Alocação Dinâmica e Transparente de Computadores Ociosos em Java*

Márcia Cristina Cera, Rodrigo da Rosa Righi e Marcelo Pasin
Laboratório de Sistemas de Computação - Campus UFSM - Santa Maria, RS
{cera, rodrigor, pasin}@inf.ufsm.br

Resumo

*O uso de arquiteturas paralelas compostas pela interligação de computadores convencionais é cada vez mais freqüente. O modelo de programação paralela empregado nessas arquiteturas não é intuitivo, uma vez que sua memória está distribuída entre os computadores que a integram. Esse fato faz com que haja a necessidade de interação pela rede entre eles para a resolução de um problema em comum. Buscando simplificar o desenvolvimento de aplicações paralelas que possam ser executadas em sistemas distribuídos dessa natureza, foi desenvolvido o sistema Cadeo. O Cadeo (**C**ontrole e **a**locação **d**inâmica de **e**stações **o**ciosas) oferece um modelo simples de programação paralela, semelhante ao empregado em aplicações que executam sobre memória compartilhada. Ele disponibiliza dinamicamente computadores de sistemas distribuídos baseado na ociosidade dos mesmos, gerenciando-os de forma totalmente transparente à aplicação. Este artigo apresenta o Cadeo, bem como o desenvolvimento de um protótipo em Java desse sistema e a sua avaliação preliminar.*

1. Introdução

As arquiteturas paralelas resultantes da agregação de computadores convencionais, interligados por um dispositivo de rede, popularizaram-se nos últimos anos. O interesse por esse tipo de arquitetura está baseado, principalmente, na sua inferior relação de custo por unidade de processamento, quando comparado à outras soluções. Essas arquiteturas têm memória distribuída e podem ser generalizadas como sistemas distribuídos. Como exemplos desses sistemas distribuídos têm-se os aglomerados de computadores (*clusters*) [1] e as grades de computadores (*grids*) [6].

O desenvolvimento de aplicações para arquiteturas paralelas com memória distribuída não é trivial. A maioria dos modelos de programação voltados a essas arquiteturas deixam a exploração do paralelismo a cargo do programador na organização do código fonte. Uma maneira simples de idealizar um programa paralelo é considerá-lo como um conjunto de procedimentos, onde o paralelismo pode ser obtido através da chamada assíncrona deles. A programação direcionada a sistemas distribuídos implica na determinação da localização dos procedimentos no nível da aplicação. Para simplificar a programação nesses ambientes, seria interessante que a localização dos procedimentos fosse transparente à aplicação, principalmente em sistemas dinâmicos tais como grades de computadores.

Buscando maior produtividade e facilidade na implementação de aplicações paralelas em sistemas distribuídos, a orientação a objetos vem sendo cada vez mais empregada nesse cenário. Entre as linguagens orientadas a objetos, Java tem se destacado por sua simplicidade e portabilidade [8, 11]. Essa linguagem apresenta vantagens decorrentes da orientação a objetos além de possibilitar, nativamente, a programação concorrente através de múltiplos fluxos de execução (*multithreading*) e suporte a programação com memória distribuída: soquetes e RMI (*Remote Method Invocation*) [8]. Considerando o modelo de paralelismo baseado no assincronismo das chamadas de procedimentos, existem implementações assíncronas de RMI [2, 5, 14] que viabilizam o paralelismo na execução dos métodos remotos. Por outro lado, no nível da aplicação, a localização dos procedimentos segue não sendo transparente.

Com o intuito de simplificar o desenvolvimento de aplicações paralelas para sistemas distribuídos foi concebido o sistema Cadeo (**C**ontrole e **a**locação **d**inâmica de **e**stações **o**ciosas). O Cadeo possibilita que as aplicações paralelas sejam programadas através de um modelo semelhante ao empregado em aplicações para arquiteturas com memória compartilhada. Em linhas gerais, o Cadeo gerencia, transparentemente às aplicações paralelas, uma plataforma de execução dinâmica composta por computadores de sistemas distribuídos em ociosidade. No Cadeo, o termo **ocioso** caracteriza os computadores disponíveis a executar tarefas de aplicações paralelas. Um protótipo em Java foi desenvolvido para validar as idéias do Cadeo no qual as aplicações paralelas seguem um modelo baseado na invocação assíncrona de métodos.

