Implementação de um Mecanismo de Recuperação por Retorno para o Ambiente de Computação OurGrid

Hélio Antônio Miranda da Silva, Tórgan Flores de Siqueira, Leonardo Rech Dalpiaz, Ingrid Jansch-Pôrto, 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

{hamsilva, torgan, lrdalpiaz, ingrid, taisy}@inf.ufrgs.br

Abstract

OurGrid is a middleware that supports the execution of applications on resources available in a grid. A central node performs the scheduling and control of applications, which makes it a critical element in the system in case of faults. Aiming to decrease the loss of computation caused by crash failures of the central node, we propose an implementation of a rollback recovery mechanism. The overhead of this mechanism on the OurGrid scheduler is assessed in scenarios with various characteristics.

Resumo

O OurGrid é um middleware que oferece suporte à execução de aplicações nos recursos disponíveis em um grid. Nesta ferramenta, o controle e o escalonamento das aplicações são feitos por um nó central, o que o torna um elemento crítico do sistema, em situação de falha. Visando diminuir a perda de computação causada por colapso no nó central do OurGrid, este trabalho apresenta uma implementação de mecanismo de checkpointing e respectiva recuperação. O impacto do mecanismo no desempenho do escalonador é avaliado em cenários com características distintas.

1. Introdução

A computação em *grid* (ou computação em grade) tem emergido como uma área de pesquisa importante por permitir o compartilhamento de recursos computacionais geograficamente distribuídos entre vários usuários [Foster; Kesselman, 2003]. Contudo, a heterogeneidade e a dinâmica do comportamento dos recursos em ambientes de *grid* torna complexo desenvolver e executar aplicações [Bosilca *et al*., 2002]. Existem várias plataformas de *grid* que tentam contornar estas dificuldades, e dentre elas pode-se citar: *Globus* [Foster; Kesselman, 1998], *Legion* [Grimsham, 1997] e OurGrid [Andrade *et al*., 2003]. Estes são exemplos de sistemas que oferecem suporte para a execução e gerência de aplicações paralelas e distribuídas em ambientes de *grid*.

O OurGrid, além de aspectos relativos à execução de aplicações distribuídas em ambientes de computação em *grid*, oferece e gerencia um esquema de troca de favores entre usuários. Neste esquema, instituições (ou usuários) que possuam recursos ociosos em um dado momento podem oferecê-los a outros que deles necessitem. Quanto mais um domínio oferecer recursos ao *grid*, mais será favorecido quando precisar, ou seja, terá prioridade mais alta quando requisitar máquinas ao *grid*. O *software* MyGrid [Costa

et al., 2004] é parte integrante do OurGrid. É através dele que o usuário interage com o *grid*, submetendo e gerenciando suas aplicações. Assim, as funcionalidades incorporadas ao MyGrid repercutirão sobre as execuções do OurGrid.

No modelo de execução do MyGrid, as tarefas são lançadas pelo nó central, chamado de **máquina base**, que coordena todo o escalonamento. As máquinas que executam as tarefas são denominadas **máquinas do grid** (GuMs). A máquina base apresenta um comportamento caracterizado na literatura como "ponto único de defeitos", pois seu colapso faz com que todos os resultados do processamento corrente sejam perdidos. Isto pode significar horas ou até mesmo dias de processamento perdido, dependendo das aplicações. Visando suprir esta deficiência, este trabalho descreve o funcionamento e a implementação de um mecanismo de *checkpointing* (ou salvamento de estado), usado como base para a recuperação por retorno, que permite ao sistema voltar a um estado consistente, com perda parcial de dados, após um defeito da máquina base. A viabilidade do mecanismo de recuperação foi estudada, de forma preliminar, em um trabalho anterior [Balbinot *et al*., 2005]. Os resultados do presente trabalho, entretanto, não são diretamente comparáveis em termos numéricos, pois o código da simulação implica em diversas simplificações e estimativas – que se modificaram na implementação real.

A recuperação por retorno é uma técnica utilizada para a obtenção de tolerância a falhas em sistemas computacionais. Ela se baseia no salvamento de estados de execução de uma aplicação. O conteúdo deste salvamento é chamado *checkpoint*, e corresponde ao conjunto de informações sobre o estado de execução de uma aplicação que é suficiente para restabelecer, consistentemente, o fluxo de processamento após a ocorrência de uma falha e detecção correspondente [Plank, 1997]. Esta técnica foi incorporada à máquina base do MyGrid, modificando-se o código-fonte. Com isto, ela salva, em armazenamento estável, o estado (estruturas de dados e informações de controle imprescindíveis) capaz de restaurar o sistema após uma falha, oferecendo uma alternativa à sua característica de ponto único de defeito. Os *checkpoints* são obtidos e salvos a cada mudança de estado do escalonador do nó central; essas mudanças ocorrem ao ser lançada uma nova tarefa e quando uma tarefa retorna um resultado (execução bem-sucedida).

O texto está organizado como segue. Na seção 2, são descritas as características do OurGrid e é detalhado o funcionamento do MyGrid, sua estrutura operacional e política de escalonamento. Na seção 3, é apresentado o modelo de funcionamento e a implementação do mecanismo de *checkpointing,* bem como as ações para recuperar a máquina base em caso de falhas. Na seção 4, é avaliado o impacto do mecanismo sobre o desempenho, através da execução de aplicações em diferentes cenários, explorando diferentes características. Por fim, na seção 5, são apresentadas algumas conclusões.

