{\nu_i}$ define o tempo de processamento que já foi realizado e $\frac{(1-\gamma)W}{\nu_j}$ define o processamento restante; onde W representa a carga de trabalho do processo, γ a fração de trabalho que já foi realizado até o momento da migração e ν_i e ν_j representam a capacidade de processamento dos nós n_i e n_j, respectivamente.

Da mesma forma, o custo de comunicação pode ser decomposto em duas partes quando uma migração é realizada. No entanto, diferentemente do processamento, o custo

de comunicação depende da vazão entre o nó em que o processo executa e cada um dos nós dos processos comunicantes. O conjunto de processos que trocam informação com o processo migrante será representado por N. Para cada processo k, o custo de comunicação antes e depois da migração pode ser definido como $\frac{\alpha B}{V_{ik}}$ e $\frac{(1-\alpha)B}{V_{jk}}$ respectivamente; onde B representa a quantidade de dados trocados com o processo k (em comunicações bloqueantes), α a fração de dados que já foram transmitidos até o momento da migração e V_{ik} e V_{jk} vazão de comunicação do nó n_i e n_j com o nó que executa o processo k. O custo total de comunicação de um processo é obtido pelo somatório do custo de comunicação com cada processo k pertencente ao conjunto N.

Portanto, o tempo total de execução levando em consideração uma migração de processos pode ser definido, de maneira geral, por

$$T_{total} = \frac{\gamma W}{\nu_i} + \frac{(1-\gamma)W}{\nu_j} + \sum_{k \in N} \left(\frac{\alpha_k B_k}{V_{ik}} + \frac{(1-\alpha_k)B_k}{V_{jk}} \right) + C_{ij} \tag{5.2}$$

Em muitos casos, o ambiente de execução é formado por recursos homogêneos e as capacidades de processamento e vazão da rede podem ser consideradas, respectivamente, como $\nu_i = \nu_j$ e $V_{ik} = V_{jk}$. Neste caso, o tempo total de execução com uma migração passa a ser definido como

$$T_{total} = \frac{W}{\nu} + \sum_{k \in N} \frac{B_k}{V} + C_{ij} \tag{5.3}$$

A seguir são apresentadas as modelagens de custo de migração C_{ij} para um mecanismo baseado em *checkpoint/restart* e um baseado em migração de máquinas virtuais.

5.2 Modelo de Custo de Migração de Processos Baseada em C/R

A migração baseada em C/R normalmente possui três fases bem definidas: *checkpoint* (aquisição e possível gravação dos dados), transmissão pela rede e *restart* (recuperação da execução). Desta forma, para um mecanismo genérico de migração baseada em C/R, sem qualquer tipo de otimização, o custo de migrar um processo do nó n_i para o nó n_j pode ser dado por

$$C_{ij} = Check(S) + Trans(S, V) + Restart(S) \tag{5.4}$$

onde $Check(S)$, $Trans(S, V)$ e $Restart(S)$ representam os custos de *checkpoint*, transmissão e *restart* em função do tamanho S da imagem em memória do processo e da vazão V da rede entre os nós n_i e n_j.

O custo de cada uma dessas fases é altamente dependente do mecanismo de migração utilizado. No caso do suporte a C/R da Open MPI, discutido no Capítulo 3, o custo de migração pode ser definido como

$$C_{ij} = \tau S + \psi \frac{S}{V_{ij}} + \beta S \tag{5.5}$$

onde τ representa o sobrecusto envolvido na operação de *checkpoint*, ψ o sobrecusto na transmissão e β o sobrecusto na recuperação. Substituindo esse custo na Equação 5.2 obtém-se

$$T_{total} = \frac{\gamma W}{\nu_i} + \frac{(1-\gamma)W}{\nu_j} + \sum_{k \in N} \left(\frac{\alpha_k B_k}{V_{ik}} + \frac{(1-\alpha_k)B_k}{V_{jk}} \right) + \psi \frac{S}{V_{ij}} + (\tau + \beta)S \quad (5.6)$$

para representar o tempo total de execução com uma migração baseada em C/R. Da mesma forma, a substituição do custo de migração na Equação 5.3 resulta em

$$T_{total} = \frac{W}{\nu} + \frac{\sum_{k \in N} B_k}{V} + \psi \frac{S}{V} + (\tau + \beta)S \quad (5.7)$$

que representa o tempo total de execução com uma migração baseada em C/R em um ambiente de execução formado por recursos homogêneos.

5.3 Modelo de Custo de Migração de Processos Baseada em MV

Diferentemente da migração baseada em C/R, o custo de migração de processos em MV não depende diretamente do tamanho da imagem em memória do processo, mas sim do tamanho total de memória da MV. Além disso, na migração de MV do tipo *live migration* a execução da aplicação não é interrompida durante a migração. A MV na origem continua executando a aplicação enquanto a memória é transferida para o destino. Assim, pode-se dizer que há uma sobreposição do tempo de migração com o tempo de execução da aplicação. Neste contexto, o custo de migrar um processo que executa em MV, do nó n_i para o nó n_j, utilizando um mecanismo de migração de MV do tipo *live migration*, pode ser dado de modo geral por

$$C_{ij} = Trans(M) - \left(\frac{\phi W}{\nu'_i} + \sum_{k \in N'} \frac{\varphi B_k}{V'_{ik}} \right) \quad (5.8)$$

onde $Trans(M)$ representa o custo de transferência de uma MV com tamanho total de memória M; $\frac{\phi W}{\nu'_i}$ define o tempo de processamento realizado pela aplicação durante a migração, onde ϕ representa a fração de trabalho realizada nesse período e ν'_i a capacidade de processamento disponível para a aplicação; já $\sum_{k \in N'} \frac{\varphi B_k}{V'_{ij}}$ define o custo de comunicação realizada durante a migração, onde φ representa a fração de dados comunicados com o processo k e V'_{ik} a vazão disponível para a aplicação.

No caso do suporte a migração de Xen, a transferência da MV é realizada em turnos e, a cada turno, a porção de memória modificada no turno anterior precisa ser retransmitida. Neste contexto, Xen introduz o conceito de WWS, que é a porção de memória frequentemente modificada. Conforme apresentado na Seção 4.2, o tamanho do WWS tem impacto direto na duração da migração e precisa ser levado em consideração no cálculo do custo de transmissão da MV ($Trans(M, WWS)$).

Além disso, levando em consideração o projeto inicial de Xen, é possível utilizar limites de vazão para a transmissão de forma a não interferir demasiadamente no desempenho das aplicações. A cada turno uma vazão limite é calculada dinamicamente. Assim, o custo de transferência de uma MV é dado pelo somatório do custo de cada turno, conforme

$$Trans(M, WWS) = \sum_{0 < r \leqslant R} Trans_r(M, WWS) \quad (5.9)$$

sendo M o tamanho de memória da MV e WWS o tamanho da porção de memória frequentemente modificada. No primeiro turno, toda a memória da MV é transferida. Nos

demais turnos, somente a porção de memória modificada no último turno é transferida. Utilizando estas informações sobre o comportamento dos turnos, pode-se calcular o custo transmissão de cada turno como

$$Trans_r(M, WWS) = \begin{cases} \frac{M_{total}}{V_r}, & r = 1 \\ \frac{WWS}{V_r}, & 1 < r \leqslant R \end{cases} \tag{5.10}$$

onde a vazão V_r pode ser calculada dinamicamente de acordo com a Equação 4.2, para o projeto inicial de Xen, ou ser fixada no valor máximo de vazão ($V_r = V_{max}$), para a versão atual de Xen. Substituindo a Equação 5.9 na Equação 5.8, obtém-se

$$C_{ij} = \sum_{0<r\leqslant R} Trans_r(M, WWS) - \left(\frac{\phi W}{\nu'_i} + \sum_{k \in N'} \frac{\varphi B_k}{V'_{ik}} \right) \tag{5.11}$$

que representa o custo de migração de MVs para um mecanismo de tipo *live migration* como o de Xen.

Substituindo o custo de migração diretamente na Equação 5.2, obtém-se o tempo total de execução com uma migração baseada em MV como

$$\begin{aligned} T_{total} \quad = \quad & \frac{\gamma W}{\nu'_i} + \frac{(1-\gamma)W}{\nu'_j} + \sum_{k \in N} \left(\frac{\alpha_k B_k}{V'_{ik}} + \frac{(1-\alpha_k)B_k}{V'_{jk}} \right) + \\ & Trans(M, WWS) - \left(\frac{\phi W}{\nu''_i} + \sum_{k \in N'} \frac{\varphi B_k}{V''_{ik}} \right) \end{aligned} \tag{5.12}$$

onde ν' e V' representam a capacidade de processamento e comunicação no ambiente virtualizado, enquanto ν'' e V'' a capacidade disponível para a aplicação durante a migração. Da mesma forma, a substituição do custo de migração na Equação 5.3 resulta em

$$T_{total} = \frac{W}{\nu'} + \frac{\sum_{k \in N} B_k}{V'} + Trans(M, WWS) - \left(\frac{\phi W}{\nu''} + \frac{\sum_{k \in N'} \varphi B_k}{V''} \right) \tag{5.13}$$

que representa o tempo total de execução com uma migração baseada em MV sobre um ambiente de execução formado por recursos homogêneos.

5.4 Conclusão

Este capítulo apresentou uma proposta de modelagem para o custo de migração de processo em aplicações MPI. O objetivo principal destes modelos é permitir dimensionar o impacto de uma migração de processo em uma dada aplicação.

Inicialmente, foi definido um modelo para descrever a execução de aplicações MPI que realizam migração de processos. Este modelo leva em consideração tanto ambientes compostos por recursos homogêneos (Equação 5.3) quanto heterogêneo (Equação 5.2). Na sequência, foram definidos modelos de custo de uma migração para as duas abordagens estudadas nesse trabalho.

A Equação 5.4 representa o modelo de custo para um mecanismo genérico de migração baseada em C/R. Já a Equação 5.8 representa o modelo de custo de migração

utilizando um mecanismo genérico de migração de MVs. Estes modelos foram derivados, com base no estudo das soluções de migração específicas, para gerar os modelos definidos pela Equação 5.5 e Equação 5.11, que representam, respectivamente, o custo de uma migração baseada em C/R de Open MPI e o custo de uma migração baseada em MV de Xen.

O Capítulo 7 realiza experimentos de migração para estimar os parâmetros dos modelos propostos. Através da estimativa de parâmetros, deseja-se obter um modelo preenchido que poderá ser utilizado para calcular, de antemão, o custo de migrar um processo em uma dada aplicação. Além disso, é realizada uma avaliação dos modelos na execução de aplicações MPI reais.

6 METODOLOGIA DE AVALIAÇÃO

Este capítulo descreve a metodologia utilizada na realização dos experimentos que estimam os parâmetros e verificam a corretude da modelagem de custo de migração apresentada no Capítulo 5. Os experimentos também avaliam o desempenho das solução de migração estudadas. A metodologia apresentada envolve desde a descrição do ambiente de execução até os métodos estatísticos utilizados na análise dos resultados.

O texto do capítulo está organizado da seguinte forma. A Seção 6.1 descreve a infraestrutura de *hardware* e o *software* básico utilizados nos testes. Na sequência, a Seção 6.2 apresentada os *benchmarks* aplicados na avaliação, classificados de acordo com os objetivos de teste. A Seção 6.3 apresenta o mecanismo desenvolvido para padronizar e automatizar as solicitações de migração nas aplicações de teste. A Seção 6.4 descreve como os tempos de execução foram coletados para permitir a medição de cada fase das migrações. A Seção 6.5 discute os métodos estatísticos utilizados para obter resultados significativos. Por fim, a Seção 6.6 faz um resumo dos principais pontos apresentados no capítulo.

6.1 Ambiente de Execução

Para a realização dos testes, foi utilizado um agregado homogêneo composto por máquinas biprocessadas Intel Pentium III a 1.1 GHz, cada qual com 1 Gbyte de memória principal e 256 Kbytes de memória *cache*. Os nós são interligados por uma rede Fast Ethernet. Todas as máquinas possuem o sistema operacional GNU/Linux na distribuição Debian Sarge 3.1 com duas opções de *kernel*: versão 2.6.18-4 e versão 2.6.18-4-xen (este último com suporte ao Xen 3.0.3).

A distribuição MPI utilizadas em todos os testes é a Open MPI. Até o início da realização dos testes, o suporte a C/R ainda não estava disponível na versão estável da biblioteca. Por esse motivo, optou-se por utilizar Open MPI em duas versões: 1.2.2 (última versão estável), foi utilizada em testes de execução sem migração e em testes de migração de MV; e 1.3 (compilada a partir da revisão 16117 do repositório SVN para permitir a utilização do suporte a C/R), foi utilizada nos testes de migração baseada em C/R. A versão 1.3 de Open MPI foi modificada para otimizar a fase de *checkpoint* e viabilizar a migração de processos conforme descrito na Seção 3.4.

6.2 *Benchmarks* e Descrição dos Testes

Os testes realizados nesse trabalho visaram avaliar o sobrecusto das soluções de migração de processos estudadas, estimar os parâmetros dos modelos propostos no Capí-

Tabela 6.1 Relação dos *Benchmarks* utilizados.