* Pesquisa financiada pela CAPES.

O restante desse artigo está organizado da seguinte forma. A próxima seção mostra o sistema Cadeo apresentando sua idéia, arquitetura e o modo de funcionamento. A seção 3 apresenta detalhes da implementação de um primeiro protótipo do sistema. Em seguida, tem-se a avaliação do protótipo do Cadeo e a análise dos resultados obtidos. Após serão apresentados alguns trabalhos relacionados e por fim a conclusão do artigo.

2. Sistema Cadeo

O sistema Cadeo foi idealizado visando conciliar três aspectos de pesquisa: (i) simplicidade na programação paralela e distribuída; (ii) utilização de arquiteturas paralelas com memória distribuída; (iii) aproveitamento de computadores ociosos. Com base nessa tríade, o Cadeo visa disponibilizar meios para o desenvolvimento de aplicações paralelas e distribuídas de forma simples e intuitiva, oferecendo à elas transparência na localização de suas tarefas. O Cadeo disponibiliza computadores de sistemas distribuídos que estejam ociosos e administra a dinamicidade dos recursos que integram a plataforma de execução.

As arquiteturas paralelas alvo do Cadeo são aquelas compostas por computadores interligados via rede, tais como os aglomerados de computadores, as redes de estações de trabalho e as grades de computadores. O sistema disponibilizará às aplicações paralelas uma plataforma de execução dinâmica composta por computadores que estejam ociosos nas arquiteturas alvo. Essa plataforma de execução é capaz de adequar-se a constante entrada e saída de computadores e é chamada de **aglomerado dinâmico**. Essa última denominação será empregada no decorrer do artigo.

O Cadeo provê escalonamento para garantir um melhor aproveitamento dos computadores disponíveis e obter um equilíbrio na distribuição das cargas das aplicações paralelas. Optou-se por um modelo de escalonamento em dois níveis a fim de tratar separadamente a distribuição de computadores e o balanceamento de cargas das aplicações [10]. No primeiro nível de escalonamento são distribuídos computadores entre aplicações paralelas e, no segundo nível, tarefas entre computadores disponíveis.

2.1. Funcionamento e Arquitetura do Sistema

O funcionamento do Cadeo acontece da forma descrita a seguir. Quando um computador torna-se ocioso, ele informa ao Cadeo sobre sua disponibilidade em receber algum tipo de trabalho. Enquanto não houver nenhuma aplicação que necessite de recursos, os computadores disponíveis ao Cadeo permanecerão sem realizar processamento. Ao existir uma aplicação paralela a ser executada, o Cadeo, através de um alocador, destinará alguns dos computadores disponíveis para a execução dessa aplicação. Toda aplicação paralela que utiliza o Cadeo recebe um aglomerado dinâmico para sua execução.

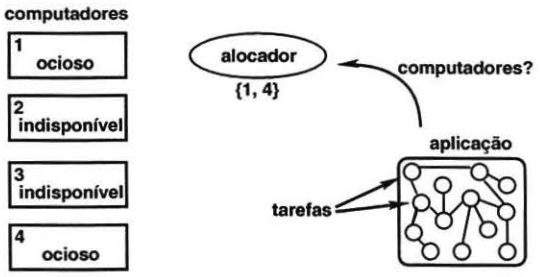

Figura 1. Cenário do Sistema Cadeo

A figura 1 apresenta um cenário do funcionamento do Cadeo. Nela têm-se um conjunto de computadores que podem estar em dois estados: ocioso ou indisponível. Tem-se um alocador com uma lista dos computadores ociosos que nesse caso são os computadores 1 e 4. Uma aplicação paralela solicita ao gerenciador computadores para a execução de suas tarefas, os quais serão concedidos conforme o conjunto de computadores ociosos. Para viabilizar o funcionamento coerente do Cadeo, estruturou-se sua arquitetura baseada em três módulos básicos: (i) alocador, (ii) trabalhador; (iii) escalonador. A seguir têm-se a descrição de cada um desses módulos.