2. Estrutura do OurGrid

O OurGrid é uma plataforma que oferece suporte à execução de aplicações *bag-of-tasks* (BoT) em *grids* computacionais. Ele abstrai o gerenciamento de recursos e o escalonamento de tarefas, escondendo do usuário a complexidade intrínseca dos *grids*, como a heterogeneidade e a dinâmica de recursos, por exemplo. Ele também é responsável pela alocação de recursos, desonerando o usuário, que pode se concentrar exclusivamente na aplicação. A plataforma é de código aberto (disponibilizada sob a licença GPL), desenvolvida em Java e *scripts shell*. A comunicação entre processos é feita via RMI (*Remote Method Invocation*).

O OurGrid é composto basicamente por três entidades: o MyGrid, os *Peers* e os *UserAgents*. O MyGrid é o principal componente, sendo responsável pelo escalonamento de tarefas e pela gerência dos recursos. Ele também é a interface do sistema: através dele ocorre toda a interação entre o usuário e o *grid* (especificação da topologia, submissão e acompanhamento da execução de tarefas) [OurGrid, 2005]. O *Peer* organiza e disponibiliza o acesso às máquinas do *grid*, denominadas GuMs, que compõem um domínio administrativo. Este domínio pode ser uma rede local, um *cluster* ou qualquer conjunto de recursos computacionais compartilháveis (para processamento e/ou armazenamento). O conjunto de vários *peers* forma uma comunidade, na qual cada *peer* doa uma determinada quantidade de recursos. A partir disso, é formada a *rede de favores*, na qual os recursos são compartilhados de acordo com a prioridade de cada *peer*. A prioridade é determinada através de créditos que um domínio obtém em função dos recursos que cedeu ao *grid*. Domínios que contribuíram mais são priorizados quando requisitarem recursos do *grid* [Cirne, 2002]. O *UserAgent* é a entidade localizada nas máquinas do *grid*, tornando-as aptas a participar do *grid*. O *UserAgent* fornece um conjunto de serviços como: a execução e o cancelamento de tarefas submetidas pela máquina base; recebimento de arquivos vindos da máquina base e a transferência de arquivos para a máquina base.

2.1. Funcionamento do MyGrid

No modelo de computação *bag-of-tasks* existe um conjunto de tarefas cuja execução é completamente independente, implicando a inexistência de comunicação entre elas. Além disso, não existe relação de ordem entre as tarefas [Cirne *et al.*, 2004].

A figura 1 ilustra a relação entre a máquina base, que controla a execução das tarefas, e as máquinas do *grid*, que efetivamente as executam. A máquina base armazena os programas que serão executados remotamente, os dados de entrada das tarefas e recolhe os resultados das tarefas computadas. A máquina base, assim como as do *grid*, não tem requisitos especiais, pode ser o *desktop* do usuário. Ela recebe do usuário a descrição das tarefas a executar, escolhe qual processador usar para cada uma, submete e monitora cada uma delas segundo a estratégia de escalonamento definida.

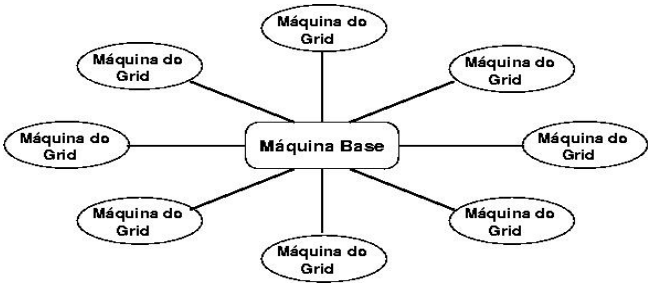

Figura 1: Estrutura do MyGrid

Executar uma tarefa envolve as seguintes etapas: *i)* transferir os arquivos necessários da máquina base para a máquina que executará a tarefa; *ii)* executar o(s) programa(s) transferido(s); *iii)* transferir os arquivos de saída, caso tenham sido gerados, da máquina do *grid* para a máquina base.

O MyGrid oferece duas estratégias de escalonamento de tarefas: *Workqueue* com Replicação [Paranhos; Cirne; Brasileiro, 2003] e o *Storage Affinity* [Santos-Neto *et al*., 2004]. A primeira é recomendada para aplicações que realizam computação intensiva, enquanto a segunda é mais adequada para aplicações que processam grandes

volumes de dados.

2.2. Aplicações como *Jobs*

Uma aplicação é composta de tarefas (*tasks*) relacionadas (mas idempotentes e sem relação de ordem), que o MyGrid agrupa na forma de *jobs*, e cuja descrição é feita em um arquivo de descrição de *jobs* (*job description file*). Para executar as tarefas de um *job*, é criado um conjunto de entidades, chamadas de réplicas. Para cada tarefa será criado um número de réplicas, definido pelo usuário, e elas são criadas em duas situações: quando uma réplica falha, uma nova réplica pode ser criada para tentar concluir a execução, ou pelo mecanismo de criação de réplicas visando aumento de desempenho. Neste segundo caso, o MyGrid gerencia a execução de múltiplas réplicas da mesma tarefa em diferentes máquinas.

