Desempenho de operações de checkpoint/restart em aplicações MPI

Fabrice Dupros (fdupros@brgm.fr)
Bureau de Recherches Géologiques et Minières (BRGM)
3 av. C. Guillemin - BP6009 Orléans Cedex2 - França.

Alexandre Carissimi (asc@inf.ufrgs.br)
Instituto de Informática - UFRGS
Av. Bento Gonçalves 9500 - CP15064 - Porto Alegre, Brasil.

Jean-François Méhaut (mehaut@imag.fr)
Laboratoire Informatique et Distribution (ID)-IMAG/ENSIMAG
ZIRST 51, avenue Jean Kuntzmann - 38330 Montbonnot Saint Martin - França

Resumo

É comum em intranets coorporativas que as máquinas usadas como ponto de trabalho fiquem ociosas fora do horário de expediente. O projeto RNTL-IGGI tem por objetivo recuperar as máquinas em seus períodos de ociosidade para comporem um cluster virtual e executarem aplicações durante esse período. Entretanto, se a aplicação possuir uma duração de execução superior ao período de ociosidade é necessário salvar o processamento feito até então para retomá-lo no próximo período de ociosidade. Nesse contexto, o uso de mecanismos de checkpoint/restart *surge como uma possibilidade para solucionar o problema de discontinuidade do período de processamento. Neste artigo é apresentado a solução proposta no projeto IGGI e discute-se os principais fatores que influenciam o tempo necessário ao* checkpointing, *em especial, em aplicações paralelas baseadas em MPI.*

1 Introdução

Uma realidade bastante comum em vários organismos de um certo porte, como por exemplo, empresas e universidades, é a existência de uma rede cooporativa (intranet) composta por muitos computadores. Nesses mesmos organismos, também é comum que esses recursos computacionais fiquem ociosos fora do horário do expediente. Cita-se, por exemplo, a situação do organismo francês de pesquisas em geociências, BRGM (*Bureau de Recherches Géologiques et Minières*), que possui cerca de 1700 máquinas distribuídas em 25 prédios que são usadas das 8h00 às 12h00 e das 14h00 às 18h00. Esses pontos de trabalhos ficam ociosos durante a noite, finais de semana e feriados. Em uma semana normal de trabalho, no BRGM, o período de ociosidade é cerca de 128 horas, o equivalente a 5.3 dias. Com base nessa constatação surgiu o projeto IGGI (*Infrastructure pour Grappes,Grilles sur Intranet*-http://iggi.imag.fr) que tem por objetivo definir clusters e grids virtuais durante os períodos de ociosidade da intranet do BRGM. Essa possibilidade é particularmente interessante para o caso do BRGM, já que grande parte de suas aplicações são programas paralelos MPI que apresentam um grande consumo de poder de processamento.

A idéia de empregar recursos ociosos segue o mesmo princípio dos projetos SETIhome[1] e XtremWeb[2]. Entretanto a novidade reside em dois aspectos. Primeiro, os pontos de trabalhos normalmente são usados com o sistema operacional Windows e no início de um período de ociosidade elas são reinicializadas (*reboot*) em Linux e passam a integrar um cluster virtual. Segundo, é necessário evitar que o processamento realizado seja perdido antes de restituir a máquina ao seu uso normal de expediente e/ou para ser retomado em um próximo período de ociosidade. É nesse contexto que surge o interesse de utilizar mecanismos de *checkpoint/restart* como forma de escalonamento *batch*, salvando o contexto de uma aplicação em um instante de tempo e retomando mais tarde sua execução a partir desse ponto. Para que esse procedimento seja possível é importante prever o tempo necessário para realizar as operações de *checkpoint* já que o sistema deve automaticamente, antes do início do expediente, salvar o processamento realizado durante a noite para após reinicializar a máquina e disponibilizá-la ao seu usuário.

Este artigo está organizado em cinco seções, além desta introdução. Na seção 2 discute-se uma série de aspectos re-

lacionados com *checkpointig* que foram estudados durante o desenvolvimento do projeto IGGI. A seguir, na seção 3, são apresentados os principais componentes da infraestrutura de cluster virtual implementados e instalados no BRGM. Na seção 4 discute-se os principais resultados experimentais relacionados ao dimensionamento do tempo de *checkpoint/restart*. Introduz-se ainda, brevemente, as principais aplicações de geociências (riscos sísmicos, gestão de recursos naturais etc) utilizadas no BRGM e empregadas nesta parte experimental. Por fim, na última parte, conclui-se com uma avaliação e trabalhos futuros.