Categoria	Nome	Objetivo	Métricas utilizadas
Micro-benchmark	Linpack	Computação	Tempo de execução de cada fase do programa (s) e desempenho (Mflops).
	NetPIPE	Comunicação	Vazão (Mbps) e latência (μs) de mensagens.
	MigTest	Migração de Processos	Tempo de execução de cada fase da migração (s).
Macro-benchmark	HPL	Aplicação	Tempo total de execução (s) e desempenho (Mflops).
	NPB		

tulo 5 e avaliar o impacto de uma migração de processo no tempo total de execução de aplicações MPI. Para tanto, utilizou-se programas de *benchmark* amplamente aceitos. A Tabela 6.1 apresenta estes *benchmarks* e classifica-os de acordo com o objetivo da medição: medir desempenho de computação, comunicação, migração de processo ou execução completa de aplicação.

A avaliação do sobrecusto das soluções de migração foi realizada com auxílio de *micro-benchmarks*, que são aplicações (possivelmente sintéticas) utilizadas para avaliar isoladamente alguma característica específica como, por exemplo, desempenho de computação e comunicação. No caso de computação, utilizou-se uma aplicação sequencial do pacote Linpack (DONGARRA; STEWART, 1984), que realiza uma decomposição LU em duas fases, fatoração e *back-solve*, e reporta o tempo gasto em cada uma delas. Já para avaliar o desempenho de comunicações, utilizou-se NetPIPE (SNELL; MIKLER; GUSTAFSON, 1996), uma ferramenta que realiza testes de *ping-pong* entre dois processos MPI e mede o desempenho das comunicações. O cálculo da vazão é realizado variando o tamanho da mensagem de 0 até 8 MBytes. A latência é calculada dividindo o tempo de ida e volta de mensagens pequenas (menores que 64 bytes) pela metade.

A avaliação do custo de migração e estimativa dos parâmetros dos modelos propostos foram realizadas através da execução de uma aplicação sintética. Na ausência de um *benchmark* conhecido específico para esse fim, optou-se por desenvolver uma aplicação que pudesse variar a quantidade de memória alocada e reportar os tempos de execução e migração. A aplicação desenvolvida (referenciada na Tabela 6.1 e no resto do trabalho como MigTest) executa de forma iterativa, sendo que a cada iteração uma porção de memória é modificada. A variação na quantidade de memória alocada e na porção de memória modificada em cada iteração permite testar a migração com diferentes tamanhos de processos (S) e áreas de memórias constantemente modificadas (WWS) (este último caso, é particularmente importante em migração baseada em MV, conforme descrito no Capítulo 4). Além disso, uma mensagem (*token*) é passada entre os processos a cada iteração, o que permite verificar se a conectividade mantém-se em caso de migração.

A avaliação do impacto de uma migração de processos na execução de aplicações MPI foi realizada com o auxílio de *macro-benchmarks*, que são aplicações que resolvem problemas reais utilizadas para avaliar a execução paralela como um todo. O primeiro deles é o *benchmark* HPL (*High Performance Linpack*) (PETITET et al., 2007). HPL é uma apli-

cação que resolve sistemas lineares densos em dupla precisão e é comumente utilizado para avaliação de desempenho de computadores paralelos como agregados[1]. A outra aplicação utilizada foi o *benchmark* NPB (*NAS Parallel Benchmark*) (BAILEY et al., 1991). NPB é um conjunto de 7 sub-programas que realizam operações críticas em ambientes paralelos. Dentre eles optou-se pelo SP (*Pentadiagonal solver*) por apresentar um longo tempo de execução.

6.3 Automatização e Padronização da Migração de Processos

Os experimentos que envolvem migração de processos nas diferentes soluções de migração requerem uma forma padronizada de iniciar o procedimento migração. Por exemplo, a migração nos testes com a aplicação HPL é iniciada, de forma automatizada, logo após a primeira iteração do laço principal do programa. Para permitir isto, optou-se por criar uma chamada `MPI_Migrateto()`, inspirada na função de mesmo nome de Adaptive MPI (HUANG; LAWLOR; KALÉ, 2003). Esta função não é bloqueante e pode ser utilizada dentro da aplicação, em algum ponto específico, para requisitar a migração do processo para um nó destino.

A função `MPI_Migrateto()` se comunica com um *daemon*, chamado `migd`, que é o responsável por realizar a migração propriamente dita. A interface de comunicação entre `MPI_Migrateto()` e `migd` é única, independentemente da solução de migração utilizada. Já o *daemon* `migd` é modular e pode suportar diferentes soluções de migração. Desta forma, é possível ter uma mesma aplicação MPI, capaz de utilizar diferentes soluções de migração.

No caso de Open MPI, o *daemon* `migd` precisa executar no mesmo nó que a aplicação foi lançada (nó que executa o `mpirun`). Os passos da migração são os seguintes.

1. Aplicação utiliza chamada `MPI_Migrateto()` para solicitar a migração do processo para um nó destino;
2. `migd` recebe a solicitação e envia uma requisição de *checkpoint* para Open MPI;
3. Quando o *checkpoint* é concluído, `migd` transfere o arquivo de contexto do nó de origem para o nó destino;
4. `migd` reinicia a execução da aplicação utilizando um arquivo de máquinas que contempla o novo mapeamento;
5. Aplicação continua a execução normalmente com o novo mapeamento.

No caso de Xen, o *daemon* `migd` precisar executar no Domain0 de algum nó do agregado. os passos da migração são os seguintes.

1. Aplicação utiliza a chamada `MPI_Migrateto()` para solicitar a migração do processo para um nó destino;
2. `migd` recebe a solicitação e envia um comando de migração para MV que hospeda o processo para o nó destino;
3. Xen realiza a transferência da MV sem interromper a execução da aplicação;
4. Aplicação continua a execução normalmente com o novo mapeamento.

[1]HPL também é usado no *site* que avalia as 500 máquinas mais poderosas do mundo: www.top500.org

6.4 Monitoração da Migração e Coleta de Tempos de Execução

Os experimentos para estimar os parâmetros dos modelos de migração propostos requerem a coleta de tempo de cada fase da migração. Com isso, foi necessário instrumentar a aplicação MigTest e os mecanismos de migração para coletar internamente o tempo das fases. Esta instrumentação foi realizada com o auxílio da biblioteca libRastro (SILVA; OLIVEIRA STEIN, 2002). libRastro permite monitorar aplicações paralelas, de forma pouco intrusiva, através do registro de eventos observados durante a sua execução. Exemplos de eventos registrados podem ser o envio ou recepção de mensagens, chamadas de funções, mudanças de estados, etc. Em uma plataforma do tipo agregado, os eventos são registrados em cada nó de acordo com o relógio local para, após o término da execução, serem combinados em um único arquivo de rastros e sincronizados com um relógio de referência. No caso da migração de processo, isto é particularmente importante, porque a localização física do processo é alterada e, com isso, o relógio do sistema.

A monitoração da migração de processos baseada em C/R consistiu na instrumentação dos seguintes itens:

- Início da execução da aplicação;
- Chamada `MPI_Migrateto()`, dentro da aplicação;
- Coordenação do *checkpoint* global, dentro do módulo de C/R da Open MPI;
- Realização do *checkpoint* local, dentro de BLCR;
- Transferência do arquivo de contexto para o nó destino da migração, dentro do *daemon* `migd`.
- Retomada da execução após a migração (*restart*), dentro de Open MPI;
- Realização do *restart* do processo local, dentro de BLCR; e
- Fim da execução da aplicação.

Um ponto crítico na coleta dos tempos das fases da migração baseada em C/R foi a dificuldade em registrar o momento exato que o processo MPI retoma a execução (*restart*). Este registro é importante porque indica o fim da migração. Quando Open MPI requisita a retomada da execução de cada processo local para BLCR, este cria um processo filho e não retorna ao processo pai antes do término de toda a execução restante. A instrumentação realizada registra o último nível de processamento do arquivo de contexto antes de BLCR passar o controle da execução para o processo filho. Como não é possível garantir que esse registro representa o momento exato do final da migração, também foi registrado o primeiro evento (envio ou recebimento de mensagem) dentro da aplicação após a migração. A cada iteração de MigTest novos eventos são gerados, dessa forma, a observação do tempo entre o registro do fim da migração e o primeiro evento gerado pela aplicação revelou uma diferença média de apenas 100 ms. Como os testes com MigTest duram algumas dezenas de segundos, conforme os resultados do Capítulo 7, esta diferença de tempo pode ser considerada aceitável.

No caso da migração de processos baseada em MV, as informações sobre o andamento da migração são disponibilizadas no registo (*log*) do sistema com informações de tempo relativas ao Domain0. Neste caso, a instrumentação envolveu os seguintes itens:

- Início da execução da aplicação;
- Chamada `MPI_Migrateto()`, dentro da aplicação;
- Andamento da migração de Xen, direto do *log* do sistema; e
- Fim da execução da aplicação.

A geração do rastro de execução com um único relógio de referência foi realizada através da inclusão de todos os nós da aplicação no procedimento de sincronização. Como Xen disponibiliza informações de tempo em relação ao Domain0 e os registros da aplicação são realizados dentro da MV (DomainU), a sincronização envolveu a inclusão do Domain0 e DomainU de cada nó. No caso da migração com Open MPI, os eventos registrados possuem informações de tempo de cada nó físico e, após a migração, este nó será diferente. A versão original de libRastro identifica a localização física do processo uma única vez no início da execução e, desta forma, não percebe mudanças durante a execução. Para contornar esse problema, libRastro foi modificada para permitir, em caso de migração, a identificação do nó a cada novo registro. Com isto, foi possível a geração de rastros de execução com tempos de diferentes nós sincronizados com um único relógio de referencia, mesmo em caso de migração, tanto para migração baseada em C/R quanto para MV.

Uma vez que os rastros de execução foram gerados, utilizou-se a ferramenta de visualização Pajé (CHASSIN DE KERGOMMEAUX; OLIVEIRA STEIN, 2001) para verificar a corretude da instrumentação e da sincronização dos tempos. Pajé é uma ferramenta de visualização que permite observar o comportamento de aplicações em ambientes de execução paralelos e distribuídos. Para isso, Pajé utiliza rastros gerados por outras ferramentas durante a execução das aplicações, como libRastro. Após está verificação, foi possível extrair os tempos de execução de cada fase das migrações – utilizados para estimar os parâmetros dos modelos – diretamente dos rastros de execução sincronizados.

6.5 Testes Estatísticos e Regressão Linear

Os resultados apresentados nesse trabalho representam a média aritmética de execuções consecutivas. Para testes com *micro-benchmark*, que apresentam um tempo de execução menor, foram realizadas 30 execuções para cada caso. Já para os testes com *macro-benchmarks* foram realizadas 10 execuções para cada caso. Em cada conjunto de execuções, foi calculado o desvio padrão, que permite avaliar a variabilidade dos resultados obtidos e a representatividade da média apresentada como resultado. Em alguns casos, foram utilizados gráficos para apresentar as resultados. Nestes gráficos, utilizou-se, sempre que possível, o recurso de barras de erros com desvio padrão para visualizar a distribuição dos dados em torno da média.

Nos testes de comparação, foram realizados testes T (*t-test*) com intervalo de confiança de 95 % ($\alpha \geq 0,95$) para comparar as médias de dois grupos de experimentos (por exemplo, para a execução de uma aplicação com e sem uma funcionalidade habilitada). O *t-test* permite verificar se a diferença entre as médias observadas é estatisticamente significativa.

Para estimar os parâmetros dos modelos propostos no Capítulo 5, utilizou-se regressão linear. Regressão linear é o método estatístico que tenta modelar o relacionamento entre uma variável dependente (variável de resposta) e uma ou mais variáveis independentes específicas (variáveis explicativas), através do preenchimento de uma equação linear

utilizando os dados observados em experimentos. O resultado de uma regressão linear simples é um modelo linear definido pela equação da reta com a forma $Y = a + bX$, onde X é a variável explicativa e Y a variável de resposta. A intersecção da reta com o eixo vertical é dado por a (valor de Y quando X = 0) e a inclinação da reta por b.

A qualidade de um modelo linear pode ser verificada através da comparação dos dados observados, dispostos em gráfico de dispersão, com a linha reta (para uma variável), ou o plano (para duas variáveis), obtidos através da equação linear do modelo. Um bom modelo tende a minimizar a distância, medida verticalmente, entre os pontos observados e a representação gráfica do modelo. Uma medição numérica da qualidade de um modelo é o coeficiente de determinação (R^2), o qual é um valor entre 0 e 1 que indica a proporção da variação total da variável resposta Y que é explicada pela equação do modelo. Por exemplo, a obtenção de R^2 = 0,75 indica que 75 % de variância é explicada pelo modelo. Neste caso, quanto maior for o valor de R^2, melhor é o modelo.

Neste trabalho, utilizou-se o programa de estatística R (R Project, 2008) para preencher a equação dos modelos propostos, utilizando o Método dos Mínimos Quadrados (MMQ). A ideia básica de MMQ é tentar descobrir quais são os valores dos coeficientes da equação, de tal modo que a soma dos quadrados das distâncias (tomadas na vertical) da referida reta $Y = a + bX$, a cada um dos pontos dados (Y), seja a menor possível.

6.6 Conclusão

Este capítulo apresentou a metodologia utilizada nos experimentos realizados. Os objetivos dos experimentos foram avaliar o desempenho das soluções de migração estudadas, estimar os parâmetros e verificar a corretude da modelagem de custo de migração apresentada no Capítulo 5. Para alcançar estes objetivos, foram selecionados alguns *benchmarks* amplamente aceitos, divididos em duas categorias. Estes *benchmarks* permitiram avaliar, de forma isolada, tanto o sobrecusto das soluções utilizadas quanto o custo da migração propriamente dita. Em todos os testes, utilizou-se o agregado de computadores disponível no grupo GPPD.