Uma tarefa do MyGrid possui três fases: a *inicial*, a *remota* e a *final*. Na inicial, são transferidos para a máquina do *grid* os programas que realizarão a tarefa e, se existirem, os arquivos de entrada necessários. Na fase remota, a tarefa é executada; e na final, a máquina base busca os arquivos resultantes da execução. As fases inicial e remota são executadas por todas as réplicas da tarefa, mas somente uma terá permissão para executar a fase final. Quando a réplica que obteve permissão para executar a fase final concluí-la, as outras serão abortadas. Entretanto, se esta réplica falhar, outra obterá a permissão para concluir a fase final.

Os *jobs*, as tarefas e as réplicas, passam por vários estados durante seus ciclos de vida. O estado de um *job* depende do estado das tarefas que o compõem, que por sua vez dependem do estado das suas réplicas. Quando um *job* é submetido ao escalonador, ele entra no estado ***ready*****,** pronto para ter suas tarefas escalonadas. As tarefas também entram no estado *ready*, pois já foram criadas mas não tiveram a sua execução iniciada. Para que uma tarefa tenha sua execução iniciada, é preciso que sejam criadas e escalonadas as suas réplicas. Se um *job* possui pelo menos uma tarefa sendo executada, ele está no estado ***running***. Já uma tarefa é dita *running* se possui no mínimo uma das réplicas em execução. Se um *job* ou uma tarefa concluírem sua execução com sucesso, eles entram no estado ***finished***. O estado ***failed*** caracteriza uma falha durante a execução de uma ou mais tarefas. O estado ***canceled*** ocorre, tanto para um *job* quanto para uma tarefa, quando a execução é interrompida. As transições de estado das réplicas são idênticas às das tarefas, exceto por não entrarem no estado *ready*.

2.3. Visão geral dos componentes do MyGrid

O MyGrid é composto por um conjunto de componentes, que interagem permitindo à máquina base gerenciar a execução dos *jobs* submetidos. Os principais componentes são:

- ***Scheduler***: é o escalonador propriamente dito do MyGrid. Ele escalona as tarefas para que sejam executadas nas máquinas do *grid*.
- ***Replica Executor***: cuida da execução e gerência das réplicas nas GuMs.
- ***GuMProvider (GuMP)***: fornece máquinas ociosas para o *Scheduler*, quando este as requisitar. Cada GuMP gerencia um conjunto de máquinas do *grid*, definindo quais estão disponíveis e a forma de acessá-las.

A figura 2 ilustra, numa visão de alto nível, o funcionamento do MyGrid. Após criado, o *Scheduler* aguarda que *jobs* sejam submetidos pelo usuário. Ao receber um

job, o *Scheduler* determina quantas máquinas serão necessárias e as requisita aos *GuMProviders*. A requisição é feita para todos os *GuMProviders* conhecidos pelo *Scheduler*. A quantidade de máquinas requisitadas varia de acordo com o número de tarefas do *job*, o número de réplicas por tarefa e a heurística de escalonamento.

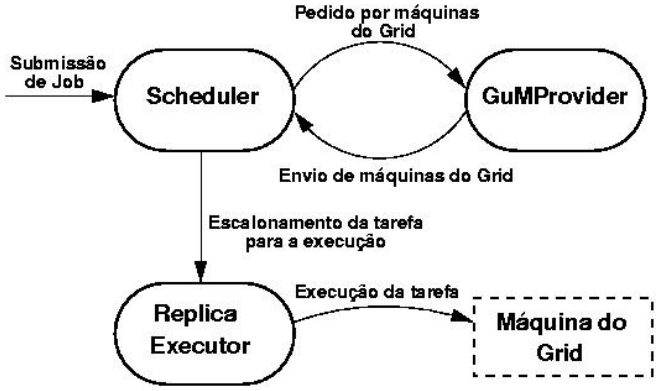

Figura 2: Visão geral dos componentes do MyGrid

Cada *GuMProvider* é responsável por um conjunto de GuMs, e ao receber um pedido por uma determinada quantidade de máquinas, ele tenta disponibilizar exatamente este número de máquinas, ou, se não for possível, somente as máquinas disponíveis. Portanto, as GuMs são entregues ao *Scheduler* à medida em que se tornam disponíveis, até que o número necessário de máquinas seja alcançado. Para cada GuM recebida, o *Scheduler* cria uma réplica da tarefa escolhida para ser executada. Depois de criada, a réplica e o nome da GuM são entregues ao *Replica Executor*, para que seja efetivamente executada.

3. Implementação do Sistema de Checkpointing e Recuperação

Utilizar técnicas de tolerância a falhas é fundamental para fazer com que o sistema complete sua computação mesmo na presença destas. Adicionalmente, os mecanismos empregados não devem degradar significativamente o desempenho global do sistema.