2 Checkpointing

Por *Checkpointing* entende-se o salvamento, em um meio de armazenamento permanente, de uma imagem do estado de execução de uma aplicação em um dado instante de tempo *t*. Com base nessa imagem é possível reconstruir o estado de um processo e de suas estruturas de dados e assim retomar sua execução a partir do ponto onde o *checkpoint* foi feito. Devido a esse comportamento, essa técnica é comumente denominada de *checkpoint/restart* (CR) sendo empregada em três atividades principais:

- Tolerância a falhas: é possível realizar periodicamente o *checkpoint* de uma aplicação e, em caso de falha no sistema, após sua reinicialização, retomar sua execução a partir do ponto (estado) em que foi feito feito o último *checkpoint* válido. Isso assume vital importância para aplicações que possuem um grande tempo de processamento.
- Escalonamento: como *checkpoint* é o salvamento de um estado de execução de um processo (aplicação) é possível que uma aplicação seja parada em qualquer ponto de sua execução e posteriormente retomada. Isso é similar escalonamento de curto prazo de um sistema operacional. A diferença entre eles está no fato que *checkpoint* é o salvamento de uma imagem da aplicação (contexto) em um meio permanente, como por exemplo, um arquivo em disco, ao passo que o salvamento de contexto feito por um escalonamento de sistema operacional é mantido em memória.
- Migração de processos: o *checkpoint* sendo uma imagem de um estado de execução de um processo (aplicação) é possível imaginar que o mesmo possa ser reinicializado em uma máquina distinta daquela em que o *checkpoint* foi feito. Essa possibilidade é interessante em várias situações, como por exemplo, parada de máquinas para *upgrades* de sistemas, falhas, ou ainda, como mecanismo de balanceamento de carga.

O mecanismo de *checkpointing* pode ser implementado em três níveis diferentes: aplicação, biblioteca ou núcleo. Por *checkpoint* em nível de aplicação entende-se aquele onde a própria aplicação é responsável por registrar todos os valores necessários para que sua execução seja retomada em um determinado ponto. Essa retomada é feita recuperando esses valores, reinicializando suas estruturas de dados e relançando a aplicação de uma forma apropriada. A principal vantagem é que a aplicação sabe o que é realmente é importante salvar para retomar uma execução, o que pode levar a um volume de dados registrados menor do que, por exemplo, efetuar uma imagem completa do estado do processo. Isso se reflete diretamente no tempo necessário ao *checkpointing*. Outra vantagem é a sua portabilidade. Por outro lado, essa técnica apresenta uma série de desvantagens. Primeiro, é necessário modificar o código fonte do aplicativo para incluir pontos de *checkpoint*, o que nem sempre é possível por eventualmente não se ter acesso aos fontes. Segundo, o procedimento de retomada deve ser coerente com o feito no *checkpoint*, o que implica em um tratamento caso a caso, ou seja, não é genérico. Terceiro, introduzir pontos de *checkpoint* não é tão simples quanto parece pois é preciso verificar, entre outros, se não há seções críticas, bufferizações no núcleo do sistema operacional etc. Em função dessas desvantagens, o *checkpoint* em nível de aplicação é raramente empregado.

Uma solução para contornar alguns desses problemas é inserir o mecanismo de *checkpoint/restart* dentro de bibliotecas usadas pelo programa. Nesse caso, para passar a utilizar *checkpoint/restart* basta religar o código objeto da aplicação com a biblioteca instrumentada. É necessário apenas ter-se acesso ao código objeto da aplicação. Dependendo da implementação da biblioteca, o *checkpoint* pode ser realizado automaticamente quando se executa todas suas funções, algumas delas, ou ainda, periodicamente. A desvantagem do *checkpoint* em nível de biblioteca é que a capacidade de *checkpoint* é limitada às funções que contemplam essa funcionalidade. São exemplos, entre outros, de implementações baseadas nessa técnica: libckpt[3],Condor[4], Score[5] e libtckpt[6].

Por último, a implementação *checkpoint* em nível de núcleo, ou seja, incluir no próprio *kernel* do sistema operacional suporte a esse mecanismo. A principal vantagem desse método é o acesso privilegiado às estruturas de dados empregadas para a gerência de processos como o estado da pilha, cópia dos dados das regiões estáticas e dinâmicas (*heap*) e os valores dos registradores de hardware do processador. Nessa abordagem, o *checkpointing* é completamente transparente para a aplicação, porém há três desvantagens a serem consideradas. Primeira, o tamanho do arquivo de *checkpoint* tende a ser grande o que, consequentemente, se traduz em um aumento no tempo necessário para realizá-lo. Segundo, por se basear em estruturas internas do núcleo, é estabelecido uma forte dependência com esse, limitando seu emprego a máquinas que possuam uma mesma família de

kernel e/ou de estruturas de dados. Por fim, é necessário privilégios de *root* para configurá-lo no sistema. Como exemplos de implementações que seguem essa abordagem, cita-se, VMADump[7], CRACK[8], BLCR[9] e Cryopid[10].

Embora o mecanismo de *checkpoint/restart* seja relativamente simples de compreender e utilizar em aplicações seqüenciais, o mesmo não ocorre com aplicações paralelas que empregam comunicações baseadas em TCP/IP. Uma conexão TCP é caracterizada pela 4-tupla {endereço IP fonte, porta fonte, endereço IP destino, porta destino} os quais são identificadores locais a uma máquina. Além disso, o controle de fluxo e de erro do TCP é baseado em números de seqüência que são negociados no momento do estabelecimento da conexão (segmentos SYN e ACK), junto com a largura da janela de recepção. Esses valores também são locais a uma máquina. Nada garante que na reinicialização de uma aplicação (após *reboot*), na retomada em momento posterior, ou ainda, no lançamento dessa aplicação em outra máquina que esses valores serão os mesmos. Outra dificuldade que surge é a necessidade de considerar a bufferização de mensagens pela pilha de protocolos a fim de evitar que mensagens consideradas como enviadas pela aplicação estejam na realidade bufferizadas no sistema. Essa é a origem do problema denominado de *checkpoint* consistente[18].