Módulo Alocador O módulo alocador gerencia os computadores disponíveis ao Cadeo e executará em algum dos computadores do sistema. No cenário da figura 1 o alocador está representado pela elipse, onde ele mantém as informações dos computadores ociosos e aloca aglomerados dinâmicos às aplicações paralelas. O primeiro nível de escalonamento previsto no Cadeo está contemplado no módulo alocador. Para obter equilíbrio na distribuição dos computadores entre às aplicações paralelas o alocador emprega políticas de distribuição. Essas políticas consideram, além de aspectos como capacidade de processamento e quantidade de computadores disponíveis, a prioridade das aplicações e sua quantidade de tarefas associadas.

O Cadeo esconde das aplicações paralelas a dinamicidade da plataforma de execução. Supondo que um dos computadores de um aglomerado dinâmico deixe sua condição de ociosidade, por exemplo com a volta da utilização por seu usuário, o sistema contorna a perda desse recurso sem comprometer a execução da aplicação. Graças a essa característica e baseado nas políticas de distribuição e balanceamento de cargas, o alocador pode adicionar ou remover computadores dos aglomerados dinâmicos alocados às aplicações durante seus processamentos.

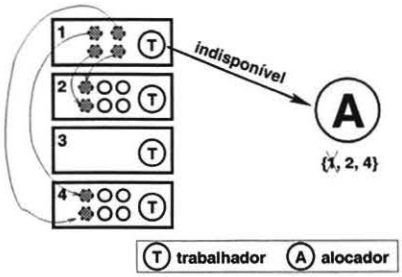

Figura 2. Notificação de indisponibilidade

Módulo Trabalhador Independente da arquitetura paralela ao qual pertençam, no Cadeo cada um dos computadores potencialmente ociosos possui um módulo trabalhador. Esse módulo é responsável pela execução das tarefas das aplicações e por manter o alocador ciente da condição de disponibilidade (ocioso ou indisponível) daquele computador. A condição de disponibilidade varia conforme o tipo de sistema distribuído. Por exemplo, computadores de um aglomerado alocados através de seu gerenciador de recursos são computadores ociosos no Cadeo, assim como computadores de uma rede de estações de trabalho que não estão sendo utilizados por seus usuários.

O trabalhador é encarregado de avisar ao alocador quando um computador torna-se indisponível, ou seja, deixa de estar a disposição do sistema Cadeo. Porém, se existirem tarefas em execução no computador, o trabalhador deverá providenciar a transferência ou relançamento delas em outros computadores. Um exemplo do comportamento do trabalhador em um caso desses é apresentado na figura 2, onde as tarefas em execução estão sendo migradas a outros computadores do aglomerado dinâmico.

Módulo Escalonador O segundo nível de escalonamento previsto para o Cadeo foi estruturado no módulo escalonador. Este módulo é responsável por solicitar computadores ao alocador e distribuir as tarefas da aplicação entre os computadores que receber. O Cadeo foi organizado de modo que existirá um escalonador associado a cada aplicação paralela.

Conforme a demanda por processamento da aplicação paralela o escalonador irá solicitar um aglomerado dinâmico ao alocador antes do inicio da execução da aplicação. O alocador irá avaliar a disponibilidade de computadores e alocará um aglomerado dinâmico buscando melhor atender a demanda da aplicação. A figura 3 mostra o escalonador solicitando computadores e o alocador concedendo-lhe um aglomerado dinâmico.

Em posse de um aglomerado dinâmico o escalonador distribuirá as tarefas da aplicação e gerenciará suas execuções, comunicando-se diretamente com os trabalhadores dos computadores do aglomerado dinâmico. Um

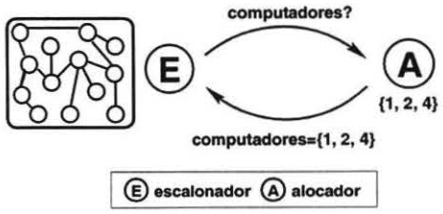

Figura 3. Alocação de aglomerado dinâmico

exemplo da interação entre escalonador e os computadores do aglomerado pode ser visto na figura 4. Nela, cada um dos computadores do aglomerado dinâmico recebe uma tarefa a ser processada.