Na atual implementação do MyGrid, a máquina base centraliza o gerenciamento e o escalonamento das tarefas, e a ocorrência de falhas nela resulta na perda dos resultados de todas as tarefas que estavam sendo executadas no *grid* sob seu controle, pois ocorre a perda de toda computação relativa às concluídas (*finished*) e às que estavam em andamento (*running*). Neste caso, é necessário re-executar todas as aplicações que estavam em andamento no *grid*. As falhas que ocorrem em máquinas do *grid* não são tão críticas quanto às da máquina base, pois naquelas somente a tarefa que estava em execução é perdida. Além disso, o efeito dessas falhas pode ser contornado através de um novo escalonamento da tarefa, causando atrasos que podem variar de acordo com a computação perdida.

A solução, portanto, consiste em fazer com que a máquina base possa ter seu estado recuperado após uma falha, reduzindo seu nível de essencialidade ao mesmo de qualquer outra máquina do *grid*, tornando-a substituível, à custa de degradação do tempo total de execução. Para tanto, é necessário criar uma nova instância desta e iniciar o escalonador com o último estado consistente salvo antes da ocorrência da falha. Adicionalmente, é necessário recuperar as informações relativas à topologia do *grid*. Restaurando-se o estado do escalonador, é possível retomar a execução das aplicações que estavam em execução, sem perdas. A solução desenvolvida baseia-se em

recuperação por retorno, pois são realizados *checkpoints* da máquina base contendo um conjunto de informações necessárias para a recuperação do estado do escalonador.

3.1. Modelo de Falhas

O modelo de falhas adotado é o de **colapso** [Birman, 1996], no qual, quando uma falha ocorre em um computador ou processo, simplesmente é interrompida a execução, sem realizar computações ou ações incorretas.

No MyGrid, caso uma máquina se torne inatingível, devido a um colapso (parada) ou devido a falhas na rede de comunicação, ela deixa de receber tarefas e passa a ser considerada fora do *grid*. Contudo, há uma diferenciação entre falhas transientes e falhas permanentes. Caso ocorra uma falha na execução de uma tarefa, e a máquina que a estava executando continuar ativa, então esta máquina não é excluída do sistema, porque a falha pode ter sido gerada por problemas no ambiente de execução ou na configuração da tarefa. Assim, outras tarefas poderão ser lançadas neste nó e ter as suas execuções concluídas com sucesso, já que provavelmente o problema não vá se repetir.

3.2. Salvamento de estados

A implementação do mecanismo de salvamento e de recuperação de estado é feita modificando-se o código do MyGrid. São inseridos procedimentos para que a própria máquina base realize o salvamento em armazenamento estável das informações pertinentes à retomada de execução.

O escalonador gerencia uma lista de objetos do tipo `Job`, que representam os *jobs* que são sendo executados no *grid*. Cada instância do tipo `Job` possui, entre outros atributos, uma lista de objetos da classe `Task`, cada um representando uma das tarefas que compõem o *job*. As informações sobre as tarefas incluem as de estado e as de controle de execução, como a linha de comando que aciona a execução da tarefa os arquivos de entrada e de retorno. Cada tarefa contém uma lista com as suas réplicas que foram criadas. Portanto, o conteúdo de um *checkpoint* consiste na lista dos `Jobs` submetidos ao escalonador e na identificação dos diretórios de trabalho criados pelo MyGrid (individualizados para cada réplica). O **arquivo de descrição do *grid*** (*Grid Description File - GDF*), que contém a topologia do *grid*, é armazenado juntamente com o *checkpoint*, mas não faz parte dele. Esta cópia é criada (ou atualizada) quando o usuário submete um GDF ao MyGrid. Este mecanismo não faz parte do *checkpointing*, é apenas uma modificação do comando de submissão de GDF. Os arquivos de resultados também não são parte do *checkpoint*, embora sejam armazenados no mesmo diretório, e são referenciados separadamente, quando for o caso.

No *checkpointing*, o módulo de recuperação requisita ao *Scheduler*, através de chamadas RMI (*Remote Method Invocation*), as estruturas que contêm informações sobre o estado da execução das aplicações. A seguir, serializa os objetos que representam os *checkpoints* e os salva em disco.

Armazenar o *checkpoint* no sistema de arquivos local implica em uma cobertura somente de falhas transitórias. Para cobrir falhas permanentes, o armazenamento deve ser feito em outra máquina do *grid*, ou em um sistema de arquivos distribuído ou de rede, por exemplo, o NFS (*Network File System*). Esta funcionalidade é oferecida através do arquivo de configuração. Neste caso, uma falha permanente na máquina base ainda permite a recuperação em outra máquina pertencente ao *grid*.

A política de salvamento determina que o mecanismo de salvamento de estado

deve ser acionado sob duas condições (eventos), que correspondem às mudanças de estados mais significativas durante a operação de escalonamento. A primeira delas se refere ao escalonamento de uma nova tarefa no *grid*, e a outra ocorre quando há o retorno de uma tarefa, indicando que esta concluiu sua execução com sucesso.

Na figura 3 podem ser observados os momentos exatos em que é acionada a realização dos *checkpoints*. Esta figura descreve a seqüência de ações desde o escalonamento da réplica de uma tarefa até a sua execução remota e posterior finalização. O *Scheduler*, após determinar em qual GuM irá executar a réplica, entrega uma referência da réplica e da GuM selecionada para que o *Replica Executor* inicie a execução. Este componente cria e gerencia os fluxos de execução (*threads*) das réplicas, já que para cada uma é necessário um fluxo ativo. Cada um destes fluxos é uma instância da classe `MGReplicaExecutorThread`, e para cada nova réplica recebida, é criada uma nova instância desta classe. Esta classe efetua todas as fases da execução de réplicas, através da interação direta com o *UserAgent* das GuMs.