Na realidade, o problema de manutenção de conexões TCP surgiu com as aplicações móveis e, nesse contexto, várias propostas foram feitas para resolvê-lo. Basicamente, essas propostas podem ser classificadas em duas estratégias: modificar a pilha de protocolos ou refazer, do zero, as conexões. São exemplos da primeira abordagem, entre outros, Migrate[11], MobileIP[12] e VNAT[13]. Migrate propõe a modificação da sintaxe e da semântica do estabelecimento de uma conexão TCP através da criação de um identificador adicional, denominado de *token*, que é utilizado como um nome global no sistema. É através do *token* que Migrate mantém a consistência entre os números de seqüência do TCP em todos os nós do sistema. Já MobileIP baseia-se sobre uma estrutura de *daemons* agentes que servem para reencaminhar datagramas IPs para seus novos destinos quando um processo muda de uma máquina a outra. Entretanto, o MobileIP não resolve o problema de números de seqüência, o que dificulta sua aplicação em mecanismos de troca de mensagens baseados em TCP. Por fim, VNAT, que define uma infra-estrutura virtual de endereços IP e de portas criando uma rede IP virtual sobre uma rede IP real. Um serviço de controle baseado em *daemons* que executam em todas as máquinas do sistema mantêm uma tabela de conversão entre IP/porta virtuais e IP/Porta reais. A conexão é feita considerando-se IP/portas virtuais independentemente de onde o processo está sendo executado. A segunda estratégia possível é refazer as conexões TCP no momento da relançamento da aplicação, reinicializando as estruturas de controle (IP, porta, números de seqüência) adequadamente. Porém para que isso seja possível é necessário registrar o estado exato das conexões abertas e garantir que não haja mensagens em trânsito ou bufferizadas no sistema. A ferramenta TCPCP[14], (TCP *Connexion Passing*) é um exemplo de implementação dessa estratégia.

Na área de processamento paralelo, a maioria das aplicações são escritas utilizando MPI, que por sua vez é baseado em TCP. Ao se desejar empregar mecanismos de *checkpointing/restart* é necessário considerar que o MPI define suas próprias semânticas de troca de mensagens e as bufferiza de forma independente do sistema operacional, o que é um problema adicional a ser tratado. A implementação de LAM-MPI, a partir de sua versão 7.1, oferece suporte a *checkpoint* considerando esse problema. Essa versão do LAM é baseada no emprego de BLCR para realizar *checkpointing* e no TCPCP para tratar a questão de conexões TCP. Em relação ao MPICH, existe esforços para dotá-lo de suporte de *checkpointing*. Uma dessas iniciativas é projeto MPICH-V[17], desenvolvido no LRI (Laboratoire de Recherche en Informatique) da Universidade Paris-Sud, França. Atualmente MPICH-V oferece três versões que diferenciam-se entre si em relação a políticas e condições de contorno para se realizar *checkpoint* consistente.

3 Arquitetura ambiente IGGI

A arquitetura proposta para o ambiente IGGI é formada por dois componentes básicos: Compute Mode e escalonador OAR (http://oar.imag.fr). O Comp.uteMode é uma ferramenta comercializada pela *startup* francesa Icatis (http://icatis.com.fr), parceira no projeto RNTL-IGGI. ComputeMode monitora a utilização de um sistema, e ao detectar sua ociosidade, o reinicializa no sistema operacional linux. O procedimento de inicialização do linux é configurado de tal forma que faz com que a máquina se integre a um cluster virtual. O cluster virtual é composto por um conjunto de máquinas identificadas por uma faixa de endereçamento IP privativa, como por exemplo, 192.168.20.0/24, e por um servidor NFS. A máquina ociosa, quando reinicializada, recebe um IP nessa faixa de endereçamento e se torna um cliente NFS. No entanto, Compute Mode não oferece nenhum mecanismo de *checkpointing*. No caso do projeto IGGI, no BRGM, as máquinas que compõem o cluster virtual são disponibilizadas apenas na faixa de horário 20h00 às 6h00 e durante finais de semana, se tornando imprescindível salvar o processamento realizado nesse período e retomá-lo mais tarde.

O segundo componente é o escalonador *batch* OAR que foi desenvolvido no laboratório ID-IMAG com o objetivo de permitir que usuários submetam tarefas (*jobs*) paralelas ou seqüenciais em um conjunto de máquinas. O diferencial que OAR apresenta em relação a outros escalonadores *batch* para clusters, como por exemplo, PBS, é o fato de ter sido

projetado para trabalhar em ambientes de *grids*, o que lhe confere algumas características particulares. Por exemplo, como em ambientes de grids é comum a heterogeneidade de equipamentos, tanto em nível de software como de hardware, OAR prevê a possibilidade de agrupar máquinas em função de suas configurações e/ou distribuição geográfica. Para prover portabilidade, OAR foi desenvolvido em Perl e emprega uma base de dados MySQL para registrar os recursos disponíveis e agendamentos de utilização. Outra preocupação foi com escalabilidade já que OAR é o escalonador empregado no projeto francês Grid5000 [17] que agrupa cerca de 1000 nós de cálculo dispersos em várias regiões francesas (Paris, Nancy, Grenoble, Lyon, Lille, Rennes, Toulouse, Bordeaux e Nice).

