Avaliação de um Mecanismo de Checkpointing para o MyGrid

Jeysonn Isaac Balbinot, Ingrid Jansch-Pôrto,
Hélio Miranda Silva, Taisy Silva Weber

Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{jibalbinot,ingrid,hamsilva,taisy}@inf.ufrgs.br

Abstract. *The heterogeneity, complexity and software and hardware diversity found in grids make the probability of failures higher than in traditional distributed systems. MyGrid front-end is a middleware that gives support for the execution of applications organized as Bag-of-Tasks in grid environments. A central node controls the application execution. To avoid the loss of computation in case of crash of this central node, which is a single-point-of failure, a rollback-recovery mechanism was proposed to work in conjoint with the scheduler. This paper uses SimGrid toolkit to evaluate, by simulation, the impact of this mechanism over the performance of the applications.*

Resumo. *A heterogeneidade, complexidade e diversidade de software e hardware dos grids tornam a probabilidade de defeitos maior que em sistemas distribuídos tradicionais. O MyGrid é um middleware que dá suporte à execução de aplicações do tipo Bag-of-Tasks em um grid. O controle das aplicações é feito por um nodo central que se torna ponto único de falha. Para evitar a perda de computação no caso de colapso deste, foi proposto um mecanismo de recuperação por retorno vinculado ao código do escalonador. Este trabalho avalia, via simulação, o impacto deste mecanismo no desempenho das aplicações, utilizando a ferramenta SimGrid.*

1. Introdução

O crescente avanço do *hardware* e *software*, bem como o aumento da velocidade e qualidade dos serviços de rede oferecidos atualmente, mostram-se como uma alternativa para a exploração do poder computacional. O uso de uma grande quantidade de máquinas interligadas, com fraco acoplamento, dá origem a um novo conceito, conhecido como computação em *grid*[1].

O *grid* constitui-se em uma plataforma para o compartilhamento coordenado de recursos distribuídos geograficamente. Possui como objetivo resolver uma gama de problemas e aplicações distribuídas nas mais diversas áreas, como ciência, engenharia e comércio, atendendo assim a organizações virtuais multi-institucionais e dinâmicas. Os serviços oferecidos pelo *grid*, e.g. o acesso sob demanda aos recursos, devem ter baixo custo, oferecendo um certo grau de consistência e confiabilidade [6][7].

[1] Embora o termo *grid* venha sendo traduzido por alguns pelo vocábulo *grade*, em português, optou-se por mantê-lo em sua versão original.

Devido às características intrínsecas de uma plataforma *grid* como, por exemplo: heterogeneidade de *hardware* e *software*, grande variedade de recursos espalhados em diversos domínios, compartilhamento do ambiente por várias aplicações e possibilidade de executar aplicações que podem utilizar até milhares de nodos, a probabilidade de ocorrência de falhas e defeitos torna-se muito alta [8], de maneira que a ocorrência de falhas pode ser vista quase como a regra, e não mais como a exceção.

O MyGrid [4][5][9] é uma ferramenta que auxilia no desenvolvimento de aplicações paralelas, com apoio ao gerenciamento de recursos e tarefas que compõem uma plataforma *grid*. O MyGrid provê suporte a execução de aplicações que sigam o modelo *Bag-of-Tasks* (BoT) – são tarefas independentes entre si e que podem ser escalonadas e executadas em qualquer ordem e em qualquer recurso que esteja disponível no *grid*. A total independência entre as tarefas faz com que não exista comunicação entre elas durante a computação [3].

Algumas soluções de escalonamento oferecidas pelo MyGrid utilizam replicação de tarefas como alternativa para o aumento de desempenho, sem a necessidade de informações adicionais sobre o *grid* ou sobre a aplicação. Além da busca de melhoria de desempenho com o uso de replicação, esta técnica também auxilia na tolerância a falhas. Desta forma, são tolerados defeitos em recursos do *grid* que estejam processando uma das réplicas. Atualmente, não existe outro mecanismo específico para tolerância a falhas. Defeitos que afetem o nodo central, o qual coordena a aplicação, acarretam a perda de todas as tarefas e resultados daquela fase da computação.

Considerando então a probabilidade real de falhas nestes sistemas, decidiu-se trabalhar no suporte às atividades do *grid*, melhorando a capacidade de resposta do sistema à ocorrência de defeitos. O mecanismo escolhido é o uso de *checkpoints* relacionados às atividades do escalonador de tarefas, visando a posterior recuperação de estado caso a máquina onde está sendo executado o escalonador sofra defeito. Porém, embora a técnica permita que a aplicação possa ser recuperada após a ocorrência de defeitos, ela não deve degradar o desempenho do sistema, de maneira significativa.

Dessa forma, é importante avaliar o impacto que o salvamento de estado causará sobre o desempenho normal do MyGrid. Decidiu-se fazer esta avaliação através de simulação, em função das vantagens: necessidade de recursos virtuais, controlabilidade do ambiente e facilidade de repetição dos experimentos.