Ao ser concluída a execução de todas as tarefas da aplicação paralela os computadores serão desassociados daquele aglomerado dinâmico e devolvidos ao alocador. Esses computadores poderão ser imediatamente repassados para aglomerados dinâmicos que carecem por recursos ou então permanecer em posse do alocador até que uma nova requisição aconteça.

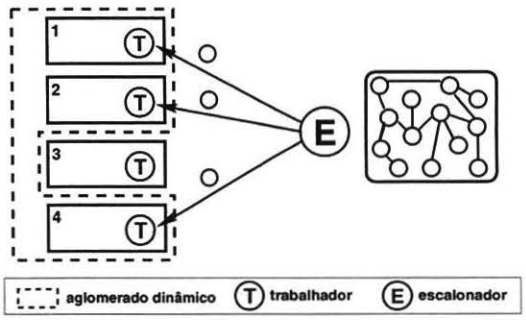

Figura 4. Lançamento de Tarefas

3. Implementação do Protótipo do Sistema

3.1. Decisões de Projeto

Um protótipo do Cadeo foi implementado usando a linguagem Java, visando aproveitar seu suporte à programação concorrente e tirar proveito das vantagens da orientação a objetos no projeto do sistema. O ponto negativo dessa linguagem é referente ao seu desempenho, pois Java é naturalmente interpretada e agrega um sobrecusto ao desempenho das aplicações quando comparada a outras linguagens não interpretadas [9]. Muitas iniciativas têm buscado amenizar a diferença de desempenho de Java, como exemplo compiladores diretos, compiladores JIT (*Just In Time*) e

serialização mais eficiente de objetos [9, 12]. Essas iniciativas têm obtido melhoras no desempenho de Java e estimulam sua utilização.

Em função do uso de Java, a solução imediata para o lançamento das tarefas das aplicações paralelas seria sua implementação via RMI. Porém, atualmente, o RMI padrão do Java atua de forma síncrona e a execução paralela das tarefas não seria trivial, sendo assim buscou-se uma implementação de RMI assíncrono. Optou-se pelo assincronismo oferecido pela biblioteca ProActive [2, 3]. O ProActive utiliza objetos futuros os quais representam um retorno imediato do resultado de métodos invocados e ainda não processados. Quando os retornos dos métodos estiverem disponíveis eles substituirão transparentemente os objetos futuros. A biblioteca trabalha com o conceito de objetos ativos onde cada objeto tem um fluxo de execução associado a ele. O ProActive se encarrega, transparentemente, do registro dos objetos remotos e do assincronismo nas invocações de métodos, porém todos os aspectos relacionados a alocação dos objetos são de responsabilidade do programador.

Considerando o contexto oferecido pelo ProActive, buscou-se definir como as tarefas das aplicações paralelas seriam representadas no Cadeo. Em uma primeira possibilidade, as tarefas compreenderiam as chamadas RMI assíncronas. Nesse caso, a execução das tarefas seria dependente da execução das chamadas remotas no código da aplicação. Essa dependência caracterizaria um modelo de aplicação semelhante ao modelo dividir para conquistar (*divide-and-conquer*) [16]. Nesse modelo, o escalonamento das tarefas aconteceria no momento das chamadas remotas, onde elas seriam distribuídas entre os computadores disponíveis e, em conseqüência, aconteceria também a distribuição dos objetos ativos.

Em uma segunda alternativa, as tarefas seriam objetos ativos, onde chamadas RMI assíncronas representariam possíveis interações entre as tarefas. Neste caso, as interações também representariam dependências entre as tarefas, porém, o fluxo de execução dos objetos ativos poderia realizá-las. Dessa forma é possível conceber um modelo de programação para as aplicações do tipo sacola de tarefas (*bag-of-tasks*) [4], onde as tarefas são independentes entre si. Para esse modelo, o escalonamento aconteceria na instanciação dos objetos, onde eles seriam distribuídos entre os computadores disponíveis no aglomerado dinâmico. No primeiro protótipo do Cadeo, optou-se pelo modelo mais simples de aplicação onde as tarefas são objetos ativos. Embora simples, o modelo sacola de tarefas é empregado na solução de problemas importantes tais como simulações, processamento de imagens, mineração de dados entre outras. Esse modelo possibilita a adequação da aplicação a dinamicidade da plataforma de execução já que suas tarefas são independentes [4].