A solução proposta no projeto IGGI é acoplar o funcionamento de ComputeMode com o escalonador OAR. O primeiro aspecto de interação é que ComputeMode oferece a possibilidade de no momento do *reboot* informar a configuração de hardware da máquina. O OAR, de posse dessa informação, pode tomar decisões de escalonamento de tarefas selecionando, por exemplo, máquinas que sejam homogêneas. Da mesma forma, é possível agrupar máquinas que pertencem a uma mesma subrede, ou tecnologia de interconexão, para evitar tráfego na rede ou velocidades diferentes entre diferentes nós de um mesmo cluster virtual. Uma vez o cluster virtual instanciado e a aplicação sendo executada, é necessário que antes do fim do período de utilização o processamento realizado seja salvo. Isso se traduz na necessidade de x minutos antes do final do período de utilização, por exemplo, início do expediente a 8h00 da manhã, seja escalonada uma tarefa que faça o *checkpoint* das aplicações em execução. Feito o *checkpoint*, o OAR repassa o controle a ComputeMode para que esse reinicialize a máquina no sistema operacional Windows para ser novamente utilizado por seu usuário habitual.

A interação entre o ComputeMode e OAR é toda baseada no envio e recepção de mensagens que acionam sinais UNIX. O estado atual de implementação da arquitetura IGGI é um protótipo que oferece o funcionamento descrito nos parágrafos anteriores.

4 Resultados experimentais

Como visto na seção 2, BLCR é um mecanismo de *checkpointing* implementado em nível de núcleo (kernel). Quando acionado, cria um arquivo binário com uma imagem do estado do processo. Sendo assim, o tempo de *checkpointing* é influenciado pelo tamanho do processo em sua memória virtual e pelo tempo de criação/escrita desse arquivo. Portanto, para os objetivos do projeto IGGI, é importante ter-se uma estimativa desse tempo.

Para avaliar o tempo de *checkpointing* procedeu-se uma série de experimentos com o objetivo de verificar a influência do tamanho do processo, uso de sistemas de arquivos remoto (NFS) e a relação com a capacidade de memória real (RAM) dos nós de cálculos. As experiências foram feitas usando-se aplicações reais do BRGM, seqüenciais e paralelas (MPI), e aplicações sintéticas para melhor controlar alguns parâmetros. Os resultados obtidos são discutidos nesta seção.

Plataforma experimental: As experiências foram conduzidas sobre a infraestrutura Grid5000[15] onde é possível alocar um número significativo de nós e configurá-los de acordo com necessidades próprias. Em todas as configurações, um dos nós era empregado como servidor NFS e os demais como seus clientes. Para uniformizar o hardware da máquina, todos os experimentos foram conduzidos no site Orsay (Paris) da Grid5000 que possuem a seguinte configuração: AMD Opteron 2 GHz, 2 GB de memória RAM, discos rígidos de 80 GB (IDE-7200 rpm) interconectados por um switch gigabit ethernet. Em relação ao software, cada nó foi configurado com um núcleo linux (versão 2.6.12.1) com LAM MPI versão 7.1.2b31, BLCR 0.4.2, gcc 4.0.2 e gfortran. A partição de *swap* é de 4 GB. É importante salientar que há uma forte dependência de versões dos softwares instalados devido as estruturas do núcleo e do próprio MPI que são consultadas no momento do *checkpoint*.

Metodologia : Todas as aplicações foram executadas normalmente e as operações necessárias para disparar o *checkpoint* e gerar os arquivos de rastros (tempo e tamanho do arquivo de *checkpoint*) foram feitas por um processo independente (*shell script*). Todas medidas foram realizadas de forma a validá-las estatisticamente através de cálculo de média (μ), desvio padrão ($\hat{\sigma}$) e intervalos de confiança. Os resultados apresentados são os valores médios obtidos.

4.1 Aplicações de benchmark

As aplicações utilizadas para a obtenção dos resultados apresentados nesta seção são, com exceção de uma (Mandelbrot), aplicações reais utilizadas quotidianamente no BRGM, a saber:

- **EDP:** aplicação seqüencial de mecânica de fluidos que executa cálculo de equações diferenciais parciais. Essa aplicação é desenvolvida em C e caracteriza-se pelo seu médio consumo de memória e por uma duração de alguns minutos. Esses valores são influenciados pelo tamanho do problema.

- **Phreeqc:** avalia através de simulações de escoamentos polifásicos, acopladas a simulações de processos geoquímicos, o nível de CO_2 geologicamente armazenado. Phreeqc é amplamente utilizada pela comunidade de geoquímica. Sua execução é caracterizada por um baixo consumo de memória (menor que 10 MB),

porém o tempo de processamento pode ser de vários dias. Phreeqc é um código sequencial escrito em C.

- **Mandelbrot:** implementa o cálculo de fractal de Mandelbrot sobre uma região de $n \times n$ pontos. É a única aplicação não utilizada no BRGM. Foi usada como aplicação sintética devido a facilidade de controlar o consumo de memória da aplicação através do aumento do número de pontos (n).