Portanto, o objetivo deste trabalho é avaliar o custo da abordagem proposta – sobreposição do *checkpointing* à operação do escalonador do MyGrid – com intenção de melhorar a resposta a defeitos em arquiteturas em *grid*. A degradação do desempenho durante a operação normal (livre de falhas) de aplicações executadas no MyGrid é avaliada com o uso de simulações realizadas com o SimGrid [1][2][14].

Este artigo está organizado como segue. Na seção 2, são detalhados o funcionamento do MyGrid, sua estrutura operacional, políticas de escalonamento, e o modelo de *checkpointing* e recuperação propostos. Na seção 3, é descrita a ferramenta de auxílio para a simulação de *grids*, SimGrid. Na seção 4, é apresentado o modelo de simulação criado e um estudo de caso. Na seção 5, são apresentados os resultados obtidos com os experimentos. As conclusões finais estão na seção 6.

2. MyGrid

MyGrid [9] é uma ferramenta que visa dar suporte à utilização de recursos em um *grid* computacional e visa contornar dificuldades de gerenciamento e de escalonamento de tarefas neste ambiente. É um *software* de código aberto, implementado em Java e em

scripts shell. Pode ser utilizado tanto em plataformas Windows quanto Linux; a última versão (atualmente é 2.2) está disponível.

O *grid* de um determinado usuário é formado por todas as máquinas às quais ele tem permissão de acesso: isto é extremamente interessante para aplicações do tipo BoT, devido ao fraco acoplamento das tarefas. A execução simultânea de aplicações é coordenada pelo nodo central que escalona as tarefas nas máquinas que executam cada aplicação. O nodo central é chamado de *home machine* e os demais nodos são as máquinas do *grid* (*grid machines*)[2]. Cada máquina do *grid* executa uma tarefa de cada vez. Em uma configuração típica, essas máquinas não compartilham o sistema de arquivos, nem possuem os mesmos *softwares* instalados. Toda a comunicação é iniciada e coordenada pela *home machine* a qual, além de conter os dados de entrada de cada aplicação, é responsável por fazer a coleta dos resultados da computação.

O objetivo do MyGrid é oferecer um sistema simples, completo e seguro para execução de aplicações BoT em *grids*. O esforço para uso do MyGrid deve ser mínimo e o mais próximo possível de uma solução pronta (*plug-and-play*), por isso, simples. Completo identifica a necessidade de cobrir todo o ciclo de uso de um sistema computacional, do desenvolvimento à execução. A segurança é atingida pela proteção a vulnerabilidades no ambiente computacional do usuário [5].

Cada máquina do *grid* deve oferecer um conjunto mínimo de serviços, definidos pelo MyGrid, que compõem a *Grid Machine Interface* [4]. Estes serviços são: disparo (*start-up*) de tarefas em máquinas do *grid* (execução remota); cancelamento de tarefas que estejam em execução; transferência de arquivos das máquinas do *grid* para a *home machine*; transferência de arquivos da *home machine* para as máquinas do *grid.*

A *home machine* possui um escalonador (denominado *Scheduler*), o qual recebe do usuário a descrição das tarefas a executar, escolhe onde cada tarefa será executada, submete e monitora cada uma delas segundo sua estratégia de escalonamento.

2.1. Escalonamento de tarefas

Na prática, não é trivial obter informações precisas sobre o ambiente e a aplicação, principalmente em ambientes amplamente dispersos como os *grids* computacionais [12]. Características do *grid* tais como heterogeneidade dos recursos, dinâmica natural das máquinas, carga, largura de banda, latência e topologia da rede de comunicação, devem ser consideradas durante o escalonamento, principalmente quando a aplicação utiliza grande quantidade de dados [10]. Como alternativa à dificuldade de obtenção de informações dinâmicas sobre os recursos e sobre o tempo de execução da aplicação, existem escalonadores de aplicações BoT que atingem bom desempenho utilizando replicação. Porém, o uso de tal heurística implica um consumo extra de recursos [12].

Em sua versão atual, o MyGrid oferece como estratégias de escalonamento os algoritmos *Workqueue*, *Workqueue with Replication* e o *Storage Affinity*.

A estratégia *Workqueue* (WQ) não necessita de nenhuma informação ou previsão de desempenho para fazer o escalonamento das tarefas, utiliza apenas informações necessárias, como nomes das tarefas a serem escalonadas e processadores disponíveis[3]

---

[2] Ao longo deste artigo, a denominação *home machine*, usada no MyGrid, será mantida. As demais máquinas, entretanto, serão referenciadas pelo termo traduzido (máquinas do *grid*, máquina remota (para enfatizar a execução de atividades externamente à *home machine*) ou como máquinas, apenas).

[3] Do ponto de vista do MyGrid, processador disponível é aquele que está conectado ao *grid* e ainda não recebeu qualquer tarefa.

para executarem estas. As tarefas são escolhidas de forma arbitrária e escalonadas aos processadores disponíveis. Após a tarefa ter sido completada, o processo envia o resultado ao escalonador, ficando assim apto a receber uma nova tarefa [10].