Com a definição de como seriam representadas as tarefas no Cadeo, buscou-se definir seu comportamento quando os computadores que às executam tornam-se indisponíveis. As opções seriam abortar suas execuções e relançá-las posteriormente, ou mantê-las em execução com prioridade baixa, ou então migrá-las a outros computadores disponíveis. Considerando que o ProActive disponibiliza migração de objetos ativos, o protótipo atual do Cadeo utilizou esse mecanismo e migra as tarefas dos computadores indisponíveis.

3.2. Estrutura básica do Protótipo

O enfoque do protótipo do sistema foi a estruturação e criação de uma base funcional para o Cadeo. Para possibilitar o funcionamento do protótipo, foram empregadas algumas **políticas de decisão** *ad hoc*. O uso dessas políticas não esteve condicionado a estudos que considerassem o contexto e características do Cadeo e que pudessem avaliar se estas seriam as melhores políticas a serem empregadas. No protótipo atual o sistema é capaz de oferecer transparência na distribuição e localização das tarefas bem como suporte a utilização de aglomerados dinâmicos como plataforma de execução. Ainda, procurou-se planejar e desenvolver a base funcional do sistema o mais adaptável possível, ou seja, buscou-se implementar um protótipo capaz de facilmente acoplar novas características e implementações visando seu aperfeiçoamento. A base funcional do Cadeo compreendeu a implementação dos três módulos básicos (alocador, escalonador e trabalhador) e da correta interação entre eles.

O módulo alocador foi implementado de forma centralizada para simplificá-lo. Porém, tomou-se o cuidado de planejá-lo de modo que uma implementação distribuída pudesse ser desenvolvida futuramente, para contornar possíveis sobrecargas nesse módulo. O alocador é representado por um objeto da classe Allocator, a qual disponibiliza aos demais módulos do sistema quatro métodos. Dois desses métodos, addWorker() e subtractWorker(), encarregam-se, respectivamente, da inclusão e exclusão de computadores ociosos no sistema e são invocados pelos módulos trabalhadores. Os dois métodos restantes são invocados pelos módulos escalonadores e encarregam-se de atender a solicitações por aglomerados dinâmicos (getCluster()) e pela devolução de aglomerados após a execução das aplicações (returnCluster()).

Segundo a estrutura do Cadeo, os módulos escalonador e trabalhador precisam interagir com o alocador, e para isso ambos são ligados ao módulo alocador que é previamente conhecido. Esta decisão simplificou a implementação do protótipo, mas espera-se que, futuramente, o alocador possa ser encontrado automaticamente, por exemplo através de servidores de recursos ou serviços [15].

O módulo trabalhador é representado pela classe Worker e apresenta três métodos principais. O método idle() in-

forma ao alocador sobre a ociosidade do computador onde o trabalhador se encontra. Na mesma linha, o método busy() informa ao alocador sobre a indisponibilidade de um computador. A execução de ambos os métodos estará vinculada ao estado de disponibilidade do computador, informado por um mecanismo de controle. Esse mecanismo, atualmente, varia conforme a natureza do sistema distribuído, por exemplo um detector de ociosidade numa rede de estações de trabalho ou gerenciador de recursos num aglomerado. Um terceiro método do trabalhador, chamado migrate(), é executado conjuntamente com o método busy() e migra tarefas em execução quando um computador torna-se indisponível.