A preparação dos testes envolveu a criação de um mecanismo para padronizar as chamadas às soluções de migração. Desta forma, foi possível ter uma mesma aplicação MPI capaz de utilizar diferentes soluções de migração. Além disso, foi necessário instrumentar as soluções para permitir a medição correta do custo de cada fase das migrações. Por fim, métodos estatísticos foram utilizados para obter resultados mais significantes e para preencher e avaliar o modelo de custo de migração proposto.

7 ESTIMATIVA DE PARÂMETROS E AVALIAÇÃO DO MODELO

O objetivo principal deste capítulo é estimar os parâmetros e avaliar os modelos propostos. Através da estimativa de parâmetros, deseja-se obter um modelo preenchido que poderá ser utilizado para calcular, de antemão, o custo de migrar um processo em uma dada aplicação. A qualidade das previsões dos modelos é importante, principalmente quando utilizadas na tomada de decisão em algoritmos de escalonamento. Neste caso, uma previsão errada pode prejudicar o desempenho final do escalonamento.

Visto que ambas as soluções de migração estudadas adicionam funcionalidades ao MPI ou ao sistema operacional, este capítulo também avalia o sobrecusto na execução de programas MPI com essas soluções. No caso da migração baseada em MVs, esta avaliação é particularmente importante para verificar o sobrecusto imposto pela camada de abstração oferecida para prover a virtualização. No caso das migrações baseadas em C/R, avaliam-se também as modificações realizadas para otimizar o suporte a C/R de Open MPI.

Este capítulo ainda traz uma avaliação do impacto de uma migração na execução de aplicações reais. Esta avaliação é importante para verificar a viabilidade da utilização de migração de processos em aplicações com diferentes comportamentos.

O texto deste capítulo esta organizado da seguinte forma. Primeiramente, a Seção 7.1 apresenta uma avaliação do sobrecusto das soluções de migração. A Seção 7.2 apresenta os experimentos para estimar os parâmetros dos modelos. Na sequência, a Seção 7.3 apresenta os experimentos de migração de processos em aplicações reais. Os resultados desta seção são utilizados para avaliar o impacto de uma migração na execução das aplicações e a qualidade das previsões dos modelos propostos. Por fim, a Seção 7.4 contém as considerações finais e conclusões obtidas no decorrer capítulo.

7.1 Avaliação do sobrecusto

Esta seção apresenta uma avaliação do sobrecusto das soluções de migração nos quesitos de computação e comunicação. No caso de migração de MV, observou-se também o impacto da virtualização de recursos na execução de programas MPI. Nos testes de comunicação, foram utilizados diferentes mapeamentos de processos (e diferentes esquemas de máquinas virtuais, no caso de Xen) para avaliar largura de banda e latência.

7.1.1 Sobrecusto do Suporte a C/R de Open MPI

A migração de processo em Open MPI é realizada com o auxílio do módulo de CR. Tradicionalmente, as soluções de C/R são usadas para tolerar falhas e retomar a execu-

ção a partir de um estado consistente. Este tipo de uso requer a realização de operações de *checkpoint* em intervalos de tempo definidos, o que pode impactar no desempenho da aplicação. No caso da migração de processos descrita nesse trabalho, a operação de *checkpoint* só é realizada quando uma migração é requisitada. Neste contexto, não é objetivo deste trabalho avaliar o sobrecusto de múltiplas operações de *checkpoint* na execução de programas MPI. Além disso, existem trabalhos na literatura que avaliam o uso de C/R para esse fim (DUPROS; CARISSIMI; MEHAUT, 2006).

7.1.1.1 Sobrecusto de Comunicação

O módulo de C/R de Open MPI atua diretamente sobre as operações de comunicação, conforme discutido no Capítulo 3. Desta forma, é pertinente avaliar o sobrecusto imposto por esse módulo sobre latência e vazão, já que estas métricas têm um impacto direto no desempenho total das aplicações. Para isso, executou-se NetPIPE sobre Open MPI com o suporte a C/R habilitado e não habilitado. No primeiro teste realizado, utilizaram-se 2 nós do agregado, sendo mapeado um processo para cada nó, e mediu-se o desempenho de comunicações MPI. Através dos resultados obtidos com esse teste, foi possível observar um aumento médio na latência de 4 % quando o suporte a C/R é habilitado. Este aumento teve impacto no tempo de transmissão de mensagens pequenas (menor que 64 Bytes), já para mensagens maiores não foi possível observar diferença estatisticamente significativa. Este sobrecusto é atribuído às chamadas de funções realizadas a cada mensagem (HURSEY et al., 2007). No caso da vazão, não foi possível detectar sobrecusto significativo.

O mesmo teste foi realizado colocando 2 processos em um mesmo nó do agregado. O objetivo deste teste foi verificar o sobrecusto do suporte a C/R na comunicação entre processos locais. Este cenário é comum em aplicações reais, principalmente em ambientes formados por nós multiprocessados. Os resultados obtidos revelaram um aumento na latência de 2 μs para 30 μs quando o suporte a C/R é habilitado. O gráfico da Figura 7.1 apresenta a vazão de rede obtida nos resultados. Através do gráfico, é possível observar uma degradação no desempenho da vazão quando C/R está habilitado. Por exemplo, quando são transmitidos pacotes de 1 MByte, o módulo de C/R degrada a vazão da comunicação em 15 %.

7.1.1.2 Avaliação da Modificação para Permitir Migração

O suporte a C/R original de Open MPI realiza a transferência dos arquivos de contexto para um servidor de armazenamento persistente. Como mencionado na Seção 3.4, foi criado uma novo componente para SnapC com o objetivo de reduzir o custo das operações de C/R e viabilizar a migração de processos. Nessa nova implementação, apenas metadados são trafegados pela rede. Neste contexto, foi realizada uma comparação de desempenho entre o suporte a C/R original de Open MPI e a modificação realizada para permitir migração. O gráfico da Figura 7.2 apresenta os resultados da fase de *checkpoint* de Open MPI no programa MigTest, variando o tamanho de memória do processo de 1 à 500 MBytes. Através deste gráfico, é possível observar que o custo de *checkpoint* diminuiu significativamente com a modificação que elimina a transferência de arquivos de contexto.

Outro teste realizado foi a execução da aplicação HPL (*High Performance Linpack*) em 3 nós do agregado e a migração de um dos processos para um quarto nó. O objetivo deste teste é avaliar o ganho de desempenho obtido com a modificação do suporte a C/R de Open MPI em uma migração de processo completa. Para isto, utilizou-se a aplicação

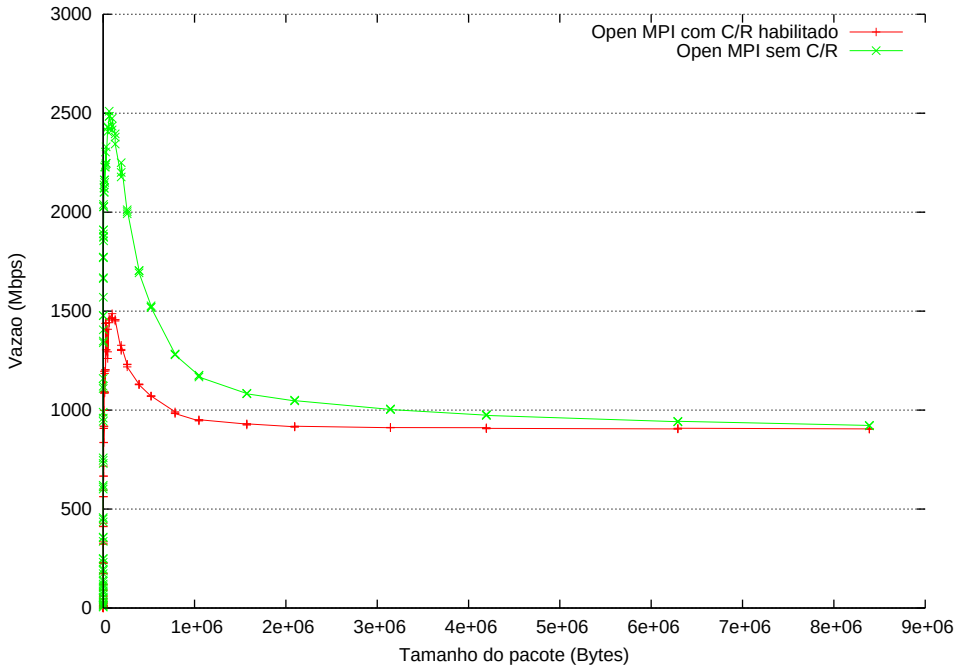


Figura 7.1 Largura de banda do NetPIPE sobre Open MPI usando um nó do agregado.

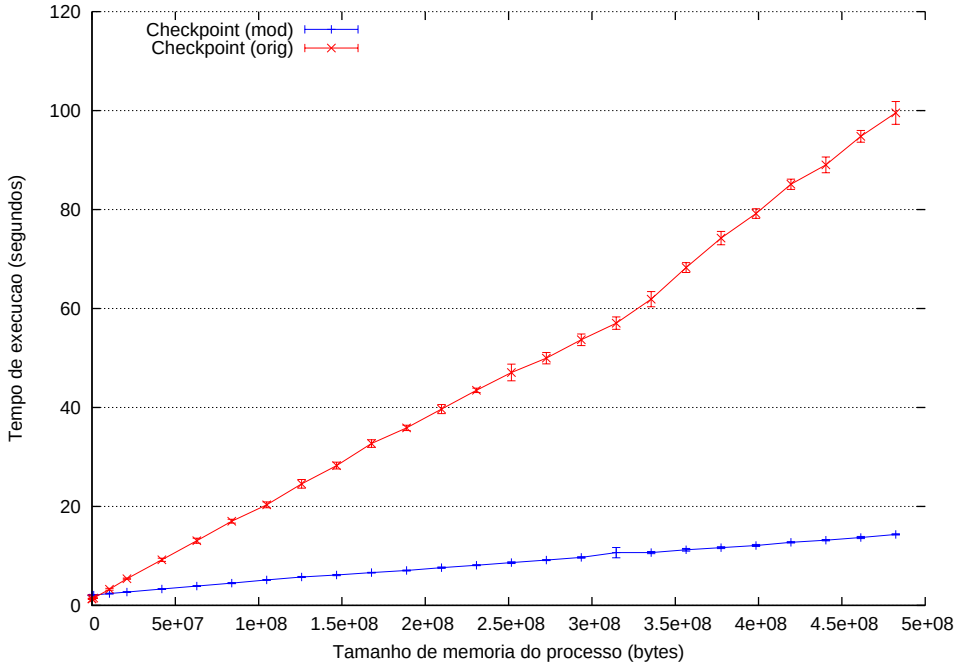


Figura 7.2 Desempenho da fase de *checkpoint* de Open MPI.

HPL com um tamanho de problema de N=8000 e a migração de processo foi inicia com uma chamada MPI_Migrateto() após a segunda iteração do laço principal da aplicação. O gráfico da Figura 7.3 apresenta uma comparação entre os tempos médios para uma execução sem migração, com 1 migração utilizando o suporte a C/R original (OMPI+BLCR) e com 1 migração utilizando o suporte a C/R modificado (OMPI+BLCR mod). Através desse gráfico é possível observar uma diminuição de aproximadamente 7 vezes no custo total de migração quando utiliza-se o suporte a C/R modificado, passando de 409,2 para 59,3 segundos em média.

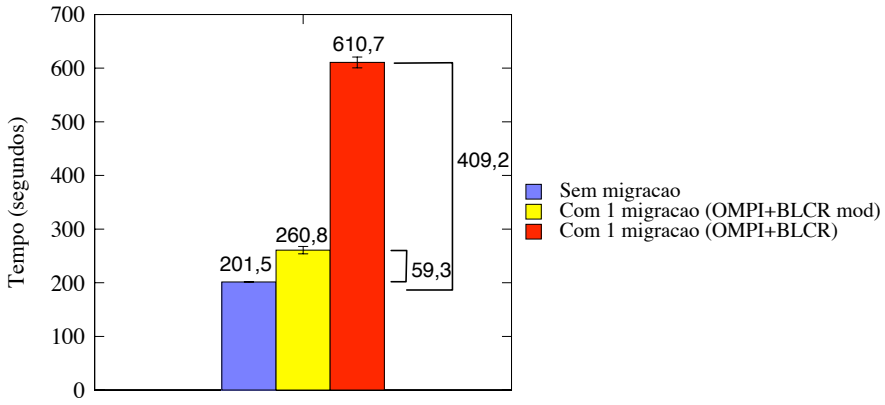


Figura 7.3 Desempenho de uma migração de processo baseada em C/R na aplicação HPL.

7.1.2 Sobrecusto da Virtualização de Xen

A virtualização de recursos computacionais historicamente impõe um sobrecusto de desempenho devido às abstrações de *software* realizadas. Neste contexto, é natural avaliar o sobrecusto apresentado pela virtualização de Xen, antes da migração propriamente dita. A seguir são apresentados os resultados obtidos. Primeiramente, observa-se o impacto da virtualização de Xen nas partes de computação e comunicação. Na sequência, apresenta-se uma análise do sobrecusto na execução de uma aplicação MPI sobre máquinas virtuais.

7.1.2.1 Desempenho de Computação

Como mencionado no Capítulo 6, utilizou-se o *benchmark* Linpack para avaliar o sobrecusto imposto por Xen na execução de um programa de computação intensiva. O gráfico da Figura 7.4 contém os tempos de execução de Linpack com uma matriz de entrada de tamanho 3000x3000 com valores em dupla precisão. Além dos tempos informados pelo próprio Linpack, também utilizou-se os tempos coletados pelo comando time do sistema operacional.