Uma variação da estratégia WQ é a *Workqueue with Replication* (WQR): há replicação de tarefas, quando um processador se torna disponível e não há mais tarefas pendentes para serem colocadas em execução. Então o WQR replica tarefas que ainda estão em execução até um número máximo definido pelo usuário ou enquanto houver processadores disponíveis. Esta replicação permite usar o primeiro resultado obtido; a execução das demais tarefas (réplicas ou original) que ainda não haviam produzido resultados é interrompida. Esta abordagem melhora o desempenho, se comparada ao WQ, pois a replicação aumenta a probabilidade de que pelo menos uma delas seja atribuída a um processador rápido ou com baixa carga de trabalho. Uma tarefa não replicada poderia ser inconvenientemente atribuída a um processador lento ou sobrecarregado por outras atividades e retardar a obtenção do resultado do *job* [10].

A estratégia *Storage Affinity* visa melhorar o desempenho de aplicações que processam grandes quantidades de dados (*Processors of Huge Data*, PHD) [13]. Para tanto, é usado um método de priorização de tarefas baseado em afinidade de armazenamento (*storage affinity*). O valor da afinidade entre uma tarefa e um site (*host*) ou máquina do *grid* determina quão próximo deste a tarefa está, ou seja, ela está associada à quantidade de *bytes* da entrada da tarefa que já está armazenada remotamente em uma determinada máquina do *grid*. Além de aproveitar a reutilização dos dados, o *Storage Afinitty* também efetua a mesma abordagem de replicação de tarefas utilizada no WQR [12].

2.2. Modelo de recuperação

Falhas que afetam isoladamente as máquinas do *grid* são menos críticas do que as que afetam a *home machine*, porque resultam na perda de apenas uma tarefa que esteja sendo executada nesse momento e não de todo o *job*. Além disso, esse problema pode ser parcialmente contornado com o uso de replicação de tarefas ou minimizado se os usuários dimensionarem tarefas de granulosidade reduzida. A introdução de recuperação vinculada às tarefas isoladas foi descartada porque sua implementação transparente (do ponto de vista dos programadores de aplicações) é bem mais complexa.

Na estrutura do MyGrid, a *home machine* é o elemento crítico na composição do sistema (“ponto único de falha”) pois, se uma falha a afetar, as aplicações dependentes dela poderão ser interrompidas ou mesmo perdidas. Além disso, é necessário lembrar que, em uma estrutura de *grid*, haverá várias máquinas atuando como *home*, uma para cada um dos diferentes usuários. Com o objetivo de melhorar a tolerância a falhas que afetem a *home machine* do MyGrid e, conseqüentemente o sistema como um todo, optou-se pelo uso do salvamento de *checkpoints* para uma posterior recuperação por retorno (*rollback-recovery*) [11].

Os *checkpoints* contêm informações que refletem o estado do sistema e processos envolvidos e, após a constatação do defeito, permitem retomar a execução a partir de um estado consistente prévio à ocorrência da falha. Com este mecanismo, a execução anterior na *home machine* pode ser recuperada e continuar a execução de um *job* no mesmo nodo ou em outro nodo disponível, evitando a repetição de execuções anteriores concluídas (cujos resultados já foram recebidos) e o relançamento de tarefas em andamento. Com as informações referentes à execução de tarefas já escalonadas, o procedimento de recuperação pode tentar a obtenção dos resultados destas, refazendo

conexões com as máquinas. O insucesso desse procedimento resulta no re-escalonamento destas, em máquinas disponíveis. Assim, a perda máxima pós-falha corresponderia a reiniciar todas as tarefas que estavam em andamento e realizar as demais ainda não escalonadas. O melhor cenário de recuperação pós-falha é aquele que permite utilizar todos os resultados já obtidos das tarefas prontas e retomar o andamento daquelas tarefas que estavam em execução.

O *checkpointing*, ou seja, o salvamento dos *checkpoints* é definido por dois tipos de condições (eventos), os quais correspondem às mudanças de estados mais significativas durante a operação de escalonamento. A primeira condição está relacionada ao escalonamento de uma nova tarefa: corresponde ao início da execução, após a tarefa ser transferida para uma máquina do *grid* com sucesso. Seu objetivo é o de manter atualizado o mapa das tarefas distribuídas em máquinas do *grid*. A segunda condição ocorre quando a *home machine* recebe o resultado de uma tarefa, ao final de sua execução. Este resultado pode ser: a) correto e com a resposta produzida; ou b) um indicativo de erro.

Na primeira condição, as mudanças afetam somente a estrutura de dados. Mas, na segunda, a tarefa que completa com sucesso retorna os resultados produzidos, através de transferência de dados para a *home machine*. Estes resultados devem ser armazenados como parte do *checkpoint* para poderem ser reutilizados caso haja falha na *home machine*. A estrutura que armazena as informações referentes ao *checkpoint* possui tamanho variável, dependente do número de tarefas do *job*, da quantidade de tarefas concluídas e dos arquivos de saída. Em conseqüência a esse tamanho, a largura de banda e a latência da rede vão influenciar no tempo de transferência do *checkpoint*, além das variáveis de tempo para salvar tais estruturas em armazenamento estável ou em uma máquina remota. Em caso de receber a indicação de erro, a tarefa deverá ser removida da tabela de tarefas em execução e re-escalonada.