O módulo escalonador é implementado através da classe Scheduler e toda aplicação paralela possuirá um objeto dessa classe. A classe Scheduler tem o método start(), invocado na criação do módulo, que solicita um aglomerado dinâmico para a execução da aplicação paralela. O método stop() irá desencadear a devolução dos computadores de um aglomerado dinâmico ao alocador no fim da execução da aplicação. Para suportar a dinamicidade do aglomerado dinâmico, a classe Scheduler implementa, assim como a classe Allocator, os métodos addWorker() e subtractWorker() que serão invocados pelo alocador a fim de ajustar o aglomerado. Quando é necessário migrar tarefas, o método getResource() da classe Scheduler retorna ao trabalhador o endereço de um computador apto a receber a tarefa em migração.

3.3. Interface para Utilização do Cadeo

No protótipo atual do Cadeo, o módulo alocador estará em algum dos computadores do sistema e sua localização deverá ser conhecida pelos trabalhadores e escalonadores. Todos os computadores potencialmente ociosos ao Cadeo possuirão um módulo trabalhador e este será seu elo com o sistema. As aplicações paralelas interagem com o Cadeo através de uma classe chamada Cadeo onde seus métodos oferecem todas as funcionalidades necessárias às aplicações. Toda aplicação paralela possui um objeto da classe Cadeo e, transparentemente, durante a criação desse objeto o módulo escalonador é instanciado e solicita um aglomerado dinâmico ao alocador. A classe Cadeo possui dois métodos básicos fork() e stop(). O primeiro deles instancia remotamente os objetos que representam as tarefas distribuindo-as entre os computadores disponibilizados à aplicação. Os parâmetros esperados pelo método fork() não incluem o endereço do computador remoto, já que sua localização é responsabilidade do Cadeo. O método stop() indica o fim da aplicação paralela e desencadeia a devolução ao alocador dos computadores do aglomerado dinâmico.

Um exemplo de aplicação implementada com o Cadeo é apresentado nas figuras 5 e 6. A figura 5 apresenta um pseudo-código de uma classe que representa as

```
1. public class Tarefa implements Serializable{
2.   String mensagem;
3.   public Tarefa() {
4.     /* Construtor da Classe */
5.   }
6.   public trabalho(parâmetros) {
7.     /* Realiza algum processamento */
8.     retorna Resultado;
9.   }
10. }
```

Figura 5. Pseudo-código das Tarefas

```
1. public class Principal {
2.   public static void main(String[] args) {
3.     Cadeo c = new Cadeo(endereço_alocador);
4.     Tarefa t[] = new Tarefa[num_tarefas];
5.     Resultado res[] = new Resultado[num_tarefas];
6.     for(int i = 0; i < num_tarefas; i++){
7.       t[i]=(Tarefa)c.fork(Tarefa.class.getName(),
                                        new Object[]{});
8.       res[i] = t[i].trabalho(parâmetros);
9.     }
10.    c.stop();
11.  }
12. }
```

Figura 6. Pseudo-código de classe Principal

tarefas da aplicação onde, no método trabalho(), é realizado algum processamento com o retorno de seu resultado. Na figura 6 tem-se o pseudo-código da classe Principal da aplicação. Na linha 3 ocorre a criação do objeto da classe Cadeo, nesse momento é criado pelo Cadeo um escalonador que solicita ao alocador um aglomerado dinâmico para a execução da aplicação. Na linha 7 acontece a instanciação remota das tarefas nos computadores do aglomerado dinâmico onde é informada a classe que representa as tarefas e seus parâmetros. A linha 8 apresenta a invocação dos métodos remotos sendo que os resultados serão armazenados em um vetor. Por fim, a aplicação é finalizada na linha 10.

4. Avaliação do Protótipo

Para avaliar o funcionamento do protótipo do Cadeo foi implementada uma aplicação paralela simples, na qual um conjunto de tarefas realizam uma série de operações sem fim específico. A vantagem dessa aplicação de teste é a facilidade para alterar o tamanho do grão das tarefas através da variação do conjunto de operações realizadas nelas. O parâmetro de entrada das tarefas é um vetor de inteiros e as operações são realizadas sobre os elementos desse vetor. O vetor de inteiros passado como parâmetro é retornado ao fim da execução da tarefa para que a quantidade de dados transferida na solicitação e no retorno sejam iguais. As tarefas da aplicação são controladas por um mestre que informa o vetor, solicita as operações e recebe os dados de retorno ao final das execuções.