- **Ondes3D:** efetua a simulação da propagação de ondas em meios elásticos e viscosos-elásticos, sendo empregada na previsão de riscos sísmicos e geotérmicos. Essa aplicação é baseada no método de diferenças finitas e é escrito em C utilizando MPI. Sua execução é caracterizada por um grande consumo de recursos de memória e sua duração é função da granularidade definida para a malha, podendo variar desde dezenas de minutos a algumas horas ou mesmo dias.

- **Heat2D :** implementa o cálculo da equação de tranferência de calor em superfícies (bidimensional). É utilizada classicamente em problemas relacionados à poluição. Essa aplicação é um código paralelo MPI, escrito em C, baseado em diferenças finitas, e caracteriza-se por apresentar um consumo de memória uniforme entre os diferentes nós de cálculo.

4.2 Resultados: Aplicações seqüenciais

Nesta seção são avaliados a influência do uso de sistema de arquivos remoto (servidor NFS) e do espaço de memória ocupado por uma aplicação no tempo necessário para o *checkpointing*. Esses experimentos foram realizados com aplicações seqüenciais objetivando eliminar aspectos temporais relacionados com a comunicação por troca de mensagem. Os resultados são apresentados a seguir.

4.2.1 Influência do sistema de arquivos

O objetivo desse experimento é medir o impacto no tempo necessário para se efetuar o *checkpoint* ao se empregar um sistema de arquivos remoto em relação a um sistema de arquivos local. Para efetuar essa medida foram usadas duas aplicações seqüenciais: EDP e Phreeqc. A metodologia empregada consistiu em executar, várias vezes, cada uma dessas aplicações usando um sistema de arquivos local e, posteriormente, repetir sua execução sobre um sistema de arquivos remoto (*mount*) em um servidor NFS.

Com os parâmetros empregados para a execução da aplicação EDP, a mesma consumiu um espaço de endereçamento virtual de cerca 1.8 GB e gerou um arquivo de *checkpoint* em torno de 380 MB. Para o caso Phreeqc, o consumo memória foi de 9 MB e um arquivo de *checkpoint* de aproximadamente 3 MB. A partir das medidas realizadas, mostradas nas figuras 1 e 2, é possível constatar uma diferença entre 1.5 a 2 vezes entre o emprego de um sistema de arquivos local e um sistema de arquivos remoto (NFS). Essa diferença é explicada em função do atraso introduzido pela comunicação na rede e pelo processamento de requisições NFS, tanto no cliente, como no servidor.

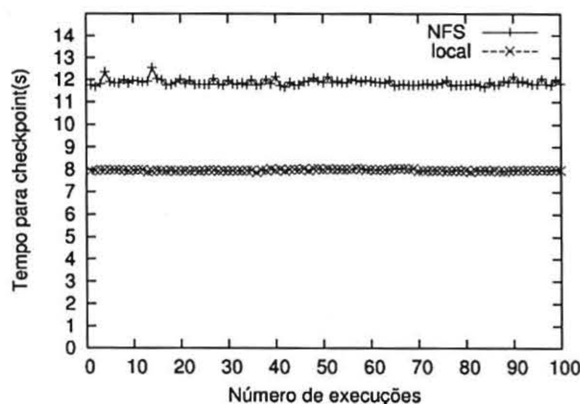

Figura 1. Aplicação EDP

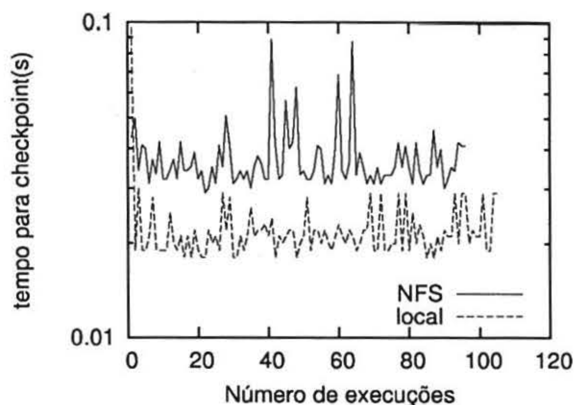

Figura 2. Aplicação Phreecq

4.2.2 Influência do tamanho do processo

Para dimensionar o impacto da quantidade de memória de uma aplicação sobre o tempo necessário para uma operação de *checkpoint* optou-se por utilizar uma aplicação em que fosse possível controlar de maneira precisa a quantidade de memória do processo. A aplicação escolhida foi a do fractal de Mandelbrot calculado sobre uma região de $n \times n$ pontos, variando-se n da seguinte forma: 512, 1024, 2048, 4096, 8192, 12228 (12 K) e 16384. A quantidade de memória equivalente para cada uma dessas configurações foi de cerca

de 17 MB, 25 MB, 48 MB, 510 MB, 1148 MB e 2050 MB, respectivamente. A figura 3 mostra o tempo necessário para o *checkpoint* para cada um desses casos.

A partir figura 3 observa-se que o tempo necessário ao *checkpoint* aumenta com o tamanho da aplicação, o que é esperado. Entretanto, o que é interessante de se notar é a proporção desse aumento. Até cerca de um quarto da capacidade de memória física instalada na máquina o aumento é linear. Quando o espaço de endereçamento virtual passa da metade da memória RAM disponível no sistema (2 GB), ou seja, 1 GB, o tempo para o *checkpoint* se torna bastante significativo. Esse aumento é explicado pelo sobrecusto introduzido pela gerência de memória (*swapping*). Esse comportamento do tempo de *checkpoint*, em função da relação entre memória virtual e memória RAM, foi constatado em outras máquinas com diferentes configurações de memória e também alterando-se as aplicações.