3. SimGrid

A ferramenta desenvolvida para auxiliar na simulação de *grids*, SimGrid [1][2][14], oferece controle de baixo nível sobre os processos e já foi utilizada pelos desenvolvedores do MyGrid para avaliação do comportamento de diferentes estratégias propostas para o escalonador [12], razão pela qual foi também escolhida para este trabalho. SimGrid é um pacote escrito em linguagem C que provê funções básicas e abstrações para a simulação de aplicações em ambientes distribuídos heterogêneos, oferecendo suporte ao estudo de algoritmos de escalonamento e políticas de gerência de recursos em *grids* computacionais.

SimGrid permite a manipulação de tarefas e recursos. Na configuração, as tarefas dividem-se em tarefas de computação ou de transferência de dados, as quais são escalonadas nos recursos de processamento (CPU) ou canal de comunicação, respectivamente. Os recursos CPU e canal de comunicação são tratados como recursos não relacionados; esta associação fica sob responsabilidade do usuário, para atender aos requisitos específicos da simulação em estudo. Assegurar a correspondência adequada entre tarefas de computação / processadores e tarefas de transferência de dados / canais de comunicação para efeito de escalonamento também ficam a cargo do usuário.

Os recursos são modelados com duas características de desempenho: latência (tempo em segundos para acessar o recurso) e taxa de serviço (número de unidades de trabalho executadas por unidade de tempo).

Duas estruturas de dados são oferecidas pelo SimGrid: `SG_Resource()` para gerenciar recursos e `SG_Task()` para gerenciar as tarefas. Um recurso é descrito por um nome, um conjunto de métricas de desempenho e *traces* ou valores constantes de configuração. Por exemplo, um processador é descrito pela medida de sua velocidade (relativa a um processador de referência) e um *trace* sobre a sua disponibilidade, isto é, a porcentagem de CPU que será alocada para um novo processo por um determinado tempo. Um canal de comunicação também é descrito por um nome, um custo e um estado. No caso de tarefas de transferência de dados, o custo é definido pelo tamanho dos dados (em bytes); tarefas de computação têm seu custo expresso pelo tempo (em segundos) de processamento exigido no processador de referência. Finalmente, o estado é usado para descrever o ciclo de vida das tarefas (não escalonada, escalonada, pronta, rodando e completa) [2].

Além disso, a API (*Application Programming Interface*) do SimGrid provê funções básicas para operar os recursos e as tarefas (criação, destruição, inspeção, etc.), funções para descrever as possíveis dependências entre tarefas, e funções para escalonar e remover (desescalonar) tarefas escalonadas previamente nos recursos.

A função `SG_Simulate()` é usada para executar a simulação, conduzir as tarefas através de seus ciclos de vida e simular o uso dos recursos. A simulação pode ser executada durante uma quantidade (virtual) pré-estabelecida de segundos, ou até que uma dada tarefa, qualquer uma, ou todas as tarefas sejam completadas. Em todos os casos, o `SG_Simulate()` retorna uma lista das tarefas que tenham sido completadas desde o último tempo em que foram escalonadas. A função `SG_getClock()` retorna o tempo global virtual da simulação; pode ser chamada a qualquer tempo durante a simulação [2].

A atual API do SimGrid (v2) possui duas interfaces: SG e MSG. A primeira (SG), usada neste trabalho, é a API original; é a de mais baixo nível, sendo bastante flexível e geral. Nesta, a simulação explicita o escalonamento de tarefas em recursos. Já a MSG é construída sobre a SG. Esta camada implementa simulações realísticas baseadas na SG e é mais orientada à aplicação, onde a simulação é construída em termos de comunicação de agentes [1].

4. Modelo de Simulação

O modelo de simulação é composto por três partes: o *job* (aplicação), o ambiente (*grid*) e o escalonador de tarefas.

O *job* é uma lista de tarefas que compõem a aplicação. Cada tarefa possui um custo computacional, o qual corresponde ao tempo necessário para ser executada no processador de referência dedicado. Um processador pode executar (processar) tarefas em diferentes velocidades, mas um processador de referência tem velocidade igual a 1. Cada tarefa tem seu tamanho associado, que representa os dados a serem transferidos pelo canal de comunicação para o recurso onde ela foi escalonada e será executada.

O ambiente (*grid*) é um grupo de sites (*hosts*), ou conjunto de máquinas, compostos por um ou mais processadores, cada um com sua velocidade de operação, que podem a executar tarefas de computação. Além disso, cada processador tem como configuração a sua disponibilidade, cujo valor pode variar entre 0 e 1.

O escalonador é o processo responsável pela distribuição das tarefas através dos processadores disponíveis no *grid*. A estratégia de escalonamento executada pelo escalonador simulado é a WQ (*Workqueue*), descrita na seção 2.1.

Uma característica importante na criação de simulação é a possibilidade de associar dependências entre tarefas. No SimGrid, a associação entre duas tarefas é feita através da função `SG_addDependency()`.