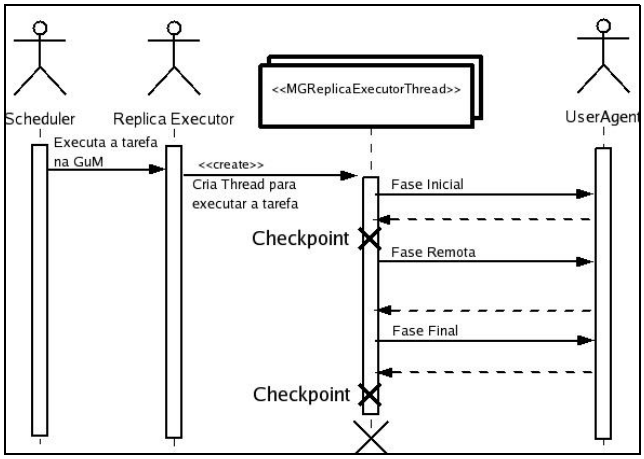

Figura 3*:* Diagrama de seqüência do lançamento de uma tarefa

O primeiro *checkpoint* feito durante a execução da réplica é realizado após o término da fase inicial. O segundo *checkpoint* é efetuado após o término das fases remota e final da tarefa. Contudo, a tarefa somente é registrada como concluída (*finished*) após o término da operação de *checkpoint.* Os *checkpoints* são realizados de forma atômica e sincronizada em relação às demais *threads* do MyGrid. Como o escalonamento das tarefas ocorre de forma concorrente, o mecanismo de *checkpointing* precisa obter acesso exclusivo às estruturas de dados do escalonador para evitar a criação de *checkpoints* inconsistentes com o estado do mesmo. O impacto deste acesso exclusivo é muito pequeno, pois as estruturas estão em memória durante a execução e são relativamente compactas (poucos quilobytes). Nos testes realizados, este tempo de acesso representou variação negligenciável sobre os valores medidos sem sua execução; assim, ele é absorvido pelos tempos das demais atividades do código de *checkpointing*.

Nos salvamentos de estados disparados pelo retorno de uma tarefa, que são aqueles feitos após a conclusão fase final da execução da réplica, é necessário que, além do *checkpoint*, os arquivos de retorno (resultantes da execução da tarefa) sejam salvos. Desta forma, estes arquivos estarão disponíveis caso haja a necessidade de realizar recuperação de estado.

3.3. Recuperação após falha

A recuperação da máquina base em conseqüência de falhas consiste em restaurar o

estado da máquina base a partir do *checkpoint*, incluindo a topologia do *grid*. Para isto, a máquina base inicia em modo de operação `restore`, no qual o mecanismo de recuperação é o responsável pela inicialização do MyGrid. A restauração da máquina base pode ser feita no mesmo computador ou em qualquer outro do *grid*, e o novo hospedeiro precisa ter acesso ao último *checkpoint* realizado. No modo de operação `restore` é criada uma nova instância da máquina base. Antes desta máquina estar apta à submissão de *jobs* é preciso restabelecer a topologia do *grid* e restaurar a execução das aplicações que estavam sendo executadas, a partir do último *checkpoint* realizado antes da falha. Após estas operações, ela prossegue seu funcionamento normal.

O *grid* é redefinido através da reconstrução da topologia com a qual a máquina base trabalhava antes da falha. Isto é feito carregando-se a cópia do arquivo de descrição do *grid*. A próxima etapa da recuperação refere-se à interpretação do *checkpoint*.

No mecanismo de recuperação, os *checkpoints* são interpretados de forma a evitar a perda de tempo causada pela repetição das computações já realizadas. Na recuperação de um *job*, o tratamento dado às tarefas varia de acordo com seu estado. Para as tarefas com estado *finished*, *failed* ou *canceled*, é necessário somente incluí-las no sistema, pois suas computações já foram realizadas e os seus resultados obtidos. Para as tarefas *ready*, que ainda não foram executadas, é necessário incluí-las na fila de escalonamento. Já para as tarefas *running*, que estavam em execução no momento da falha, o tratamento, um pouco mais complexo, é explicado a seguir.

Primeiramente, para cada réplica *running* é criado um fluxo de execução, para tentar restabelecer o contato com a GuM que executava a réplica. Obtendo sucesso no contato com a GuM (o que evita a perda da computação realizada), este fluxo de execução pergunta ao *UserAgent* dela qual é o estado da execução da réplica. O objetivo é saber se a execução ainda está em andamento ou se já foi concluída. Caso tenha sido concluída, é realizada a transferência dos arquivos de retorno e a tarefa é registrada como terminada. Caso contrário, a pergunta é repetida, de tempos em tempos, até que a réplica termine sua execução. Para que o *UserAgent* pudesse responder estas requisições, foi preciso modificá-lo adequadamente.