O ambiente de execução dos testes foi o aglomerado de computadores do Laboratório de Sistemas de Computação da UFSM. As máquinas utilizadas foram 7 computadores Pentium III duais de 1GHz, 768 MB de memória principal e uma rede Fast Ethernet. O aglomerado em questão é uma plataforma de execução estática. Para simular um comportamento dinâmico um dos computadores que integra o aglomerado dinâmico da aplicação é escolhido aleatoriamente e excluído do mesmo. Após um determinado período, o computador excluído torna-se disponível e novamente é incluído no aglomerado da aplicação. A alternância dos períodos de disponibilidade, que foi chamada de freqüência de dinamicidade, é feita de tal forma que seja possível adequar um certo número de saídas e entradas de computadores no aglomerado dinâmico durante a execução da aplicação.

O objetivo dessa seção é apresentar uma avaliação preliminar do protótipo inicial do Cadeo sendo buscadas duas metas principais. A primeira delas visou determinar a influência da utilização do Cadeo no desempenho das aplicações. A segunda meta foi demonstrar que o sistema consegue administrar, de forma transparente à aplicação, a localização dos computadores do aglomerado dinâmico. Além disso, pode-se verificar a influência da utilização de um ambiente dinâmico na execução de aplicações paralelas. Em função das características da aplicação de teste, foi possível analisar os tempos de comunicação e computação das tarefas, estimando diferentes valores de granularidades.

4.1. Granularidade das tarefas

As cargas de processamento das tarefas foram variadas até que se estabelecessem três opções de granularidade padrões para os testes realizados na avaliação do protótipo. As definições de granularidades foram realizadas entre o mestre e um único escravo, onde a invocação de tarefas aconteceu em modo síncrono. Na estimativa de granularidades foram coletados os tempos relativos a invocação remota, a computação e ao retorno dos dados das tarefas, onde o tempo de invocação e retorno compreendem o tempo de comunicação das tarefas.

A primeira granularidade é aproximadamente igual a 1, isto é, apresenta um equilíbrio entre os tempos gastos na comunicação e computação ($t_{comp} = 1 * t_{comun}$). Este valor de granularidade pode ser visto na primeira linha da tabela 1 onde os tempos de computação e comunicação são iguais a 27 segundos. A segunda linha da tabela 1 apresenta a medida de granularidade aproximadamente igual a 10, onde o tempo de computação é 10 vezes maior que o tempo de comunicação ($t_{comp} = 10 * t_{comun}$). E, por fim, a terceira linha da tabela com a granularidade aproximada a 100 ($t_{comp} = 100 * t_{comun}$). Percebe-se que com o aumento da granularidade das tarefas houve um pequeno acréscimo no tempo de comunicação. Todos os valores apresentados na tabela 1 estão em segundos e representam as médias aritméticas de 1000 execuções de tarefas.

granularidade	Computação	Comunicação
1	27,0	27,0
10	270,1	27,0
100	2971,0	30,0

Tabela 1. Padrões de granularidades

4.2. Sobrecarga na utilização do Cadeo

Uma vez que o Cadeo foi implementado fazendo uso do ProActive, buscou-se avaliar qual seria a interferência no tempo de execução das aplicações gerada pelo uso do Cadeo. Para isso foram implementadas duas versões idênticas da aplicação de teste onde tem-se uma delas fazendo uso somente do ProActive e outra com o Cadeo. Os testes ocorreram sobre uma plataforma de execução estática, uma vez que o ProActive não suporta automaticamente a dinamicidade de computadores. A configuração escolhida para a aplicação de teste era composta pelo mestre e 12 escravos. Essa escolha deu-se em virtude da plataforma de execução disponível possuir 7 computadores biprocessados e por almejar-se o máximo grau de paralelismo da aplicação.

granularidade	ProActive	Cadeo
1	23,085	23,333
10	43,043	43,048
100	267,841	272,088