Na simulação desenvolvida, para cada tarefa computacional (*computation task*) é também criada implicitamente uma tarefa de transferência (*transfer task*) baseada no tamanho do arquivo a ser transferido. Esta tarefa de transferência é escalonada no recurso canal de comunicação, antes de iniciar sua tarefa de computação correspondente. Assim, uma tarefa de computação é dependente de uma tarefa de transferência: a tarefa de computação somente irá iniciar sua execução após a tarefa de transferência correspondente ter sido completada. Tais abstrações de dependência visam representar mais adequadamente a execução de uma aplicação em um ambiente real.

4.1. Estudo de caso

O principal objetivo das simulações foi avaliar o impacto no desempenho (tempo total de execução) ocasionado pelas ações referentes ao *checkpointing* no MyGrid. Assim, foram criados dois tipos de aplicações, uma composta por um conjunto de tarefas com custos heterogêneos; e a outra com custos homogêneos. O detalhamento destas será apresentado na seção 5.

Como as simulações foram feitas usando a estratégia de escalonamento *Workqueue*, as tarefas são escalonadas nos primeiros processadores disponíveis. No modelo adotado, o salvamento de estado é efetuado na fase de escalonamento de cada tarefa e ao término destas. Assim, para representar o custo associado aos *checkpoints*, no início da simulação são implicitamente criadas duas tarefas (representando as diferentes fases) que são associadas a cada tarefa da aplicação. Nas tarefas inseridas associadas ao final da execução da tarefa da aplicação, além do custo normal referente às estruturas do *checkpoint*, é associado também o custo relativo a transferência e salvamento dos resultados calculados ou produzidos pela tarefa da aplicação. Essa nova tarefa possui estrutura e funcionamento análogos a uma tarefa normal da aplicação.

O salvamento de estados pode ser feito na máquina local (*home machine*) ou em uma máquina remota, e estas duas opções foram consideradas na simulação. Na prática, o primeiro caso é eficaz apenas para falhas temporárias, enquanto o segundo é adequado para falhas que afetem a *home machine* de forma permanente. No caso de simulação com salvamento na *home machine*, ou seja, na mesma máquina onde o escalonador está sendo executado, a tarefa que representa o *checkpoint* é representada por uma *computation task*, adicionada de um custo computacional que representa o salvamento de estados. Não há necessidade de explicitar o tamanho e o custo de transferência porque a tarefa é executada na máquina local. Assim, todos os recursos podem ser acessados por um único canal de comunicação compartilhado.

Nesse estudo, a *home machine* não executa tarefas da aplicação, mas apenas de escalonamento. Com a introdução do *checkpointing*, a *home machine* fica também encarregada do salvamento destas informações.

Para o caso de simulação da perda de dados e retomada em caso de colapso da *home machine*, os resultados das tarefas devem ser salvos em armazenamento estável, aqui representado por uma máquina remota (já que não é esta que foi atingida pelo defeito). Neste caso, o tempo gasto para a transferência também é computado nas simulações. Dessa forma a tarefa que representa o *checkpoint* é composta por duas partes – uma *transfer task* e uma *computation task* – como uma tarefa normal da aplicação. Nesse caso, a *home machine* executa as funções do escalonador e transfere as estruturas de

checkpoint e as respostas das tarefas para serem salvas na máquina remota. Assim ocorre o compartilhamento do canal de comunicação entre a máquina remota e a *home machine*; logo, essa disputa pelo recurso é refletida no resultado final da simulação. A máquina remota criada no início da simulação é exclusiva para a execução de tarefas que representam o *checkpoint*: ela não pertence ao conjunto de máquinas que executam tarefas da aplicação, como pode ser visto no exemplo de execução mostrado na Figura 1 a). Na prática, esta máquina pode abrigar um sistema de armazenamento robusto.