No processo de recuperação, se não for possível restaurar alguma das tarefas que estavam em execução, ela tem seu estado redefinido como *ready*, fazendo com que seja novamente escalonada. Portanto, o fato de não conseguir recuperar uma tarefa não leva o sistema a um estado inconsistente. Isto compatibiliza o MyGrid com *UserAgents* que não possuem as modificações necessárias para responder ao mecanismo de recuperação, pois, ao não obter sucesso nesta interação, o mecanismo faz com que esta tarefa volte ao estado *ready* e seja novamente escalonada.

4. Análise dos Resultados

A análise dos resultados envolve duas etapas: validar o modelo e avaliar o impacto sobre o desempenho do MyGrid; testes de eficácia e eficiência, respectivamente. O modelo foi validado através de um conjunto de testes que exercita o mecanismo de *checkpointing* e o de recuperação. *Jobs* são criados com um determinado número de tarefas, elaboradas de modo a permitir a injeção de falhas na máquina base e a avaliação do seu comportamento em várias etapas do ciclo de vida de uma tarefa. Para os dois primeiros casos de teste, dois tipos de tarefas foram submetidas: uma com processamento nulo e outra com tempo de processamento fixo. No primeiro caso, a máquina do *grid* não executava nenhum programa, e no segundo, executava um *sleep()*.

Entretanto, deve-se observar que o tempo de processamento da máquina do *grid* é irrelevante para os dois primeiros casos de teste, o importante é o tamanho do arquivo de resultados. O que é avaliado neste caso é a correção do algoritmo na presença de diversas possibilidades de falhas. Por diversas possibilidades entende-se provocar o colapso da máquina base tendo ela tarefas nos diversos estados: *ready*, *running*, *finished*, *canceled* e *failed*. Nos testes, o MyGrid (modificado) conseguiu sempre salvar e recuperar o último estado consistente do escalonador, reconstruir o estado da máquina e continuar a execução das tarefas.

A avaliação do impacto sobre o desempenho do MyGrid foi conduzida segundo duas métricas: o custo de realizar o *checkpointing* e o atraso percebido no tempo total de execução. O tempo de recuperação também poderia sofrer uma análise mais detalhada, mas percebeu-se que ele é negligenciável comparado ao tempo de reinicialização do MyGrid, que pode incluir a substituição da máquina.

No modelo de *checkpointing* implementado, o tamanho em *bytes* de cada *checkpoint* e o tempo gasto na sua realização é proporcional ao número *jobs* e ao número de tarefas. Quanto maior o número de tarefas que compõem os *jobs*, maior o número de objetos a serem salvos. A quantidade de processamento e a quantidade de memória gastos na execução de cada tarefa não têm influência no tamanho, pois o tamanho do *checkpoint* independe da aplicação. Contudo, os arquivos de retorno das tarefas influenciam no tempo de *checkpointing*, visto que também devem ser armazenados. Tendo em vista este modelo, o número de repetições de cada experimento foi determinado de modo que houvesse, no mínimo, 100 salvamentos de *checkpoints*. Isto significa que o número de repetições é dependente do número de tarefas de cada *job*. Ao final, são calculadas as médias aritméticas dos tempos de salvamentos.

Para os dois primeiros cenários, o *grid* consta de uma máquina base e uma GuM. A simplicidade dos cenários visa facilitar a interpretação dos resultados, sem prejuízo da validade dos testes. O código do MyGrid foi instrumentado nas classes responsáveis pelo salvamento. O fator de maior impacto no tempo de *checkpointing* é o número de tarefas, decorrente das estruturas de dados que são salvas. Portanto, a abscissa dos gráficos é uma escala logarítmica na base 2 do número de tarefas, variando de 8 a 1024. As máquinas estão ligadas via rede local, portanto, o atraso na comunicação é mínimo.

No primeiro cenário é medido o tempo de *checkpointing* para tarefas com arquivos de retorno de 100 KB, 500 KB e 1 MB. O tempo de referência é o de conclusão de tarefas sem o uso de *checkpointing*. As demais curvas de tempo correspondem, portanto, ao impacto de cada tipo de tarefa em relação ao tamanho do arquivo de resultados. Pelo gráfico da figura 4, pode-se ver que o armazenamento dos arquivos de retorno tende a influenciar no tempo total de *checkpointing*, e, conforme o tamanho destes aumenta, o tempo de salvar arquivos de resultados pode superar o próprio tempo de salvamento do arquivo de *checkpoint*. Isto corrobora a expectativa, pois as estruturas de dados a serem salvas são da ordem de poucos quilobytes.

No segundo cenário é avaliado o tempo de execução de tarefas com e sem a utilização do mecanismo de *checkpointing*. São utilizados *jobs* com quantidade de tarefas variando de 8 a 512, porém o tempo de referência é a completa execução de tarefas sem usar o mecanismo, para *tasks* com diferentes tamanhos de arquivos de retorno. Portanto, as demais curvas correspondem ao impacto causado pelo mecanismo em relação à operação normal do MyGrid. Os arquivos de retorno possuem tamanhos de 100 KB e 1 MB. O gráfico da figura 5 ilustra o impacto do *checkpointing* no tempo de execução das tarefas. Nas execuções do experimento, as tarefas realizaram um

processamento remoto de 30, 60 e 90 segundos. Nas execuções em que o tempo gasto na fase remota das tarefas é de 30s, o impacto do *checkpointing* sobre o tempo de conclusão varia entre 1,367% e 4,22%. Já para os casos em que o tempo de execução remota é de 90s, o impacto do mecanismo de *checkpointing* varia entre 0,46% e 1,48%. Sendo assim, o impacto do *checkpointing* mostrou-se aceitável, principalmente quando o tempo de execução das tarefas é o dominante, porque os tempos de *checkpointing* permanecem iguais e independentes do tempo de processamento remoto das tarefas. Deve-se destacar que, em aplicações reais, este processamento tende a ser preponderante no tempo total de execução da tarefa.