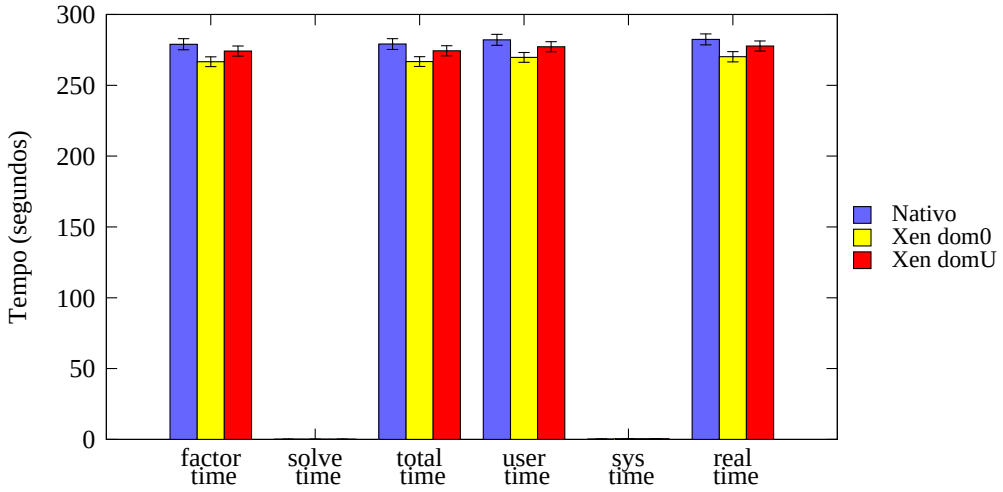


Figura 7.4 Desempenho na execução do *benchmark* Linpack.

Os resultados do gráfico da Figura 7.4 mostram que Xen apresenta um desempenho de computação superior ao do sistema nativo (usando o *kernel* que acompanha a distribuição Debian). Este resultado deve-se ao escalonador de processos do *kernel* virtualizado, que implementa de forma eficiente o escalonador BTV (*Borrowed Virtual Time*) (DUDA; CHERITON, 1999). Continuando a análise no gráfico, pode-se notar que o Domain0 (representado no gráfico por Xen dom0) apresenta um desempenho melhor que o das MVs Xen (Xen domU). Acredita-se que a causa disto seja o fato de Domain0 ter maiores privilégios que os outros domínios de Xen.

Além disso, aplicando o *t-test* e observando o intervalo de confiança percebe-se que a diferença entre os resultados do sistema nativo e de Xen no Domain0 é estatisticamente significativa. O mesmo não pode ser afirmado na comparação entre o sistema nativo e máquinas virtuais Xen. Apesar disso, a avaliação é válida pois mostra que, apesar das modificações para prover a virtualização, não existe uma degradação significativa no desempenho de computação de aplicações executando sobre MVs Xen e, portanto, é possível conseguir desempenho igual ou até mesmo superior ao sistema nativo.

7.1.2.2 Desempenho de Comunicação

A avaliação do desempenho de comunicação de Xen envolveu diferentes esquemas de mapeamentos de processos MPI sobre MVs. O primeiro cenário avaliado utiliza 2 nós do agregado, cada com uma MV. O teste realizado compara a vazão e latência de comunicação, utilizando NetPIPE, para processos MPI executando em dois ambientes: diretamente no sistema nativo e sobre MVs. Os resultados obtidos revelaram um aumento de 85 % na latência média, passando de 70 μs no sistema nativo para 130 μs no sistema virtualizado. Apesar disso, percebe-se que o sistema virtualizado possui um desempenho de vazão bastante próximo ao do sistema nativo. Em termos de percentagem, a vazão no sistema virtualizado é em média de 1 a 2% menor, principalmente para as mensagens com tamanho maior.

O outro cenário avaliado utiliza 2 MVs em um mesmo nó do agregado e mapeia um processo MPI em cada uma delas. Além de ser um cenário comum em ambientes multiprocessados, este teste permite avaliar se é viável utilizar migração de MVs para agrupar, em um mesmo nó físico, processos que trocam um volume grande de dados. O gráfico da Figura 7.5 apresenta os resultados de vazão obtidos nesta comparação. Analisando este gráfico, é possível observar uma degradação no desempenho de comunicação quando colocam-se duas MVs no mesmo nó do agregado. Por exemplo, quando se transferem 6 MBytes de dados, processos MPI no mesmo nó usando o sistema nativo atingem uma largura de banda de aproximadamente 1200 Mbps, enquanto o sistema com duas MVs (1 processo MPI em cada) no mesmo nó consegue 180 Mbps.

Acredita-se que a degradação de desempenho do sistema virtualizado deve-se ao fato de todas as operações de rede das MVs exigirem algum processamento. Assim, ocorre concorrência pelo uso da rede e o desempenho fica degradado. Uma prova dessa colocação foi que, ao desabilitar a verificação de somatório (*checksumming*) para controle de erro nas duas interfaces virtuais, o desempenho melhorou consideravelmente.

Outra linha mostrada no gráfico da Figura 7.5 é aquela onde são colocados 2 processos MPI na mesma MV de um nó. Nesse esquema, a comunicação entre os processos não sai de dentro da MV e o desempenho fica muito próximo ao do sistema nativo como pode ser visto na Figura 7.5.

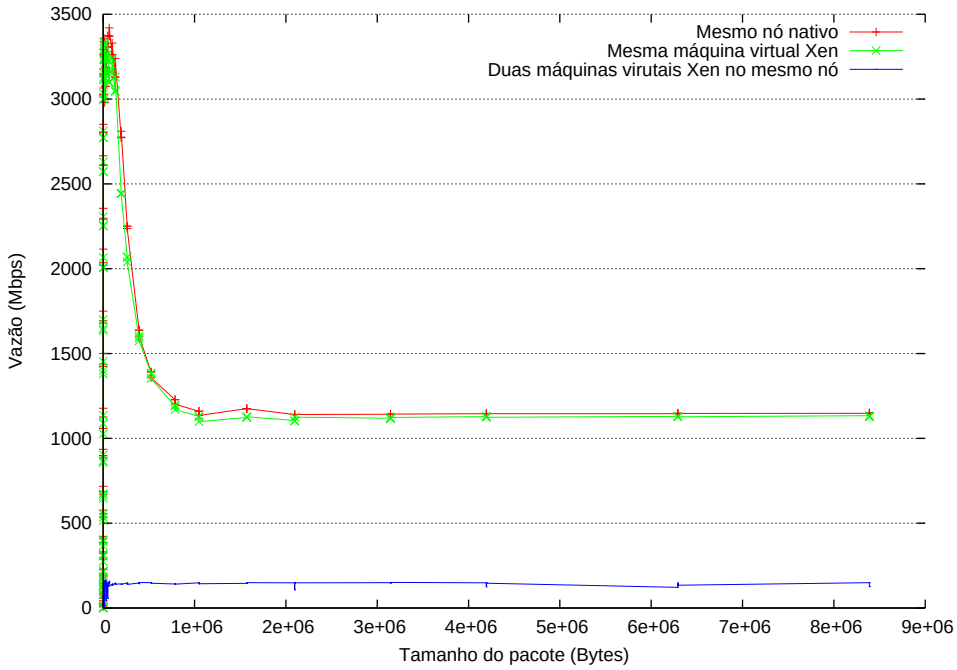


Figura 7.5 Largura de banda do NetPIPE usando um nó do agregado.

7.1.2.3 Desempenho com a Aplicação HPL

Após verificar o sobrecusto de Xen em quesitos específicos, como computação e comunicação, foram realizados testes para avaliar o desempenho na execução completa de uma aplicação MPI. Para estes testes, utilizou-se MVs com tamanho de memória de 256 MBytes e a aplicação HPL com um tamanho de problema de N=8000 (tamanho calculado para utilizar toda a memória disponível nas MVs). O gráfico da Figura 7.6 apresenta os resultados da execução de HPL em 3 nós, cada qual com 2 MVs. Assim, no total tem-se 6 processos MPI executando, um em cada MV. Os resultados mostram um tempo de execução, em média 9% maior para o ambiente virtualizado quando comparado com a execução do sistema nativo. Em termos de desempenho, o sistema nativo atingiu 2,6 GFlops, enquanto o virtualizado ficou em 2,35 Gflops. Acredita-se que isto seja causado pelo problema de desempenho de rede descrito anteriormente.

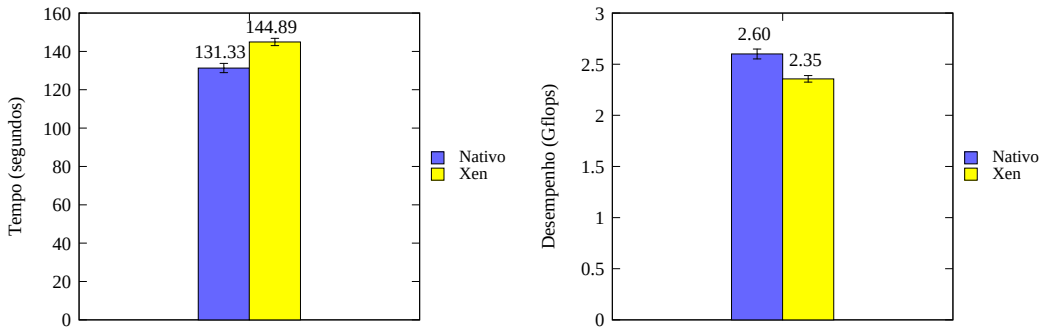


Figura 7.6 Desempenho de Xen na execução da aplicação HPL sobre 6 MVs (2 MVs por nó).

Outro teste realizado foi colocar apenas uma MV por nó do agregado e cada uma

delas executa somente um processo, totalizando uma aplicação com 3 processos MPI. A ideia é comparar o desempenho com um sistema nativo com essa mesma distribuição de processos. Os resultados são apresentados no gráfico da Figura 7.7. Desta vez, o desempenho de Xen ficou bastante próximo ao do ambiente nativo, apresentando um acréscimo de apenas 1,1% no tempo de execução da aplicação. No entanto, é importante salientar que esta configuração não explora todos os recursos disponíveis, já que os nós utilizados são biprocessados.

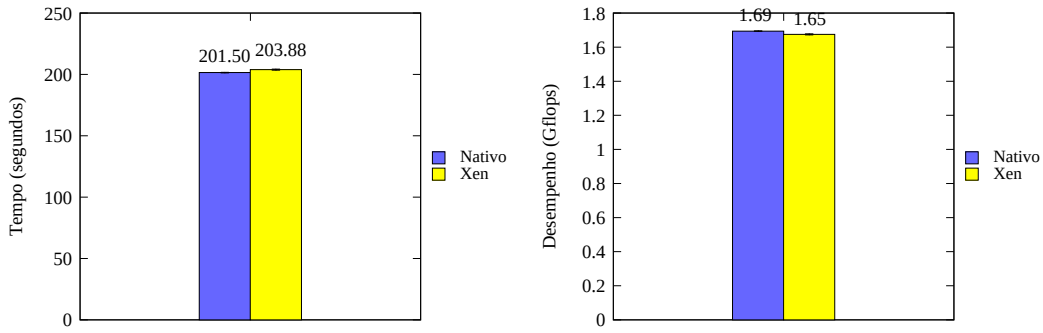


Figura 7.7 Desempenho de Xen na execução da aplicação HPL sobre 3 MVs (1 MV por nó).

7.2 Estimativa de Parâmetros para o Modelo de Custo de Migração

Esta seção apresenta a estimativa dos parâmetros dos modelos propostos. Para isso, utilizou-se o método estatístico de regressão linear. Conforme descrito na Seção 6.5, a regressão linear tenta preencher uma equação linear utilizando os dados observados em experimentos. Neste trabalho, utilizou-se o programa estatístico R, que foi alimentado com dados de entrada medidos pelo MigTest.

O programa MigTest foi executado em dois nós do agregado e migrado para um terceiro nó após quinta iteração (através de uma chamada `MPI_Migrateto()`). No caso da migração baseada em MVs, utilizaram-se os mesmos dois nós, cada um com uma MV, e migrou-se uma das MVs para um terceiro nó físico. A seguir são apresentados os resultados obtidos variando as características dos processos migrados.

7.2.1 Modelo de Custo de Migração Baseada em C/R

O modelo de custo de migração baseada em C/R, descrito pela Equação 5.4, possui duas variáveis, S e V, que representam respectivamente o tamanho de memória do processo e a vazão de rede entre os nós de origem e destino da migração. Devido à limitação do ambiente utilizado (um agregado com rede homogênea de 100Mbps) e a dificuldade de simular diferentes velocidade de rede de forma confiável, resolveu-se fixar o valor da vazão em sua capacidade máxima e variar apenas o valor de S. Nos testes realizados, utilizou-se o programa MigTest com tamanho de memória dos processos variando de 1 à 500MBytes.

O gráfico da Figura 7.8 contém os tempos de cada fase das migrações variando o tamanho do processo. Através do gráfico, é possível observar um comportamento linear para cada uma das fases, sendo a fase de transferência dos dados a que apresenta maior

duração. Os dados apresentados nesse gráfico foram utilizados para preencher o modelo de custo de migração baseada em C/R.