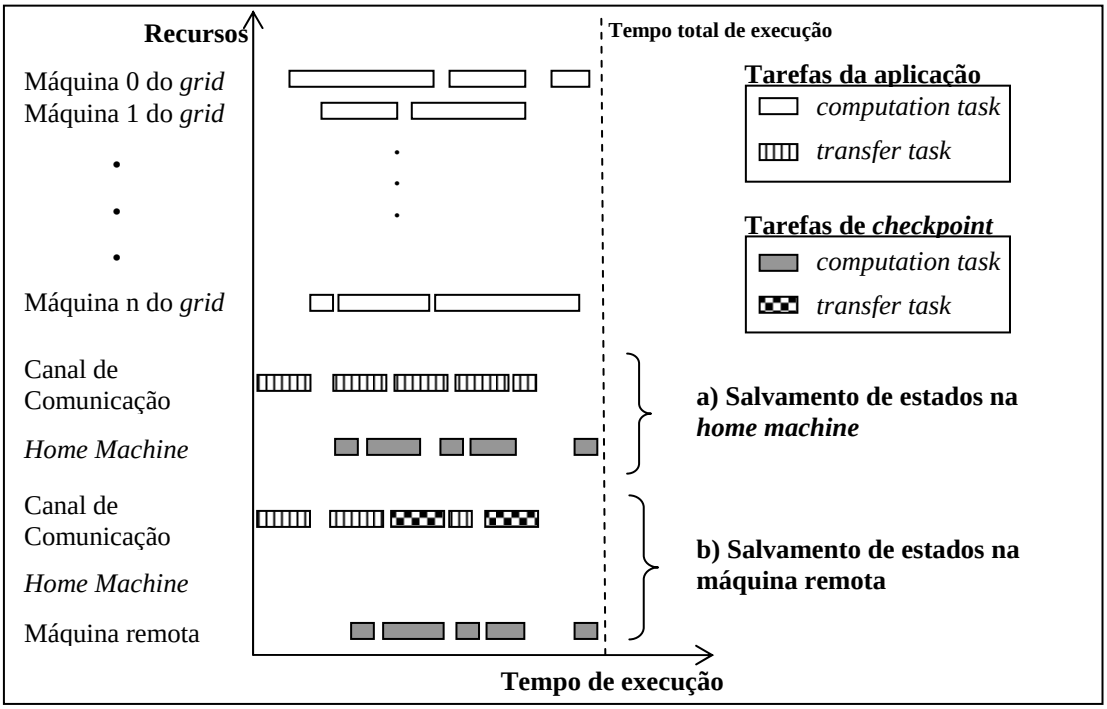

Figura 1. Execução de uma aplicação com *checkpoint*

A saída da simulação corresponde ao tempo total de execução da aplicação. Nos experimentos efetuados com a execução do *checkpointing*, este tempo reflete a sobrecarga devido aos *checkpoints*, porém mostra que os totais dos tempos de execução do *checkpointing* não se somam simplesmente ao tempo de execução das aplicações isoladas (sem *checkpointing*). Isso ocorre porque as tarefas de transferência e computação dos *checkpoints* – expressas através do canal de comunicação e dos ciclos de CPU – são executadas de forma concorrente com outras tarefas da aplicação como pode ser observado na Figura 1 b), mas seguem a ordem de execução segundo suas dependências. O uso do simulador auxilia na percepção deste comportamento referente aos recursos consumidos.

O impacto dos procedimentos de *checkpointing* sobre o escalonador é mais significativo do que sobre as aplicações, já que ele está envolvido em toda coordenação do processo e no controle da execução de suas ações. Entretanto, ao usuário do *grid* vai transparecer o tempo de obtenção dos resultados de suas aplicações. Por isso, este foi o parâmetro de observação escolhido.

5. Resultados Experimentais

O ambiente de simulação criado para executar os experimentos é formado por uma *home machine* e nove máquinas monoprocessadas das quais uma, particularizada como

máquina remota, armazena o *checkpoint*. Todas têm capacidade de processamento igual a 1. O *job* da aplicação é composto por um conjunto de tarefas com tamanho fixo em 100K *bytes*, mas cuja quantidade varia exponencialmente de 4 até 512.

Cada tarefa da aplicação possui, além de sua configuração, o custo computacional (expresso em segundos) e o tamanho (em *bytes*) associados, relativos às estruturas dos *checkpoints* que devem ser transferidas e salvas. Tais valores, utilizados para configurar o simulador, foram estabelecidos a partir de medidas efetuadas sobre uma implementação preliminar do mecanismo de *checkpointing* no MyGrid; eles podem ser observados na Tabela 1. O método usado baseou-se na criação de *jobs* com diferentes quantidades de tarefas, que eram executados no MyGrid. O *job* foi composto por tarefas que calculam a seqüência de Fibonacci até o número 35. Além dos tempos (custo do *checkpoint*), foram observados também as características das estruturas de dados relacionadas. Particularidades das tarefas que compõem o *job* (tamanho e complexidade) não influenciam no dimensionamento da estrutura do *checkpoint*; ele armazena informações de interesse do escalonamento e das respostas recebidas. Estas últimas dependem das características das aplicações.

Tabela 1. Parâmetros de *checkpoint* medidos em execuções reais

Nº de Tarefas	Tamanho (bytes)	Tempo de transferência (ms)
4	2249	81
8	3593	99
16	6287	105
32	11663	118
64	22409	184
128	43919	284
256	86927	445
512	172943	771

Com objetivo de simplificar os experimentos, a simulação idealiza as máquinas e a rede de comunicação (usada para transferência dos dados): todas as máquinas têm disponibilidade de 100%, a rede tem largura de banda de 10Mbits/s e latência zero.

O tempo total de execução da aplicação, definido como sendo o intervalo de tempo entre o momento em que a primeira tarefa do *job* é iniciada e o primeiro instante em que todas as tarefas finalizaram suas execuções, foi a métrica de desempenho adotada neste trabalho. Dentro deste tempo também é contabilizado o tempo gasto na transferência de tarefas e estruturas do *checkpointing*, quando este é efetuado.

Uma intuição que embasou os experimentos é a de que os procedimentos de *checkpoint* teriam maior impacto sobre um conjunto de tarefas homogêneas do que sobre as heterogêneas. A quase sincronização do término de muitas tarefas causaria uma rajada de *checkpoints*. Assim, foram feitas simulações comparativas com estes dois cenários.