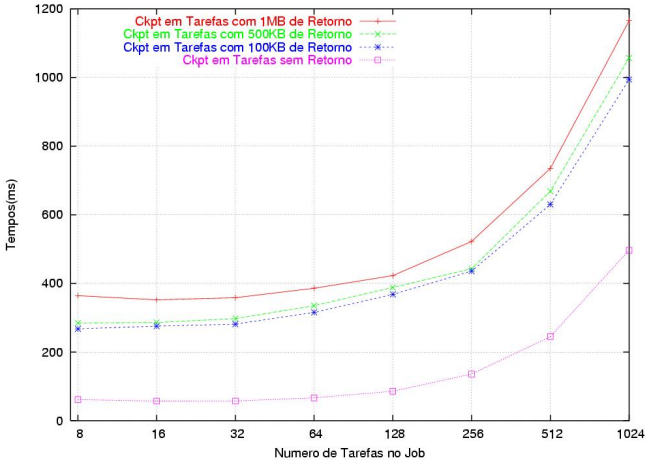

Figura 4: Tempo de realização de c*heckpoints*

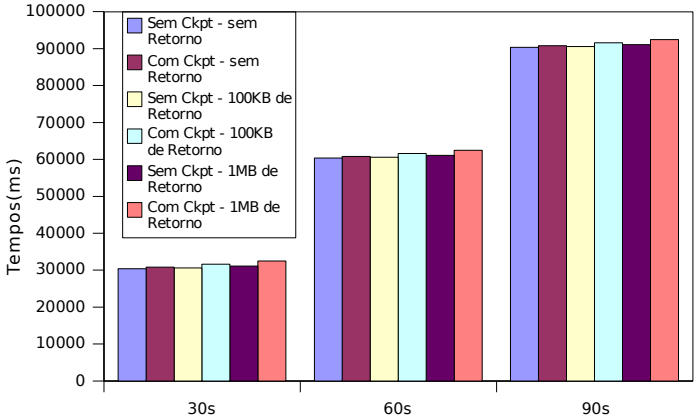

Figura 5: Tempo de realização de tarefas. São utilizados *jobs* de 512 tarefas.

No terceiro cenário é avaliado o impacto do mecanismo sobre o tempo total de execução dos *jobs*. Foram usadas cinco GuMs na mesma rede da máquina base. Neste experimento, o tempo exato entre a submissão de uma tarefa e o recebimento dos resultados não é determinístico. O mesmo vale para o tempo de serialização e salvamento dos resultados introduzido pelo mecanismo de recuperação. Por isso, a medida de tempo é feita num nível mais alto, no ato de submissão de um *job* e após a sua conclusão. A aplicação de teste multiplica matrizes quadradas, neste caso, de tamanho 800, 1000 e 1200. Para cada tarefa, são transferidas para a GuM duas matrizes a serem multiplicadas. A matriz resultante é retornada para a máquina base. O referencial de tempo é a execução das multiplicações sem ativar o mecanismo de *checkpointing*. Em cada *job* há 64 tarefas, isto é, são 64 multiplicações de matrizes distintas. Nos experimentos, o acréscimo causado pelo uso do mecanismo de *checkpointing* foi de 0,178%, 0,203% e 0,041% para os tamanhos de 800, 1000 e 1200,

respectivamente.

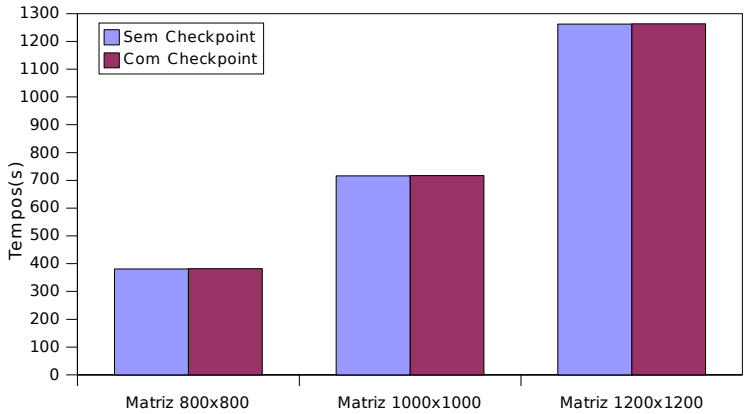

Figura 6: Tempo de Realização de Jobs

A sobrecarga observada no desempenho devida à inserção do mecanismo de *checkpointing* pode ser considerada pequena, visto que o tempo para concluir um *checkpoint* é diluído em meio às execuções (concorrentes) das tarefas.