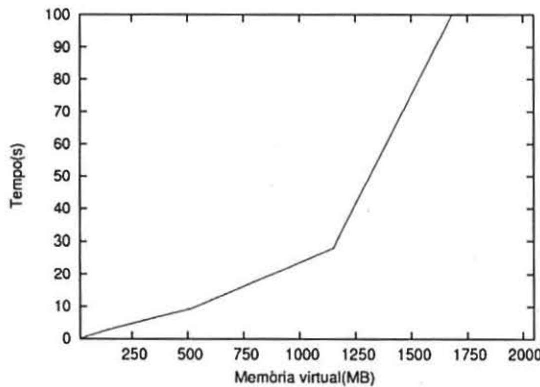

Figura 3. Relação entre tamanho do processo (memória virtual) e memória física (RAM)

4.3 Resultados : Aplicações paralelas

As aplicações paralelas MPI são caracterizadas pela existência de um certo número de processos distribuídos em diferentes nós de cálculo. Em relação as atividades de *checkpoint/restart* isso implica em salvar o estado de execução (imagem) do conjunto completo de processos. Na prática, empregando LAM MPI com suporte a BLCR, para uma aplicação lançada com n processos são criados $n+1$ arquivos de *checkpoint*: um para o processo *mpirun* e um para cada um dos processos MPI. Portanto, dois aspectos são interessantes de se avaliar: influência do gargalo de acesso ao servidor NFS em relação ao número de nós de cálculo (clientes NFS) empregados e o tamanho total e individual dos arquivos de *checkpoint*.

4.3.1 Influência do sistema de arquivos NFS

No contexto de uma intranet é importante quantificar o impacto da utilização de um servidor NFS na escrita de arquivos. Os experimentos realizados com aplicações seqüenciais mostraram um aumento de 1.5 a 2 vezes no tempo de *checkpoint*, se realizado em um sistema de arquivos remoto em comparação a um sistema de arquivos local. Entretanto, para aplicações paralelas surge um novo fator: a existência de vários nós clientes NFS. Portanto, é desejável medir a relação entre o número de nós e o tempo de *checkpoint*.

O primeiro experimento consistiu em obrigar que todos os nós escrevessem um arquivo de *checkpoint* de mesmo tamanho. Para que isso fosse possível foi necessário aumentar a dimensão do problema ao mesmo tempo que se aumentava o número de nós de cálculo. O objetivo era avaliar a carga que clientes com arquivos de mesmo tamanho produziam no servidor NFS. Para essa análise foi usada a aplicação paralela MPI Heat2D devido a sua homogeneidade no consumo de memória. Para se manter o tamanho constante do arquivo de *checkpoint* se dobrava a dimensão do problema ao mesmo tempo que se dobrava a quantidade de máquinas empregadas para o cálculo do problema. A figura 4 mostra os resultados obtidos para 2, 4, 8 e 16 nós.

Entretanto, na realidade, o primeiro experimento não é a situação comum pois, para a execução de uma mesma aplicação, a medida que se aumenta o número de nós, se diminui a quantidade de memória individualmente usada em cada nó. Para avaliar o impacto entre o número de clientes e o servidor NFS em condições mais reais, executou-se um novo experimento utilizando a aplicação sísmica Ondes3D variando-se, entre 1 e 100, a quantidade de processos MPI. Como cada processo MPI era instanciado em um nó de cálculo diferente, tem-se de 1 a 100 clientes NFS nesse segundo experimento. O resultado é dado na figura 5.

Na figura 5, caso local, nota-se que o tempo de *checkpointing* é influenciado por dois fatores: divisão do tempo de processamento em n processos e o tamanho do arquivo de *checkpoint* (quanto mais processos, menor o tamanho do arquivo). Essa relação explica o comportamento da curva da figura 5: havendo um só processo de cálculo tem-se um único arquivo sendo gravado por um único processo, com mais processos tem-se arquivos menores escritos por diferentes processos. Levando em conta que as máquinas empregadas no teste são biprocessadores, tem-se uma sobreposição de operações de escrita e de cálculo até um certo número de processos. Após esse número, há um gargalo na execução das operações de E/S e o tempo volta a subir. Isso explica o 'joelho' que aparece na curva entre 10 e 20 processos. Um outro ponto interessante a comentar é que, ao se dividir uma aplicação de x MB em n nós diferentes, o tamanho do somatório dos n arquivos é superior ao do caso em que há um só arquivo ($n = 1$).