5.1. Tarefas Heterogêneas

Neste experimento, foram executados 8 *jobs*, cada um composto por 4, 8, 16, 32, 64, 128, 246 e 512 tarefas respectivamente, onde o custo de cada tarefa dentro de cada *job* segue uma distribuição pseudo-aleatória uniforme no intervalo de 60 a 3600 segundos. Atendendo a essas configurações, dois cenários foram simulados: num cada tarefa produzia arquivos de saída com tamanho de 500Kb e, noutro, este tamanho era de 100Kb. Esses dois cenários foram comparados com um terceiro, onde a execução das

tarefas era feita sem a geração de *checkpoints*. Os tempos de transferência e tamanhos de estruturas de *checkpoint* efetuados na fase de escalonamento de uma determinada tarefa seguem a Tabela 1, enquanto que os *checkpoints* efetuados ao final da execução desta tarefa têm acrescidos aos valores da Tabela 1 o tamanho da resposta e do tempo para transferência desta. Esses tempos também foram obtidos pelas médias dos tempos de transferências reais na rede de 10Mbits/s: para 100Kb obteve-se o tempo de 314 ms e para 500Kb, 1317 ms. Vale ressaltar que estes valores variam em função do tráfego da rede. Os tempos totais de simulação para cada *job*, nas diferentes configurações, estão representados no gráfico da Figura 2.

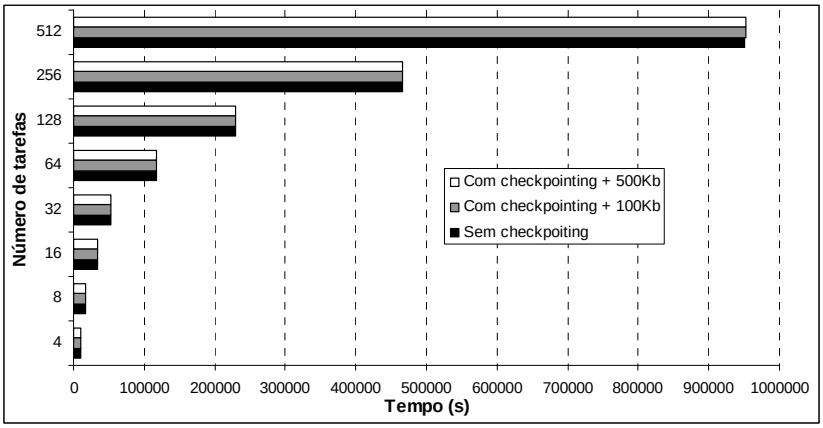

Figura 2. Gráfico com os tempos das tarefas heterogêneas (60-3600s)

Pode-se notar que a degradação de desempenho ou o acréscimo no tempo total da simulação, causado pela inclusão do mecanismo de *checkpointing*, foi baixo, variando de 0,02 a 0,12% para tarefas que produzem arquivos de 100Kb, e 0,06 a 0,19% para tarefas que produzem arquivos de 500Kb.

5.2. Tarefas Homogêneas

O segundo tipo de experimento segue as mesmas configurações do primeiro, porém todas as tarefas são homogêneas e possuem custo fixo de 120 segundos. Esses experimentos não estão associados necessariamente a cenários em que as tarefas são idênticas e as máquinas homogêneas, mas podem estar associadas a cenários heterogêneos nos quais os resultados sejam recebidos em rajadas (muitas tarefas retornam resultados quase simultaneamente). Os tempos totais de simulação em segundos para cada *job*, nas diferentes configurações, podem ser observados no gráfico da Figura 3.

Os resultados, análogos aos conjuntos de tarefas heterogêneas, apresentam baixa degradação de desempenho quando do uso de *checkpointing*, variando entre os percentuais de 0,46 a 1,83 para tarefas que produzem arquivos de 100Kb, e de 1,56 a 2,92% para tarefas que produzem arquivos de 500Kb.

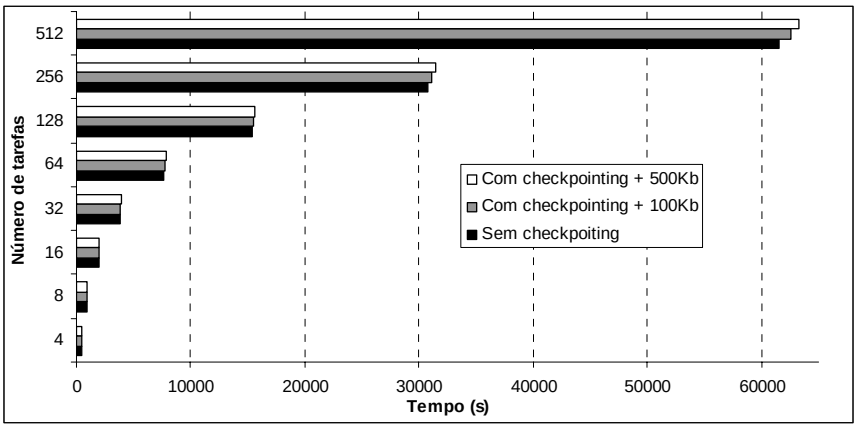

Figura 3. Gráfico com os tempos das tarefas homogêneas (120s)

6. Conclusão

Neste trabalho, considera-se a probabilidade real de falhas em ambientes de *grids*, e propõe-se o uso de *checkpoints* para aumentar a tolerância a falhas do sistema. Estes *checkpoints*, vinculados ao escalonador do *grid*, registram informações referentes à distribuição, localização e resultados das tarefas em execução. Em caso de falha, o escalonador pode ser reativado, na máquina original ou em outra máquina do *grid*, reduzindo significativamente a perda de computação já realizada.