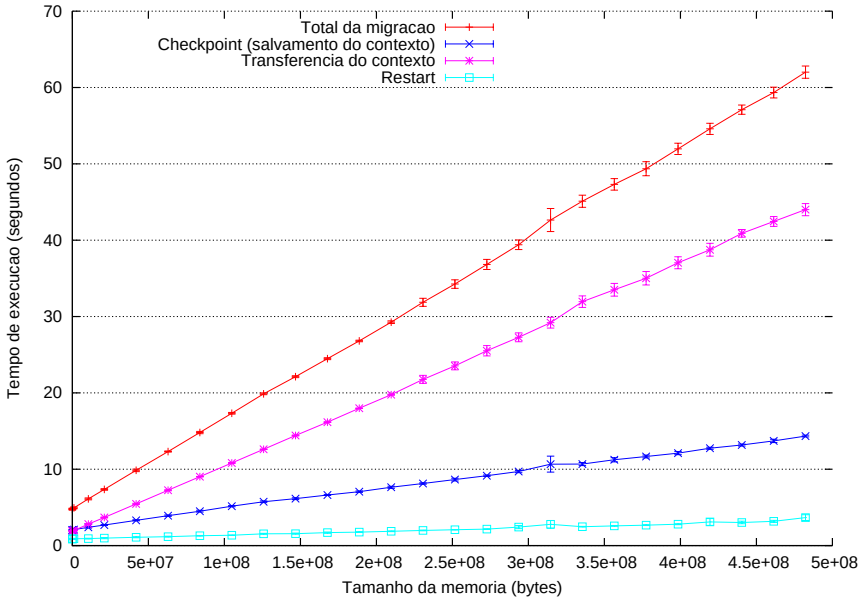


Figura 7.8 Tempos de cada fase de migração de processos baseada em C/R variando o tamanho de memória (S).

A regressão linear do custo da fase de *checkpoint* variando o tamanho de memória do processo S é dada pela equação

$$Check(S) = 2,274 + 2,514 * 10^{-7}(S) \tag{7.1}$$

a qual foi obtida com $R^2 = 0,9942$.

A regressão linear da fase de transferência é dada pela equação

$$Trans(S) = 1,660 + 8,825 * 10^{-8}(S) \tag{7.2}$$

obtida com $R^2 = 0,9983$.

A regressão linear da fase de *restart* é dada pela equação

$$Restart(S) = 0,8503 + 5,099 * 10^{-9}(S) \tag{7.3}$$

obtida com $R^2 = 0,9486$.

Os resultados da regressão linear de cada uma das fases da migração são apresentados nos gráficos da Figura 7.9. Nestes gráficos, é possível observar as retas das equações preenchidas e os resultados experimentais, representados por pontos.

Substituindo $Check(S)$, $Trans(S)$ e $Restart(S)$ na Equação 5.4, obtém-se o custo de migração de um processo de tamanho S do nó de origem i para o destino j, interligados por uma rede de 100Mbps, representado pela equação

$$C_{ij}(S) = 4,785 + 1,185 * 10^{-7}(S) \tag{7.4}$$

a qual apresenta $R^2 = 0,9989$. O gráfico da Figura 7.10 contém a representação da Equação 7.4 em forma de uma reta e os resultados experimentais como pontos. A qualidade do modelo pode ser observada pela proximidade entre a reta e os pontos do experimento.

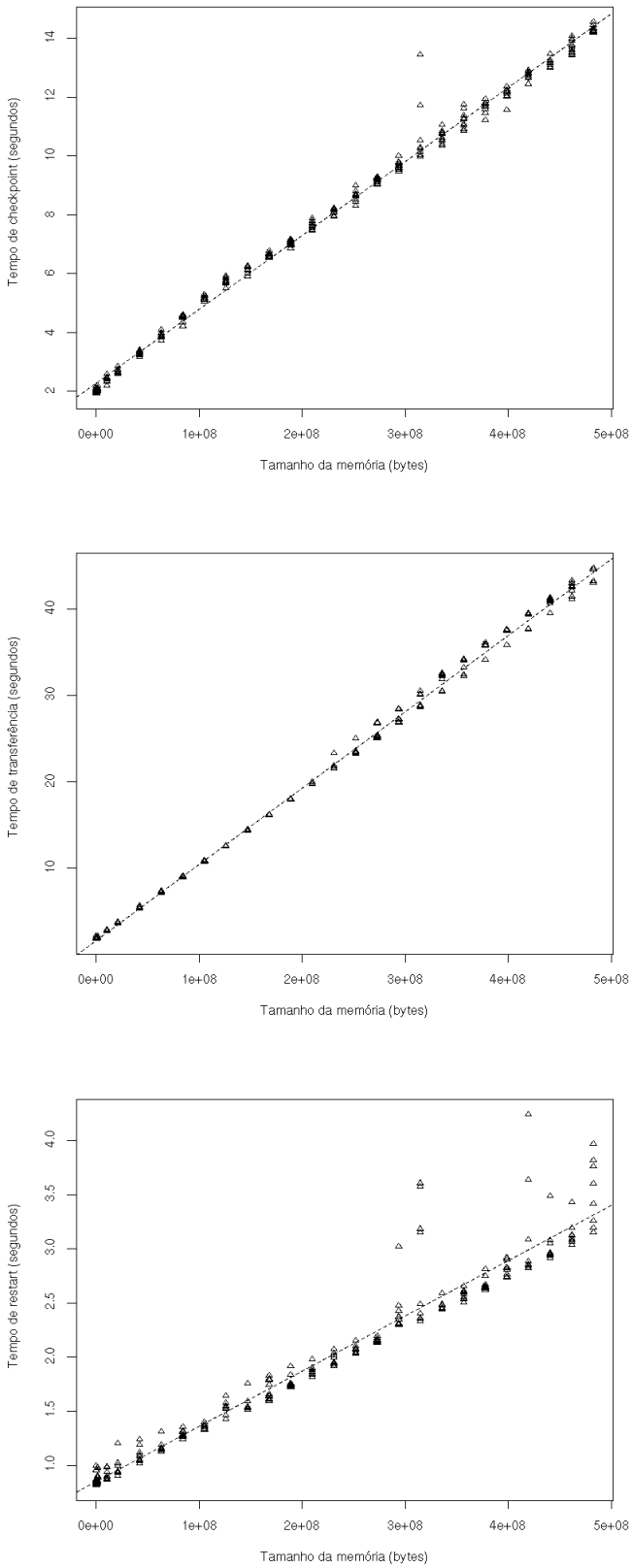


Figura 7.9 Regressão linear de cada fase de migração baseada em C/R variando tamanho de memória (S).

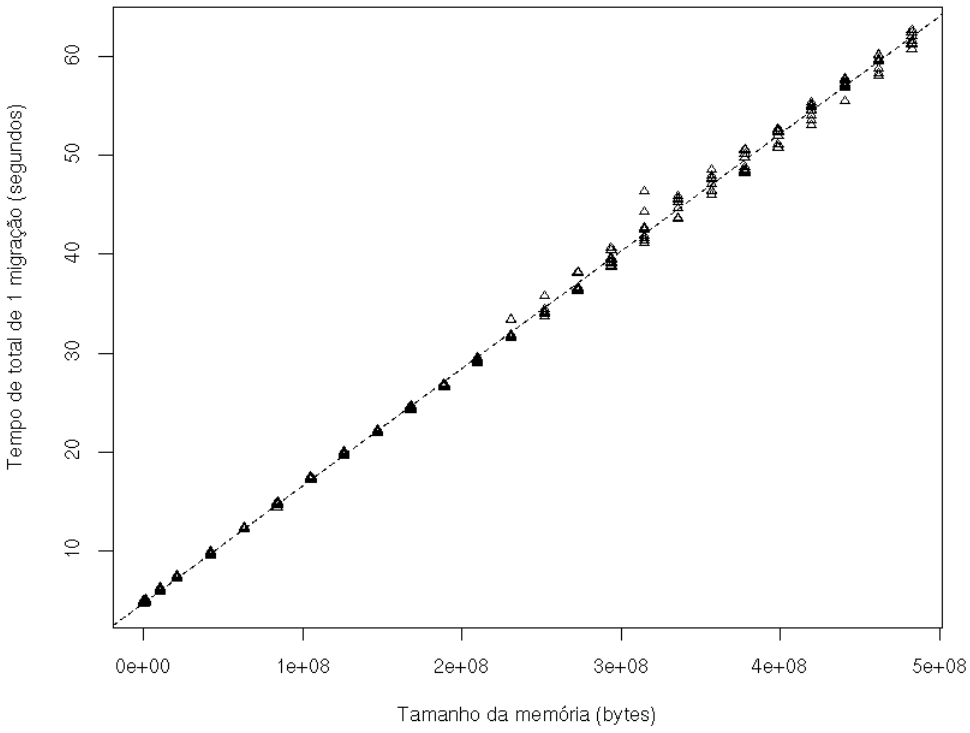


Figura 7.10 Regressão linear de uma migração baseada em C/R variando tamanho da memória (S).

7.2.2 Modelo de Custo de Migração Baseada em MV

O modelo de custo de migração baseada em MV, descrito na Equação 5.11, possui três variáveis, M, WWS e V, que representam respectivamente o tamanho de memória da MV, o tamanho da porção de memória que é frequentemente modificada e a vazão de rede entre os nós de origem e destino da migração. Neste contexto, os testes de migração com Xen foram realizados variando o tamanho de memória da MV (M) e o tamanho de memória manipulada constantemente pela aplicação (WWS). Para simular o WWS, utilizou-se o programa MigTest modificando constantemente, a cada iteração, toda a memória alocada pelo processo (S). Levando em consideração que a taxa de modificação da memória é maior que a taxa de transferência da rede, pode-se considerar $WWS = S$.

Primeiramente, utilizaram-se os resultados obtidos com MigTest para analisar isoladamente o comportamento do custo da migração em função de cada uma das variáveis. A ideia é fixar uma das variáveis, M ou WWS, e representar graficamente a variação do custo em função da outra. A fim de facilitar a visualização no gráfico, foram escolhidos apenas alguns poucos valores para a variável fixada.

O gráfico da Figura 7.11 contém os dados de migração fixando o tamanho da máquina virtual M em alguns valores (128, 192, 256, 320, 384 e 512 MBytes) e variando o tamanho do WWS. Através deste gráfico, é possível observar que o comportamento do tempo em função de WWS se mantém para diferentes tamanhos de máquinas virtuais.

Da mesma forma, o gráfico da Figura 7.12 contém os dados de migração fixando o tamanho do WWS em alguns valores (1, 32, 64, 128, 160, 192, 224 e 256 MBytes) e variando o tamanho do máquina virtual (M). Através deste gráfico, é possível observar

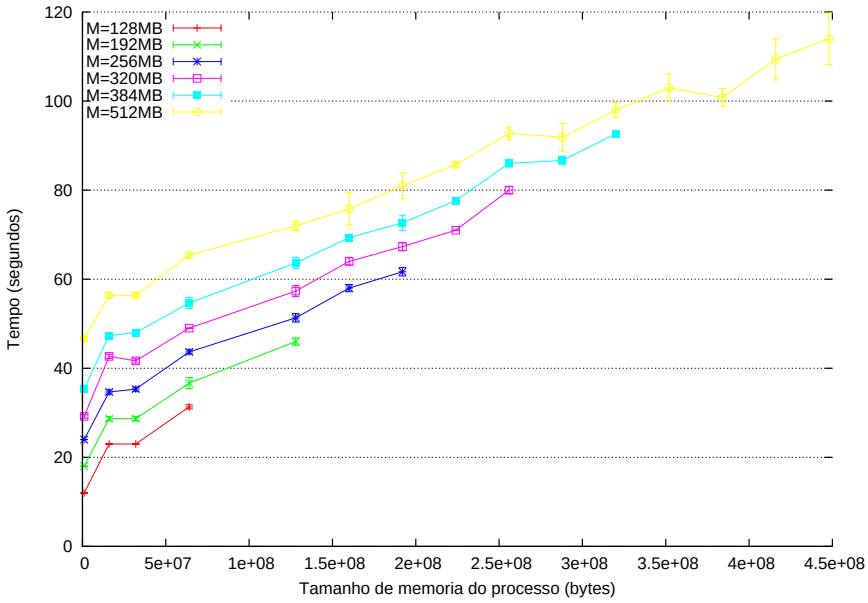


Figura 7.11 Duração de migrações de processo baseadas em MVs variando o tamanho de memória do WWS para diferentes tamanhos de memória de MV (M).

que o comportamento do tempo em função de M se mantém para diferentes tamanhos de WWS (S).

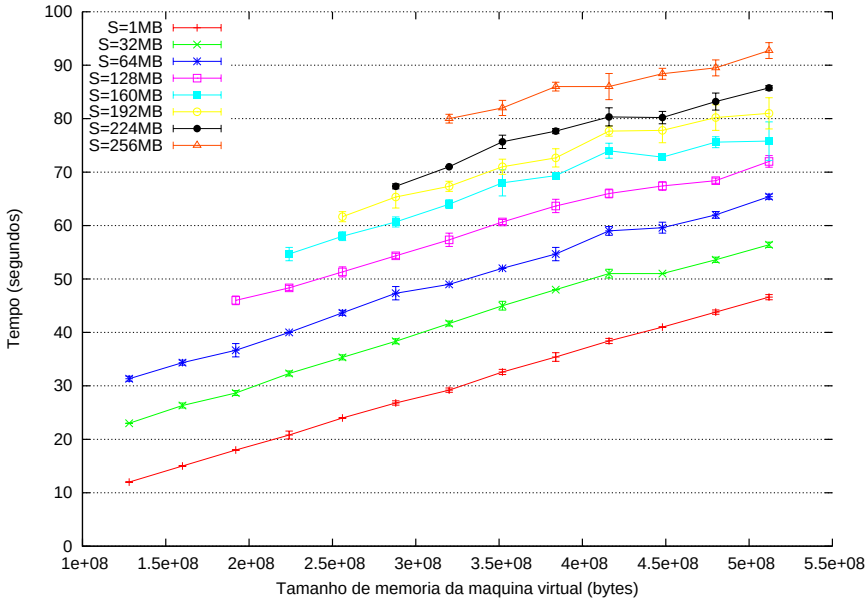


Figura 7.12 Duração de migrações de processo baseadas em MVs variando o tamanho de memória da MV (M) para diferentes tamanhos de WWS (S).