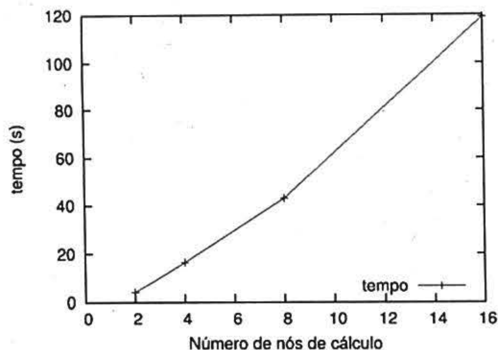

Figura 4. Heat2D:tempo de checkpointing

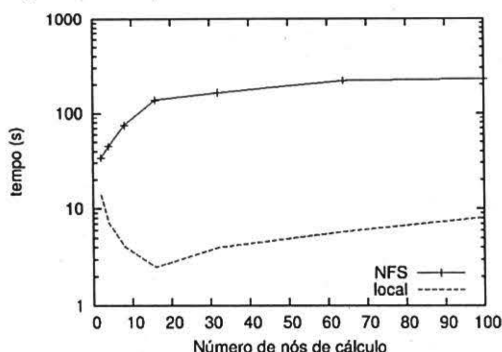

Figura 5. Ondes3D: sistema de arquivos local versus remoto (NFS)

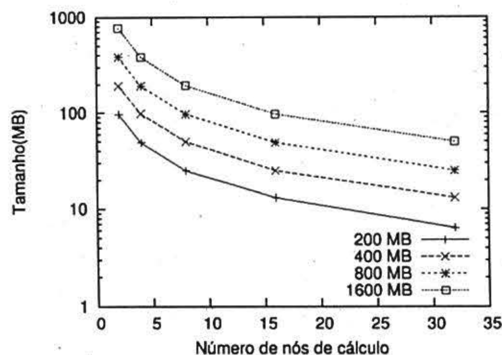

Figura 6. Heat2D:Arquivos de checkpoint

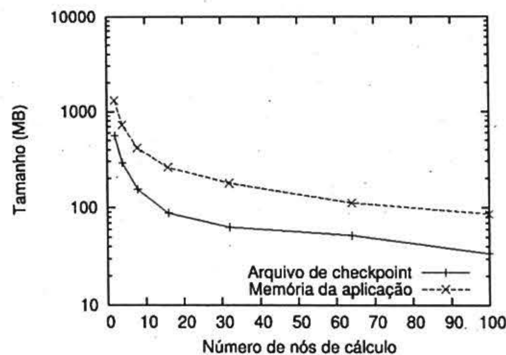

Figura 7. Onde3D:Arquivos de checkpoint

4.3.2 Influência do tamanho do processo

O objetivo é avaliar o impacto no tempo de *checkpointing* quando uma tarefa é dividida em n partes e cada uma delas alocada em um nó de cálculo diferente. Os experimentos realizados foram feitos considerando uma aplicação que apresentasse uma distribuição uniforme de trabalho e de consumo de memória entre os diferentes nós (Heat2D) e outra aplicação que não possuisse essa característica (Ondes3D).

A aplicação Heat2D implementa a solução da equação de transferência de calor em duas dimensões. Essa aplicação é interessante porque ela permite facilmente modificar a dimensão do problema mantendo uma relação direta com o consumo de memória. Os testes realizados empregaram quatro problemas de dimensões diferentes que ocupavam entre 200 MB a 1.6 GB de espaço de endereçamento em memória, e foram lançados em um número de nós variando entre 2 a 32. A figura 6 mostra, por nó, a evolução do tamanho dos arquivos de *checkpoint*. O mesmo tipo de procedimento foi executado com a aplicação Ondes-3D (risco sísmicos) onde não há uma distribuição uniforme do espaço total de endereçamento entre todos os nós de cálculo. O resultado desse último teste é apresentado na figura 7.

As figuras 6 e 7 mostram que existe um padrão de comportamento em relação ao tamanho do arquivo de *checkpoint* independente da dimensão do problema. Por exemplo, para a aplicação Heat2D ao dobrar a dimensão do problema se duplica também o tamanho do arquivo de *checkpoint*, mas o comportamento da curva se mantém o mesmo. Essa relação de duas vezes é justificada pelo fato que essa aplicação possui um comportamento bem regular em relação a demanda de consumo de memória. No que diz respeito a aplicação Ondes3D, irregular em relação ao consumo de memória, observa-se que o comportamento da curva acompanha aquele do tamanho do espaço de endereçamento. Conclui-se que, para esses casos, é possível estimar com razoável precisão o tempo necessário ao *checkpoint* para qualquer dimensão de problema.

4.4 Avaliação do tempo de retomada

O tempo de retomada (*restart*) de uma aplicação a partir de seu arquivo de *checkpoint* é díficil de ser avaliado de forma precisa devido ao funcionamento dos comandos disponibilizados pela BLCR (*cr_restart*) e pelo LAM MPI (*lamrestart*). Basicamente, quando esses dois comandos são executados, eles relançam a aplicação como um processo filho do *shell* em que foram executados. Se esses comandos são feitos empregando a opção de *background* (&) do

UNIX, mede-se o tempo de execução do comando e não o do lançamento da aplicação propriamente dita. Sem usar a opção de *background*, a aplicação só devolve o controle a seu processo pai (o processo *cr_restart* ou *lamrestart*) no final de sua execução, ou seja, o tempo medido inclui àquele da duração da aplicação e não apenas o de seu lançamento.