O estudo aqui realizado visou avaliar o desempenho da técnica em diferentes configurações de aplicações possíveis no *grid*. O ambiente que serviu para a modelagem foi o MyGrid e a ferramenta empregada para as simulações foi o SimGrid.

O uso de ferramentas de simulação, tal como o SimGrid, é uma boa alternativa para compreender o comportamento de escalonadores de tarefas, além de ser útil para análise do impacto de técnicas como o *checkpointing*. Mostrou-se bastante flexível, mas apresentou um certo grau de complexidade na implementação das simulações.

Foram simulados cenários com topologia fixa e variações no número e custo das tarefas. Os testes revelam o custo adicionado ao tempo original das aplicações, tanto para o conjunto de tarefas homogêneas, quanto heterogêneas. Pode-se notar que os resultados causados pela inclusão do *checkpointing* no escalonador na execução de conjuntos de tarefas heterogêneas e homogêneas mostraram-se semelhantes, causando baixa degradação no desempenho. O percentual máximo de 2,92% de aumento no tempo total de execução de um conjunto de tarefas (BoT) é aceitável no uso da técnica.

Além disso, vale lembrar: a) que não há necessidade do usuário introduzir qualquer tipo de alteração na programação das aplicações – portanto a técnica fica transparente aos usuários do *grid*; b) o tempo de recuperação pós-falhas deverá ser bastante reduzido, pois se restringe a reiniciar o escalonador, carregando as estruturas de dados anteriormente salvas. Resultados mais completos poderão ser obtidos observado-se outras características da aplicação, tais como: a granulosidade das tarefas, sua quantidade, o custo e o tamanho das respostas produzidas.

Agradecimentos

Este trabalho é parcialmente financiado pela HP Brasil P&D, a quem os autores agradecem.

Referências

[1] Casanova, H.; Legrand, A.; Marchal, L. **Scheduling Distributed Applications: the SimGrid Simulation Framework**. Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03), 2003.

[2] Casanova, H. **Simgrid: A Toolkit for the Simulation of Application Scheduling**. Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid'01), p. 430-437, May 2001.

[3] Cirne, W.; Paranhos, D.; Costa, L.; Santos-Neto, E.; Brasileiro, F.; Sauvé, J.; Silva, F. A. B.; Barros, C. O.; Silveira, C. **Running Bag-of-Tasks Applications on Computational Grids: The MyGrid Approach**. Proceedings of the ICCP'2003 - International Conference on Parallel Processing - October 2003.

[4] Cirne, W.; Brasileiro, F.; Sauvé, J.; Andrade, N.; Paranhos, D.; Santos-Neto, E.; Medeiros, R. **Grid computing for bag of tasks applications**. Proceedings of 3rd IFIP Conference on E-Commerce, E-Business and E-Government - Sept. 2003.

[5] Cirne, W. **Grids Computacionais: Arquiteturas, Tecnologias e Aplicações**. III ERAD - Escola Regional de Alto Desempenho, Santa Maria, RS. Janeiro 2003.

[6] Foster, I. **What is the Grid? A Three Point Checklist**. GRID today online magazine, July 2002.

[7] Foster, I.; Kesselman, C.; Tuecke, S. **The anatomy of the Grid: Enabling scalable virtual organizations**. LNCS, 2150:1. 2001.

[8] Medeiros, R.; Cirne, W.; Brasileiro, F.; Sauvé, J. **Faults in Grids: Why are they so bad and What can be done about it?**. Grid Computing, 2003. Proceedings. Fourth International Workshop, p. 18-24, Nov. 17, 2003.

[9] MyGrid Homepage: **MyGrid Online Manual**. In: http://www.ourgrid.org/mygrid.

[10] Paranhos, D.; Cirne, W.; Brasileiro, F. **Trading Cycles Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids**. Proceedings of the Euro-Par 2003: International Conference on Parallel and Distributed Computing. August, 2003.

[11] Pradhan, D. K. Fault-Tolerant Computer System Design. Upper Saddle River: Prentice Hall, 1996.

[12] Santos-Neto, E. **Escalonamento de Aplicações que Processam Grandes Quantidades de Dados em Grids Computacionais**. 2004. 71 f. Dissertação (Mestrado em Informática) – CCT – UFCG, Campina Grande.

[13] Santos-Neto, E.; Cirne, W.; Brasilero, F.; Lima, A. **Exploiting Replication and Data Reuse to Efficiently Schedule Data-Intensive Applications on Grids**. Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing, June 2004.

[14] **SimGrid Homepage**. In: http://gcl.ucsd.edu/simgrid.