Em um segundo momento, utilizou-se o conjunto completo dos dados obtidos através da execução de MigTest para preencher o modelo de custo de migração baseada em MV. Como resultado, a regressão linear do custo de migração variando M e WWS é dada pela equação

$$C_{ij}(M, WWS) = 9,783 + 8,205 * 10^{-8}(M) + 1,485 * 10^{-7}(WWS) \qquad (7.5)$$

a qual foi obtida com $R^2 = 0,9627$. Desta forma, a Equação 7.5 representa o custo de migração de um processo, que modifica constantemente uma quantidade de memória WWS e executa sobre uma MV com tamanho de memória M, de um nó de origem i para um destino j, interligados por uma rede de 100Mbps.

O gráfico da Figura 7.13 contém a representação da Equação 7.5 em forma de um plano e os resultados experimentais como pontos. A qualidade do modelo pode ser observada pela proximidade entre os pontos do experimento e o plano da equação.

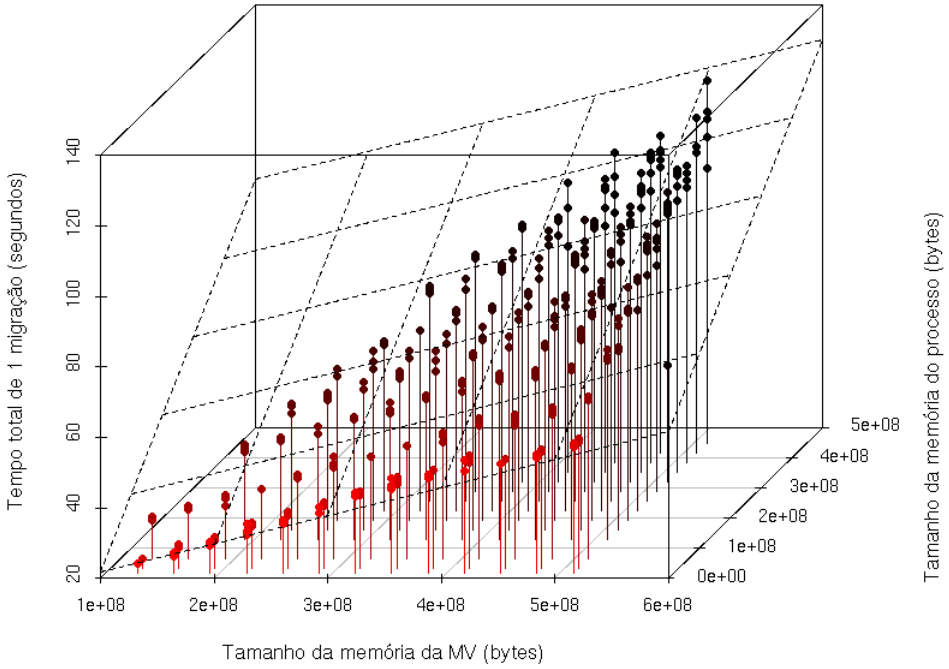


Figura 7.13 Regressão linear de uma migração baseada em MV, variando tamanho da memória da MV (M) e tamanho do WWS.

7.3 Avaliação da Migração de Processos em Aplicações Reais

Esta seção realiza a avaliação da migração de processos em aplicações reais. Conforme apresentado no Capítulo 6, foram realizados testes utilizando as aplicações HPL e SP, esta última do pacote NPB. O objetivo dos testes foi avaliar o impacto de uma migração de processo na execução de aplicações paralelas e a qualidade das previsões de custo dos modelos propostos.

O primeiro teste realizado foi a execução de HPL em 3 nós do agregado, mapeando um processo para cada nó, e a migração de um dos processos para um quarto nó. No caso de Xen, utilizou-se 3 MVs, uma para cada nó, e a migração do processo foi realizada através da migração de sua MV para o quarto nó físico. Em ambos os casos, utilizou-se uma chamada `MPI_Migrateto()` após a segunda iteração do laço principal da aplicação. O resultado desse experimento pode ser observado no gráfico da Figura 7.14.

Os testes foram realizados utilizando MVs com tamanho de memória de 512MBytes

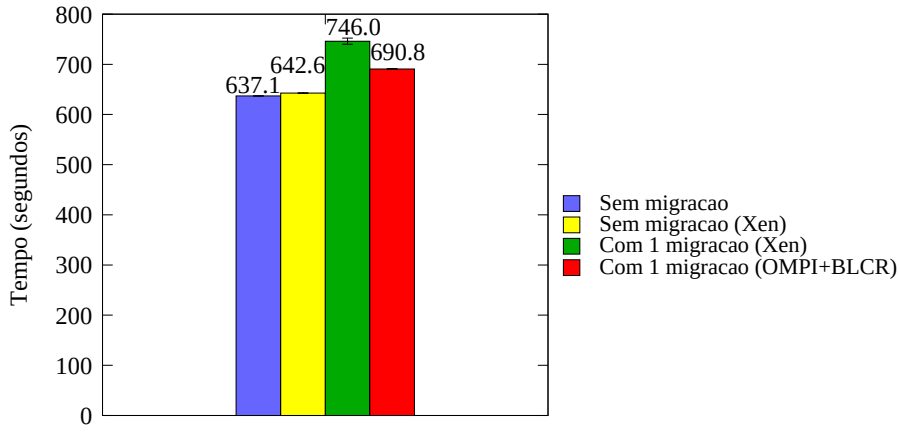


Figura 7.14 Impacto de uma migração de processo na execução da aplicação HPL.

Tabela 7.1 Comparação entre o custo medido e o custo previsto pelo modelo na aplicação HPL.

Abordagem	Custo Medido (s)	Custo Previsto (s)	Acerto (%)
C/R	53,68	52,18	97,1%
MV	108,94	111,19	97,9%

e HPL com um tamanho de problema de N=12000 (tamanho calculado para utilizar toda a memória disponível nas MVs). A duração da execução ficou em torno de 10 minutos, conforme pode ser observado no gráfico da Figura 7.14. Com a realização de uma migração baseada em C/R (OMPI+BLCR), houve um acréscimo de 8,4% no tempo de execução de HPL. No caso de um migração baseada em MV (Xen), o acréscimo foi de 17% no tempo de execução, sendo que o sobrecusto da MV foi de 0,86% (execução sobre Xen sem migração). Embora o sobrecusto da virtualização seja pequeno, o custo de migração de MV na aplicação HPL foi de aproximadamente o dobro do custo de migração baseada em C/R.

Ainda nesses testes, utilizaram-se os modelos propostos para prever o custo de migrar um processo na aplicação HPL. Foram empregadas a Equação 7.4 e a Equação 7.5 para estimar os custos de migração para as abordagens baseadas em C/R e MVs, respectivamente. O tamanho em memória de cada processo HPL é de aproximadamente 400MBytes ($S = 400 * 10^6$) e o tamanho das MVs é de 512MBytes ($M = 512 * 10^6$). Os resultados obtidos são apresentados na Tabela 7.1. A coluna Acerto apresenta, em porcentagem, a precisão da previsão (100% significa que o valor previsto é exatamente igual ao medido). Através da análise desta tabela, pode-se observar que os modelos propostos são capazes de prever valores de custo bastante próximos aos valores medidos na execução.

O segundo teste realizado foi a execução da aplicação SP do pacote NPB, utilizando um problema de classe C. O objetivo principal deste teste foi observar o impacto da migração de processo em uma aplicação que executa por um período maior de tempo. Este tamanho de problema, no ambiente de execução utilizado, garante uma execução de aproximadamente 1 hora e 20 minutos. Foram utilizados 4 nós do agregado, sendo que a migração foi realizada transferindo um dos processos para um quinto nó. No caso de

Xen, cada nó físico executa uma MV e a migração foi realizada transferindo uma das MVs para o quinto nó físico. Assim como no teste com HPL, utilizou-se uma chamada `MPI_Migrateto()` após a segunda iteração do laço principal da aplicação para iniciar a migração. O resultado desse experimento pode ser observado no gráfico da Figura 7.15.

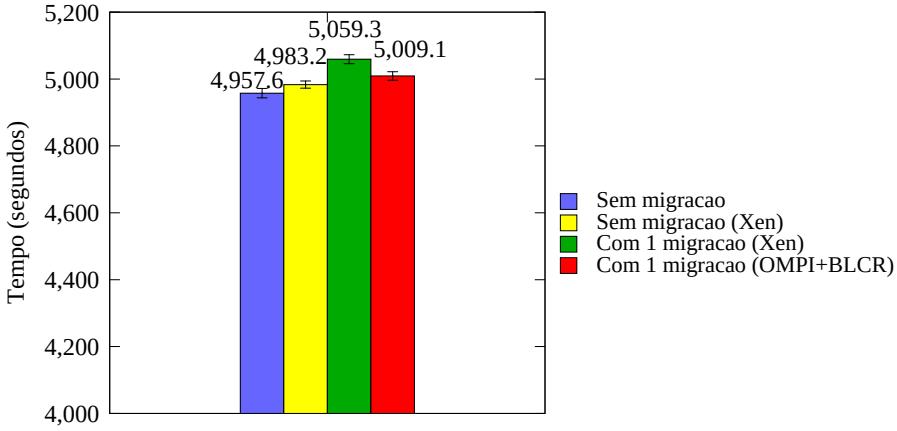


Figura 7.15 Impacto de uma migração de processo na execução da aplicação SP/NPB.

Com a realização de uma migração baseada em C/R (OMPI+BLCR), houve um acréscimo de 1,03% no tempo de execução de SP/NPB. Já no caso de uma migração baseada em MV (Xen), o acréscimo foi de 2,05%. O sobrecusto imposto pela virtualização de Xen foi de apenas 0,51% no tempo de execução (execução sobre Xen sem migração). Comparando o custo de migração baseada em MVs com a migração baseada em C/R, a diferença manteve-se em aproximadamente o dobro do custo para MVs. No entanto, nota-se que o impacto da migração em uma aplicação que executa por um longo período de tempo é menor.

Esses resultados também foram utilizados para avaliar os modelos de custo na aplicação SP/NPB. Foram estimados os custos de migração com ambas as abordagem, levando em consideração o tamanho em memória de cada processo SP/NPB, que é de aproximadamente 360MBytes ($S = 360{*}10^6$), e o tamanho das MVs, de 512MBytes ($M = 512{*}10^6$). Os resultados foram comparados com os valores medidos durante a execução da aplicação. Esta comparação é apresentada na Tabela 7.2. Através desta tabela, é possível observar que a porcentagem de acerto foi menor do que a obtida na aplicação HPL. No entanto, mantém-se acima de 90% o que reafirma a qualidade das previsões dos modelos propostos.

Ambas as aplicações fazem com que o WWS da MV migrante seja bastante próximo ao tamanho de memória do processo (S). Como mencionado anteriormente, isso deve-se ao caráter de computação intensiva deste tipo de aplicação. Nos testes apresentados nesta seção, o custo de migração baseada em MV é praticamente o dobro do custo de migração baseada em C/R. Acredita-se que, em aplicações que S seja menor que o WWS, o custo das migrações baseadas em MVs fique mais próximo ao migrações de baseadas C/R.

7.4 Conclusão

No decorrer deste capítulo, foram apresentados os resultados dos experimentos que avaliaram o sobrecusto das soluções de migração estudadas, estimaram os parâmetros

Tabela 7.2 Comparação entre o custo medido e o custo previsto pelo modelo na aplicação SP/NPB.

Abordagem	Custo Medido (s)	Custo Previsto (s)	Acerto (%)
C/R	51,48	47,44	91,4%
MV	101,73	105,25	96,6%

e verificaram a corretude da modelagem de custo de migração apresentada no Capítulo 5. Também foi observado o impacto de uma migração no tempo total de execução de aplicações reais.

A avaliação do sobrecusto das soluções de migração foi realizada observando isoladamente as partes de comunicação e computação. Em Open MPI, observou-se um aumento na latência de rede quando módulo de C/R está habilitado. No entanto, não foi possível observar um impacto significativo sobre a vazão. O sobrecusto de comunicação mais significativo foi detectado quando coloca-se dois processos no mesmo nó, já que o custo de comunicação é menor e o sobrecusto fica mais evidente neste caso. Também realizou-se uma avaliação das modificações realizadas para otimizar o suporte a C/R de Open MPI. Nos resultados obtidos, foi possível observar que as modificações realizadas melhoraram significativamente o desempenho das operações de *checkpoint* e migração de processo.

No caso de Xen, também avaliou-se a execução completa de uma aplicação MPI sobre o ambiente virtualizado. Os testes de computação revelaram que Xen é competitivo e apresenta tempos próximos, ou melhores, ao sistema nativo (não virtualizado). Nos testes de comunicação foi possível observar uma degradação de desempenho de rede obtida quando coloca-se duas MVs no mesmo nó físico. A execução da aplicação HPL sobre MVs permitiu uma comparação de desempenho com o sistema nativo. O resultado mostrou Xen com um desempenho bastante próximo ao do sistema nativo (diferença de 1,1% no exemplo da Seção 7.1.2.3). O mapeamento de duas MVs por nó físico utilizando HPL também revelou uma degradação no desempenho.