Considerando essas restrições, mediu-se o tempo de retomada a partir de observação e cronometragem manuais. Empregando esse método, obteve-se um tempo de retomada de cerca de 1 segundo para a aplicação seqüencial EDP (arquivo de *checkpoint* de cerca de 380 MB) e de 2 a 3 segundos para a aplicação paralela MPI Ondes3D (arquivo de *checkpoint* de cerca de 2 GB a 4GB de acordo com o número de nós de cálculo empregados). No entanto, salienta-se que, dentro do contexto do projeto IGGI, o tempo de retomada não é considerado como um fator fundamental. O mais importante é a estimativa de tempo de *checkpointing* para poder iniciá-lo em tempo hábil para devolver o recurso de cálculo a seu usuário normal.

5 Conclusão

As experiências realizadas mostraram que o tempo necessário para efetuar uma operação de *checkpoint* é fortemente influenciado por três fatores: (1) a relação entre memória física instalada na máquina (RAM),a memória virtual e o tamanho do processo; (2) o uso de sistema de arquivos remoto versus um sistema de arquivo local e; (3) o número de clientes NFS. Esse último aspecto é importante principalmente no contexto de aplicações paralelas, onde se tem um certo número de processos que são alocados a diferentes nós de cálculos, situação muito comum em ambientes de *cluster* e *grids*. Essas análises foram importantes para nortear os próximos passos dentro do projeto IGGI que trata da definição de clusters virtuais baseados em um sistema de arquivos remoto do tipo NFS. Uma das estratégias é a utilização de vários servidores NFS localizados em nível departamental na estrutura do BRGM. No entanto tal decisão limita o número de máquinas usadas em um cluster virtual. Como trabalho a ser investigado está o uso de um sistema de arquivos distribuído, como por exemplo, o NFSp, no lugar do NFS.

Além disso, a experiência deste trabalho forneceu subsídios para uma etapa em andamento que é o escalonamento dinâmico de processos MPI, onde o mecanismo de *checkpoint/restart* é usado com o intuito de balanceamento de carga. Nesse caso, ter-se uma previsão do tempo de *checkpointing* é importante para se tomar uma decisão sobre a vantagem, ou não, de se migrar processos MPI em tempo de execução.

Referências

[1] Germain,C et alli.; XtremWeb: an experimental platform for Global Computing. Grid2000, Dec.2000, IEEE Press.

[2] SETI. SetiHome Project. http://setiathome.berkeley.edu

[3] SPlank, J.S.; Beck, M.; Kingsley, G.; Li, K.. Libckp: Transparent Checkpoint Under UNIX. Conference Proceedings, Usenix Winter'95. Tech. Conference, pg.213-33, jan. 1995.

[4] Litzkow, M.; Tannenbaum, T.; Basney, J.; Livny, M.. Checkpoint and Migration of UNIX Process in the Condor Distributed System. http://www.cs.wisc.edu/condor/doc/ckpt97.ps

[5] Takahashi, T.; Sunimoto, S.; Hori, A.; Harada, H.; Ishikawa, Y.. PM2: Hight Performance Communication Middleware for Heterogeneous Network Envrionments. http://www.sc2000.org/techpapr/papers/pap.pap205.pdf

[6] Dieter, W.; Lumpp, J.. User-level Checkpointing for Linux Threads Programs. FREENIX Track. USENIX 2001 Annual Technical Conference. pp-81-92, june, 2001.

[7] Hendricks, E.. VMADump. http://bproc.sourceforge.net

[8] Zhong, H.; Nieh, J.; CRACK: Linux Checkpointing/Restart As a Kernel Module. TR CUCS-014-01. Department of Computer Science. Columbia University, November 2002.

[9] Duell, J., Hargrove, P., and Roman., E. The Design and Implementation of Berkeley Lab's Linux Checkpoint/Restart. Berkeley Lab Technical Report LBNL-54941.

[10] Cryopid.CryoPID - A Process Freezer for Linux. http://cryopid.berlios.de

[11] Snoeren, A.C; BALAKRISHNANH, H.; An End-to-end Approach to Host Mobility. In: Proc. of 6th Int. Conference on Mobile Computing and Networking, 2000, New York, NY, USA, ACM Press, p.155-166.

[12] Perkins, C; Mobile IP. Communications Magazine. IEEE, v. 40, n. 5, pg. 66-82, may, 2002.

[13] Su, G.; Nieh, J.; Mobile Communication with Virtual Network Address Translation. TR CUCS-003-02. Dept. of Computer Science, Columbia University, Feb. 2002.

[14] Almesberger, W..TCP Connection Passing.http://tcpcp.sourceforge.net

[15] Sankaran, S.; Squyres, J,; Barrett,B.; Lumsdaine, A.; Duell, J.; Paul Hargrove, and Eric Roman. The LAM/MPI Checkpoint/Restart Framework: System-Initiated Checkpointing. In LACSI Symposium, Oct. 2003.

[16] MPICH-V MPI implementation for volatile resources. http://www.lri.fr/bouteill/MPICH-V

[17] Cappelo, F.; et alli.; Grid'5000: A Large Scale, Reconfigurable, Controlable and Monitorable Grid Platform. Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing, Grid'2005. Seattle, Washington, USA.

[18] Plank, J.S.; An overview of checkpointing in uniprocessor and distributed systems focosing on implementation and performance. Technical Report UT-CS-97-372. Department of Computer Science. University of Tennessee, Knoxville.