Na prática, dada a diferença nos tempos de processamento das GuMs (para tarefas semelhantes), os resultados voltam em tempos diferentes, e o MyGrid pode salvar tanto o primeiro quanto o segundo *checkpoint* em momentos de inatividade, isto é, enquanto está aguardando resultados. Em outras palavras, ele pode salvar os *checkpoints* dentro da fatia de tempo de espera, que existe mesmo sem o mecanismo. A situação desfavorável é o recebimento de resultados em rajadas; contudo, estima-se que isto não ocorra com freqüência. Para tarefas que tenham tempos de execução muito semelhantes, o recebimento dos resultados acentua-se próximo ao tempo final de execução, o que não chega a degenerar o resultado para o pior caso. É importante destacar o bom resultado neste experimento, visto que certamente esta métrica é a que mais interessa ao usuário do sistema, por ser a mais perceptível do seu ponto de vista.

5. Conclusões

Com o objetivo de minimizar a perda de computação causada por colapsos na máquina base do OurGrid, foi projetado, implementado e incorporado um mecanismo de *checkpointing*. Este mecanismo permite aproveitar melhor os recursos do *grid,* pois evita a repetição de computações já realizadas. Contudo, a técnica de *checkpointing* utilizada não deve degradar significativamente o desempenho geral do sistema. Por isso, foram realizadas baterias de testes em diferentes cenários para validar o modelo e medir o impacto causado pelo mecanismo de *checkpointing*.

Exercitando-se o mecanismo de recuperação, o MyGrid comportou-se conforme o especificado, conseguindo, para todos os casos de teste, salvar e recuperar o último estado consistente do escalonamento das tarefas e restaurar a execução das aplicações. O incremento no tempo total de execução foi considerado negligenciável. Tal constatação é importante e justifica a implantação da técnica de salvamento de estados no escalonador. Demonstrou-se, também, que o fator de maior impacto no tempo de *checkpointing* é o número de *jobs* e tarefas que devem ser salvos, já que o tamanho dos *checkpoints* aumenta de acordo com crescimento destes números. Conclui-se, portanto, que o impacto causado pelo mecanismo de *checkpointing* foi bastante pequeno, o que justifica a sua utilização, já que o sistema agora é capaz de concluir sua execução mesmo na presença de falhas, sem a necessidade de refazer as computações concluídas ou em andamento no momento da falha.

Referências

Andrade, N.; Cirne, W.; Brasileiro, F. and Roisenberg, P. **OurGrid: An approach to easily assemble grids with equitable resource sharing**. Proceedings of 9th Workshop on Job Scheduling Strategies for Parallel Processing. June, 2003.

Balbinot, J. I.; Jansch-Pôrto, I.; Silva, H. A. M.; Weber, T. S. **Avaliação de um Mecanismo de Checkpointing para o MyGrid**. Anais do 6º WTF – Workshop de Testes e Tolerância a Falhas. Fortaleza – CE. 2005, pp. 39-50.

Birman, K. P., **Building Secure and Reliable Network Applications**. Publisher**:** Manning Publications. 5th Ed. 1996.

Bosilca, G. et al. **MPICH-V: Toward a Scalable Fault Tolerant MPI for Volatile Nodes**, Proceedings of ACM/IEEE Conference on Supercomputing, 2002. Baltimore, USA. Nov. 2002, pp. 1-18.

Cirne, W. ***Grids*** **Computacionais: Arquiteturas, Tecnologias e Aplicações**, Anais do 3º Workshop em Sistemas Computacionais de Alto Desempenho (WSCAD). Vitória, ES – Brasil. Out, 2002.

Cirne, W. et al. ***Grid*** **Computing for Bag of Tasks Applications**, Proceedings of 3th IFIP Conference on E-Commerce, E-Business and E-Goverment. São Paulo, SP - Brasil. Sept. 2004.

Costa, L.B. et al. **MyGrid: A complete solution for Running Bag-of-Tasks Applications**, Anais do Simpósio Brasileiro de Redes de Computadores – III Salão de Ferramentas (SBRC) Gramado, RS – Brasil. Maio, 2004.

Foster, I.; Kesselman, C.; **The Globus project: A status report.** Proceedings of IPPS/SPDP Heterogeneous Computing Workshop. Orlando, Florida – USA: IEEE Computer Society Press, Apr. 1998, pp. 4-18.

Foster I.; Kesselman C. **The grid 2: Blueprint for a New Computing Infrastructure**. Morgan Kaufmann Publishers Inc. 2003.

Grimshaw A. S.; Wulf WM. A. and The Legion Team. **The Legion Vision of a Worldwide Virtual Computer**. Communications of the ACM**,** New York, NY - USA, v. 40. n. 1, pp. 39-45, Jan. 1997.

OurGrid. Online Manual. In: http://www.ourgrid.org/ Universidade Federal de Campina Grande, 2005.

Paranhos, D.; Cirne, W.; Brasileiro, F. **Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids**, Proceedings of International Conference on Parallel and Distributed Computing (Lecture Notes in Computer Science), (EURO-PAR 2003). June, 2003.

Plank, J. **An Overview of Checkpointing in Uniprocessor and Distributed Systems, Focusing on Implementation and Performance**. Tecnical Report. Department of Computer Science University of Tennesse. Tennesse – USA, 1997.

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. 2004.