Os experimentos de estimativa de parâmetros confirmaram o comportamento linear dos modelos de custo propostos no Capítulo 5. Como resultado, foram obtidas equações preenchidas para os modelos. A Equação 7.4 descreve o custo de migrações utilizando Open MPI com C/R. A Equação 7.5 descreve o custo de migrações utilizando Xen. Essas equações foram utilizadas para prever o custo de migração nas aplicações HPL e SP/NPB e o resultado foi comparado com os custos de migração reais. Nesta comparação, os valores previstos ficaram bastante próximos dos valores observados no experimento (acima de 90% de acerto nos testes da Seção 7.3).

Por fim, os testes com as aplicações HPL e SP/NPB permitiram avaliar o impacto de uma migração na execução de aplicações reais. Na execução de HPL, que teve uma duração de 10 minutos, uma migração baseada em C/R aumentou em 8,4% o tempo de execução. No caso de uma migração baseada em MV, este acréscimo foi de 17%. Já na execução de SP/NPB, que teve uma duração maior (cerca de 1 hora e 20 minutos), o impacto de uma migração no tempo total de execução foi de 1,03%, para migração baseada em C/R, e de 2,05%, no caso de migração baseada em MV. Com base nesses resultados, acredita-se que o custo de uma migração em aplicações que executam por horas, ou talvez dias, seja ínfimo.

8 CONCLUSÃO

A migração de processos é importante em programas MPI por vários motivos, tais como permitir re-escalonamento de processos e balanceamento de cargas, explorar a localidade de recursos e tolerar falhas. Independente do tipo do uso, conhecer o custo imposto pela realização de uma migração é um problema pertinente. Quando utiliza-se migração para tentar diminuir o tempo de duração de uma aplicação paralela, este custo passa a ser um ponto crítico.

Neste contexto, esse trabalho apresentou um estudo para modelar e dimensionar o custo de migração de processos em programas MPI. Como resultados, foram obtidos subsídios para a realização de re-escalonamento de processos. O trabalho identificou, analisou, adaptou e avaliou as principais soluções disponíveis atualmente para migrar processos MPI. Com base nestas soluções, foram criados modelos de custo que poderão ser utilizados para calcular, dinamicamente, os custos de migração e auxiliar na tomada de decisão em algoritmos de escalonamento. Também foi realizada uma avaliação do impacto de uma migração de processo na execução de aplicações paralelas reais com o objetivo de verificar a viabilidade de utilizar esta abordagem para obter ganho de desempenho.

Primeiramente, foram estudadas as abordagem disponíveis para migração de processos. Os principais representantes de cada abordagem foram analisados com base nos critérios de existência de limitações/restrições de utilização, possibilidade de utilização em programas MPI e a existência de trabalhos científicos nessa área. Deste estudo, foi possível identificar as soluções de migração de processos disponíveis atualmente para aplicações MPI. As abordagens de migração baseada em C/R e migração baseada em MVs foram escolhidas para serem estudadas em detalhes.

O estudo da abordagem de migração baseada em C/R revelou as soluções de LAM/MPI e Open MPI como as mais viáveis para migrar processos MPI. Ambas as soluções são funcionais e foram testadas na prática no agregado do grupo GPPD para migrar processos em aplicações MPI. Apesar de funcional, a migração de processos em LAM/MPI apresenta um custo elevado devido à necessidade de processar os arquivos de contexto. Além disso, os esforços de desenvolvimento de LAM/MPI foram migrados para Open MPI. Open MPI, por outro lado, permite reiniciar a aplicação com um novo mapeamento sem a necessidade de modificar os arquivos de contexto. A implementação do suporte a C/R foi modificada para eliminar a transferência de arquivos de contexto para um servidor de armazenamento persistente. Assim, tornou-se possível a utilização deste suporte na migração de processos.

O estudo da abordagem baseada em MVs revelou Xen como a solução mais viável. Xen utiliza a técnica de paravirtualização para oferecer um desempenho bastante próximo ao sistema nativo. A migração de máquinas MV oferecida por Xen é do tipo *live migra-*

tion e permite migrar processos de um nó para outro sem a necessidade de interromper a execução da aplicação. Esta migração ocorre com um tempo de indisponibilidade pequeno e todas as conexões de rede são mantidas de forma transparente. Desta forma, evitam-se os problemas clássicos da migração a nível de processo.

Após o estudo dessas soluções, foi realizada uma avaliação do sobrecusto de cada uma, observando isoladamente as partes de comunicação e computação. Em Open MPI, não foi possível observar um sobrecusto significativo na maioria dos casos. O sobrecusto de comunicação mais significativo foi detectado quando se colocam dois processos no mesmo nó. Neste caso, o custo de comunicação é menor e o sobrecusto fica mais evidente. No caso de Xen, os testes de computação revelaram que Xen é competitivo e apresenta tempos próximos, ou melhores, ao sistema nativo (não virtualizado). Nos testes de comunicação foi possível observar uma degradação de desempenho de rede obtida quando se colocam duas MVs no mesmo nó físico.

A avaliação de Xen também envolveu a execução completa de uma aplicação paralela sobre o ambiente virtualizado. O objetivo deste teste foi verificar o sobrecusto na virtualização na execução de uma aplicação deste tipo. O resultado mostrou Xen com um desempenho bastante próximo ao do sistema nativo (diferença de 1,1% no exemplo da Seção 7.1.2.3). No entanto, o mapeamento de duas MVs por nó físico utilizando HPL revelou uma degradação no desempenho da aplicação.

Com base no estudo das soluções de migração, foram definidos modelos de custo de migração. Inicialmente, foi definido um modelo para descrever a execução de aplicações MPI que realizam migração de processos. Este modelo leva em consideração tanto ambientes compostos por recursos homogêneos (Equação 5.3) quanto heterogêneo (Equação 5.2). Na sequencia, foram definidos modelos de custo de uma migração para as duas abordagens estudadas nesse trabalho. A Equação 5.4 representa o modelo de custo para um mecanismo genérico de migração baseada em C/R. Já a Equação 5.8 representa o modelo de custo de migração utilizando um mecanismo genérico de migração de MVs. Estes modelos foram derivados, com base no estudo das soluções de migração, para gerar os modelos específicos para cada solução.

Através da estimativa de parâmetros, obtiveram-se modelos preenchidos que podem ser utilizados para calcular, de antemão, o custo de migrar um processo em uma dada aplicação. A Equação 7.4 descreve o custo de migrações utilizando Open MPI com C/R. A Equação 7.5 descreve o custo de migrações utilizando Xen. A qualidade das previsões dos modelos é importante, principalmente, quando utilizadas na tomada de decisão em algoritmos de escalonamento. Neste caso, uma previsão errada pode prejudicar o desempenho final do escalonamento. Essas equações foram utilizadas para prever o custo de migração nas aplicações HPL e SP/NPB e o resultado foi comparado com os custos de migração reais. Nesta comparação, os valores previstos ficaram bastante próximos dos valores observados no experimento (acima de 90% de acerto nos testes da Seção 7.3).

O trabalho também apresentou uma análise do impacto da migração de processos sobre a execução de aplicações do tipo MPI. Os resultados mostraram que o sobrecusto da migração baseada em MVs é praticamente o dobro da migração baseada em C/R. No entanto, acredita-se que em aplicações que o WWS seja menor que o tamanho do processo S os custos fiquem mais próximos. Na execução de HPL, que teve uma duração de 10 minutos, uma migração baseada em C/R aumentou em 8,4% o tempo de execução. No caso de uma migração baseada em MV, este acréscimo foi de 17%. Já na execução de SP/NPB, que teve uma duração maior (cerca de 1 hora e 20 minutos), o impacto de uma migração no tempo total de execução foi de 1,03%, para migração baseada em C/R, e de

2,05%, no caso de migração baseada em MV.

8.1 Resumo das Contribuições

Dentre as contribuições deste trabalho de mestrado, pode-se citar:

- Estudo sobre os mecanismos disponíveis atualmente para migração de processos em programas MPI;
- Modificação do suporte a C/R de Open MPI para permitir migração de processos;
- Estudo sobre a exploração da virtualização de recursos para realizar migração em programas MPI (publicado no WSCAD 2007 (NEVES et al., 2007))
- Definição de modelos para estimar o custo de migração de processos;
- Análise do impacto de migração de processo em aplicações reais e da viabilidade de utilização desta abordagem para obter ganho de desempenho;
- Criação de uma metodologia para avaliação do custo de migração de processos em programas MPI que pode ser utilizada para avaliar outras soluções;
- Agregação de conhecimento e subsídios de migração de processos para a linha de pesquisa sobre escalonamento de processos do grupo GPPD.

8.2 Trabalhos Futuros

A evolução natural deste trabalho é a aplicação dos resultados obtidos na realização de re-escalonamento de processos MPI. No que diz respeito à integração com os trabalhos do grupo GPPD, pretende-se utilizar os modelos propostos para estimar custos de migração e auxiliar MigBSP na tomada de decisão do escalonamento. Também é possível utilizar a metodologia de avaliação deste trabalho para avaliar outras soluções de migração que possam ser desenvolvidas ou utilizadas pelo grupo.

Como trabalhos futuros, pretende-se ampliar a avaliação para incluir os modelos propostos para ambientes heterogêneos ou ainda novos parâmetros para os modelos. Também pretende-se analisar o custo de múltiplas migrações na execução de aplicações MPI. Outra possibilidade é utilizar os modelos propostos para comparar as duas abordagens de migração. Por fim, pretende-se avaliar a possibilidade de integrar o suporte a migração de processos implementado para Open MPI na sua distribuição oficial.

REFERÊNCIAS

ADAMS, K.; AGESEN, O. A Comparison of Software and Hardware Techniques for x86 Virtualization. In: ARCHITECTURAL SUPPORT FOR PROGRAMMING LANGUAGES AND OPERATING SYSTEMS, 12., 2006, New York, NY, USA. **Proceedings...** ACM Press, 2006. p.2–13.

ARGENTINI, G. Use of openMosix for Parallel I/O Balancing on Storage in Linux Cluster. In: WORKSHOP LINUX CLUSTER: THE OPENMOSIX APPROACH, 2002. **Proceedings...** [S.l.: s.n.], 2002.

BAILEY, D. H.; BARSZCZ, E.; BARTON, J. T.; BROWNING, D. S.; CARTER, R. L.; DAGUM, L.; FATOOHI, R. A.; FREDERICKSON, P. O.; LASINSKI, T. A.; SCHREIBER, R. S.; SIMON, H. D.; VENKATAKRISHNAN, V.; WEERATUNGA, S. K. The NAS Parallel Benchmarks - Sumary and Preliminary Results. In: ACM/IEEE CONFERENCE ON SUPERCOMPUTING, 1991., 1991, New York, NY, USA. **Proceedings...** ACM Press, 1991. p.158–165.

BAKER, M.; BUYYA, R. Cluster Computing at a Glance. In: BUYYA, R. (Ed.). **High Performance Cluster Computing**. Upper Saddle River, NJ: Prentice Hall PTR, 1999. v.1, Architectures and Systems, p.3–47. Chap. 1.

BAR, M.; RECHENBURG, M. OpenMosix. In: INTERNATIONAL LINUX SYSTEM TECHNOLOGY CONFERENCE, 10., 2003, Saarbrucken, Germany. **Proceedings...** [S.l.: s.n.], 2003.

BARHAM, P.; DRAGOVIC, B.; FRASER, K.; HAND, S.; HARRIS, T.; HO, A.; NEUGEBAUER, R.; PRATT, I.; WARFIELD, A. Xen and The Art of Virtualization. In: ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, 2003, New York. **Proceedings...** ACM Press, 2003. p.164–177. (Operating Systems Review, v.37, 5).

BOSILCA, G.; BOUTEILLER, A.; CAPPELLO, F.; DJILALI, S.; FEDAK, G.; GERMAIN, C.; HÉRAULT, T.; LEMARINIER, P.; LODYGENSKY, O.; MAGNIETTE, F.; NERI, V.; SELIKHOV, A. MPICH-V: Toward a Scalable Fault Tolerant MPI for Volatile Nodes. In: IEEE SUPERCOMPUTING 2002, 2002. **Proceedings...** [S.l.: s.n.], 2002.

BOUFLEUR, M. P.; KOSLOVSKI, G. P.; CHARÃO, A. S. Avaliação do Uso de Xen em Ambientes de Alto Desempenho. In: WORKSHOP EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO - WSCAD 2006, 2006, Ouro Preto - MG. **Anais...** [S.l.: s.n.], 2006. p.141–147.

BOUTEILLER, A.; CAPPELLO, F.; HÉRAULT, T.; KRAWEZIK, G.; LEMARINIER, P.; MAGNIETTE, F. MPICH-V2: a fault tolerant MPI for volatile nodes based on pessimistic sender based message logging. In: SUPER COMPUTING 2003, 2003. **Proceedings...** [S.l.: s.n.], 2003.

BUYYA, R.; CORTES, T.; JIN, H. Single System Image. **The International Journal of High Performance Computing Applications**, [S.l.], n.2, p.124–135, Summer 2001.

CAO, J.; LI, Y.; GUO, M. Process Migration for MPI Applications based on Coordinated Checkpoint. In: INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED SYSTEMS, 11., 2005. **Proceedings...** IEEE Computer Society Press, 2005. p.306–312.

CASAVANT, T.; KUHL, J. A Taxonomy of Scheduling in General-purpose Distributed Computing Systems. **IEEE Transactions on Software Engineering**, Los Alamitos, CA, USA, v.14, n.2, p.141–154, 1988.

CERA, M. C.; PEZZI, G. P.; MATHIAS, E. N.; MAILLARD, N.; NAVAUX, P. O. A. Improving the Dynamic Creation of Processes in MPI-2. In: LECTURE NOTES IN COMPUTER SCIENCE - 13TH EUROPEAN PVMMPI USERS GROUP MEETING, 2006, Bonn, Germany. **Anais...** Springer Berlin / Heidelberg, 2006. v.4192/2006, p.247–255.

CHANDY, K. M.; LAMPORT, L. Distributed Snapshots: determining global states of distributed systems. **ACM Transactions on Computer Systems**, [S.l.], v.3, n.1, p.63–75, Feb 1985.

CHASSIN DE KERGOMMEAUX, J.; OLIVEIRA STEIN, B. de. Pajé: an extensible environment for visualizing multi-threaded programs executions. **Lecture Notes in Computer Science**, [S.l.], v.1900, p.133–153, 2001.

CLARK, C.; FRASER, K.; HAND, S.; HANSEN, J. G.; JUL, E.; LIMPACH, C.; PRATT, I.; WARFIELD, A. Live Migration of Virtual Machines. In: USENIX SYMPOSIUM ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION, 2., 2005, Boston, MA, USA. **Proceedings...** [S.l.: s.n.], 2005. p.20–34.

COTI, C.; HERAULT, T.; LEMARINIER, P.; PILARD, L.; REZMERITA, A.; RODRIGUEZ, E.; CAPPELLO, F. Blocking vs. non-blocking coordinated checkpointing for large-scale fault tolerant MPI. In: ACM/IEEE CONFERENCE ON SUPERCOMPUTING, 2006., 2006, New York, NY, USA. **Proceedings...** ACM, 2006. p.127.

DESHANE, T.; SHEPHERD, Z.; MATTHEWS, J.; BEN-YEHUDA, M.; SHAH, A.; RAO, B. Quantitative Comparison of Xen and KVM. **Xen Summit North America 2008**, [S.l.], 2008.

DONGARRA, J. J.; STEWART, G. W. LINPACK — A Package for Solving Linear Systems. In: **Sources and Development of Mathematical Software**. [S.l.: s.n.], 1984. p.20–48.

DU, C.; SUN, X.-H.; WU, M. Dynamic Scheduling with Process Migration. **Cluster Computing and the Grid, IEEE International Symposium on**, Los Alamitos, CA, USA, v.0, p.92–99, 2007.

DUDA, K. J.; CHERITON, D. R. Borrowed-virtual-time (BVT) Scheduling: supporting latency-sensitive threads in a general-purpose scheduler. **Operating Systems Review**, [S.l.], v.33, n.5, p.261–276, Dec. 1999.

DUELL, J.; HARGROVE, P.; ROMAN, E. **The Design and Implementation of Berkeley Lab's Linux Checkpoint/Restart**. [S.l.]: Berkeley Lab Technical Report (LBNL-54941), 2002.

DUPROS, F.; CARISSIMI, A.; MEHAUT, J. F. Desempenho de Operaçoes de Checkpoint/Restart em Aplicaçoes MPI. In: WORKSHOP EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO (WSCAD'2006), 2006. **Anais...** [S.l.: s.n.], 2006.

GOTH, G. Virtualization: old technology offers huge new potential. **IEEE Distributed Systems Online**, Piscataway, NJ, USA, v.8, n.2, p.3, 2007.

GROPP, W.; LUSK, E.; DOSS, N.; SKJELLUM, A. High-performance, Portable Implementation of the MPI Message Passing Interface Standard. **Parallel Computing**, [S.l.], v.22, n.6, p.789–828, 1996.

HARCHOL-BALTER, M.; DOWNEY, A. B. Exploiting Process Lifetime Distributions for Dynamic Load Balancing. **ACM Transactions on Computer Systems**, [S.l.], v.15, p.13–24, 1997.

HUANG, C.; LAWLOR, O.; KALÉ, L. V. Adaptive MPI. In: INTERNATIONAL WORKSHOP ON LANGUAGES AND COMPILERS FOR PARALLEL COMPUTING (LCPC 2003), LNCS 2958, 16., 2003, College Station, Texas. **Proceedings...** [S.l.: s.n.], 2003. p.306–322.

HUANG, C.; ZHENG, G.; KUMAR, S.; KALÉ, L. V. Performance Evaluation of Adaptive MPI. In: ACM SIGPLAN SYMPOSIUM ON PRINCIPLES AND PRACTICE OF PARALLEL PROGRAMMING 2006, 2006. **Proceedings...** [S.l.: s.n.], 2006.

HURSEY, J.; SQUYRES, J. M.; MATTOX, T. I.; LUMSDAINE, A. The Design and Implementation of Checkpoint/Restart Process Fault Tolerance for Open MPI. In: IEEE International Parallel and Distributed Processing Symposium (IPDPS), 21., 2007. **Proceedings...** IEEE Computer Society, 2007.

KERRIGHED. **Kerrighed**. Disponível em <http://www.kerrighed.org>. Acesso em: dez. 2006.

KHAN, M. S. **A Survey of Open Source Cluster Management Systems**. Disponível em <http://www.linux.com/article.pl?sid=06/09/12/1459204>. Acesso em: dez. 2006.

KVM. **KVM Home Page**. Disponível em <http://kvm.qumranet.com/>. Acesso em: jan. 2009.

LAM/MPI. **LAM/MPI Parallel Computing**. Disponível em <http://www.lam-mpi.org>. Acesso em: fev. 2008.

LITZKOW, M.; TANNENBAUM, T.; BASNEY, J.; LIVNY, M. **Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System**. [S.l.]: Technical Report CS-TR-1997-1346. University of Wisconsin., 1997.

LOTTIAUX, R.; GALLARD, P.; VALLEE, G.; MORIN, C.; BOISSINOT, B. OpenMosix, OpenSSI and Kerrighed: a comparative study. In: FIFTH IEEE INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID (CCGRID'05) - VOLUME 2, 2005, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2005. p.1016–1023.

MELLO, R. F. D.; SENGER, L. J. A New Migration Model based on the Evaluation of Processes Load and Lifetime on Heterogeneous Computing Environments. In: SYMPOSIUM ON COMPUTER ARCHITECTURE AND HIGH PERFORMANCE COMPUTING, 16., 2004, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2004. p.222–227.

MILOJICIC, D. S.; DOUGLIS, F.; PAINDAVEINE, Y.; WHEELER, R.; ZHOU, S. Process Migration. **ACM Computing Surveys**, Cambridge, MA 02142, Sept. 2000.

MORIN, C.; LOTTIAUX, R.; VALLÉE, G.; GALLARD, P.; UTARD, G.; BADRINATH, R.; RILLING, L. Kerrighed: a single system image cluster operating system for high performance computing. In: EUROPAR 2003: PARALLEL PROCESSING, 2003. **Proceedings...** Springer Verlag, 2003. p.1291–1294. (Lect. Notes in Comp. Science, v.2790).

MPICH-V. **MPICH-V Home Page**. Disponível em <http://mpich-v.lri.fr>. Acesso em: fev. 2008.

MPICH2. **MPICH2 Home Page**. Disponível em <http://www.mcs.anl.gov/research/projects/mpich2/>. Acesso em: fev. 2008.

NAGARAJAN, A.; MUELLER, F.; ENGELMANN, C.; SCOTT, S. L. Proactive Fault Tolerance for HPC with Xen Virtualization. In: ACM INTERNATIONAL CONFERENCE ON SUPERCOMPUTING (ICS) 2007, 21., 2007, Seattle, WA, USA. **Proceedings...** [S.l.: s.n.], 2007.

NAVAUX, P. O. A.; DE ROSE, C. A. F. **Arquiteturas Paralelas**. 1.ed. [S.l.]: Editora Sagra Luzzato, 2003. n.15. (Série Livros Didáticos).

NEVES, M. V.; ROSA RIGHI, R. da; MAILLARD, N.; NAVAUX, P. O. A. Impacto da Migração de Máquinas Virtuais de Xen na Execução de Programas MPI. In: WORKSHOP EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO (WSCAD'2007), 2007. **Anais...** [S.l.: s.n.], 2007.

OPENMOSIX. **openMosix**. Disponível em <http://openmosix.sourceforge.net>. Acesso em: dez. 2006.

OPENMPI. **Open MPI**: Open Source High Performance Computing. Disponível em <http://www.open-mpi.org>. Acesso em: fev. 2008.

OPENSSI. **OpenSSI**. Disponível em <http://openssi.org>. Acesso em: dez. 2006.

PETITET, A.; WHALEY, R. C.; DONGARRA, J.; CLEARY, A. **HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers**. Disponível em <http://www.netlib.org/benchmark/hpl/>. Acesso em: jun. 2007.

PLANK, J. S.; BECK, M.; KINGSLEY, G.; LI, K. **Libckpt**: transparent checkpointing under Unix. In: USENIX WINTER TECHNICAL CONFERENCE, 1995. **Proceedings...** [S.l.: s.n.], 1995. p.213–223.

PLUMMER, D. An Ethernet Address Resolution Protocol. **RFC 826**, [S.l.], Nov. 1982.

QEMU. **QEMU Home Page**. Disponível em <http://bellard.org/qemu>. Acesso em: jan. 2009.

R Project. **R: a language and environment for statistical computing**. Vienna, Austria: R Foundation for Statistical Computing, 2008.

RIGHI, R. d. R.; PILLA, L. L.; CARISSIMI, A.; NAVAUX, P. O. A. Controlling Processes Reassignment in BSP Applications. In: INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE AND HIGH PERFORMANCE COMPUTING, 2008., 2008, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2008. p.37–44.

ROMAN, E. **A Survey of Checkpoint/Restart Implementations**. [S.l.]: Berkeley Lab Technical Report (LBNL-54942), 2002.

SANKARAN, S.; SQUYRES, J. M.; BARRETT, B.; LUMSDAINE, A.; DUELL, J.; HARGROVE, P.; ROMAN, E. The LAM/MPI Checkpoint/Restart Framework: system-initiated checkpointing. **International Journal of High Performance Computing Applications**, [S.l.], v.19, n.4, p.479, Winter 2005.

SILVA, G. J. da; OLIVEIRA STEIN, B. de. Uma Biblioteca Genrica de Gerao de Rastros de Execuo para Visualizao de Programas. In: I SIMPSIO DE INFORMTICA DA REGIO CENTRO, 2002. **Anais...** [S.l.: s.n.], 2002.

SILVA MARTINS JR., A. da; GONÇALVES, R. A. L. Extensões na LAM/MPI para Automatizar o Checkpoint e Tolerar Falhas em Cluster de Computadores. In: WORKSHOP EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO (WSCAD'2007), 2005. **Anais...** [S.l.: s.n.], 2005.

SILVA, R. E.; PEZZI, G.; MAILLARD, N.; DIVERIO, T. Automatic Data-Flow Graph Generation of MPI Programs. In: INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE ON HIGH PERFORMANCE COMPUTING, 17., 2005, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2005. p.93–100.

SMITH, J. M. A Survey of Process Migration Mechanisms. **Operating Systems Review (OSR)**, [S.l.], v.22, n.3, p.28–40, July 1988. CS Dept., Columbia, U.

SNELL, Q.; MIKLER, A.; GUSTAFSON, J. **NetPIPE**: A Network Protocol Independent Performace Evaluator. 1996.

TANENBAUM, A. **Computer Networks**. 4.ed. [S.l.]: Prentice Hall, 2003.

TANENBAUM, A. S. **Modern Operating Systems**. [S.l.]: Prentice-Hall, International, 1992. (Prentice-Hall International Editions). TAN a 92:1 P-Ex.

WALTERS, J. P.; CHAUDHARY, V. **A Comprehensive User-level Checkpointing Strategy for MPI Applications**. [S.l.]: Department of Computer Science and Engineering, University at Buffalo, SUNY, 2007.

WANG, C.; MUELLER, F.; ENGELMANN, C.; SCOTT, S. L. A Job Pause Service under LAM/MPI+BLCR for Transparent Fault Tolerance. In: IEEE INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM (IPDPS) 2007, 21., 2007, Long Beach, CA, USA. **Proceedings...** [S.l.: s.n.], 2007.

WANG, C.; MUELLER, F.; ENGELMANN, C.; SCOTT, S. L. Proactive Process-level Live Migration in HPC Environments. In: SC '08: PROCEEDINGS OF THE 2008 ACM/IEEE CONFERENCE ON SUPERCOMPUTING, 2008, Piscataway, NJ, USA. **Anais...** IEEE Press, 2008. p.1–12.

WHITAKER, A.; SHAW, M.; GRIBBLE, S. Denali: lightweight virtual machines for distributed and networked applications. In: USENIX SYMPOSIUM ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION, 5., 2002. **Proceedings...** [S.l.: s.n.], 2002. p.195–209.

YOUSEFF, L.; WOLSKI, R.; GORDA, B. C.; KRINTZ, C. Paravirtualization for HPC Systems. In: ISPA WORKSHOPS, 2006. **Proceedings...** Springer, 2006. p.474–486. (Lecture Notes in Computer Science, v.4331).

ZHONG, H.; NIEH, J. **CRAK**: linux checkpoint/restart as a kernel module. [S.l.]: Department of Computer Science, Columbia University, 2001.