Tabela 2. Médias do ProActive e Cadeo

A aplicação de teste é composta por 1000 tarefas e a tabela 2 apresenta as médias, em segundos, dos tempo de execução utilizando o ProActive e o Cadeo, variando-se a granularidade das tarefas. Comparando os tempos com ProActive e Cadeo percebe-se uma diferença muito pequena entre eles, sendo que a maior delas é com granularidade 100 onde obteve-se uma média de 267,8 segundos com o ProActive e 272,0 com o Cadeo. Logo constata-se que a utilização do Cadeo não oferece grande sobrecarga ao tempo de execução da aplicação. Entretanto, por trás do processo de instanciação de objetos com o Cadeo existe uma infraestrutura que determina a localização dos objetos de forma transparente à aplicação. Já com o ProActive é necessário fixar o conjunto de computadores e determinar, no nível da aplicação, a localização dos obje-

tos durante sua instanciação. Com os resultados encontrados constatou-se que todo o processo que envolve a transparência na localização dos computadores oferecida pelo Cadeo não gera sobrecarga no desempenho das aplicações.

4.3. Testes com Ambiente Dinâmico

A aplicação de teste foi executada sobre um ambiente dinâmico a fim de avaliar o comportamento do protótipo do Cadeo. Em virtude das decisões de projeto adotadas, o desempenho da aplicação paralela de teste em um ambiente dinâmico refletirá a sobrecarga causada pela migração de tarefas. Os testes foram realizados seguindo a mesma configuração adotada para os testes da sub-seção anterior onde simulou-se um comportamento dinâmico para a plataforma de execução.

Na coleta de dados utilizou-se o padrão de granularidade 100, por esse apresentar um maior tempo de execução, onde se poderia perceber melhor os efeitos da dinamicidade do ambiente. O gráfico da figura 7 apresenta os tempos, em segundos, encontrados na execução (tempo gasto na execução das 1000 tarefas) da aplicação. Para cada uma das freqüências de dinamicidade testadas, o gráfico apresenta o ponto médio dos tempos encontrados, juntamente com o intervalo entre o maior e o menor tempo obtido.

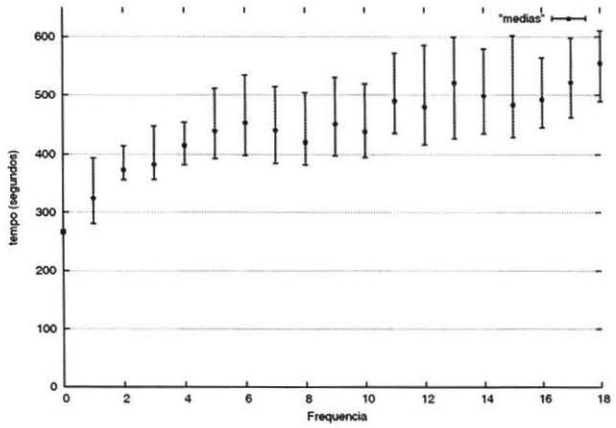

Figura 7. Tempos em um ambiente dinâmico

No gráfico da figura 7 observa-se que o intervalo de tempo com freqüência de dinamicidade zero é menor que os intervalos de quaisquer outras freqüências testadas. Tal comportamento era esperado pois, na variação do conjunto de computadores do aglomerado dinâmico, a migração das tarefas gera sobrecarga à aplicação. A migração de objetos utilizada pelo Cadeo (provida pelo ProActive) é do tipo fraca. Nesse tipo de migração, para que um objeto seja efetivamente migrado é necessário esperar o fim da execução de seus métodos que já tenham sido invocados, o que pode gerar uma grande sobrecarga.

Ao observar o gráfico percebe-se uma tendência de elevação dos tempos de execução conforme o aumento da freqüência de dinamicidade. Esse comportamento justifica-se pois, quanto maior a freqüência de dinamicidade, maior tende a ser a necessidade por migrações de tarefas. Pode-se observar também que os limites dos intervalos dos tempos de execução variam inconstantemente entre as diferentes freqüências utilizadas. Esse comportamento reflete o emprego de uma política de distribuição *ad hoc* na implementação do protótipo do Cadeo. A política de distribuição empregada objetiva viabilizar o funcionamento do protótipo sem almejar manter um equilíbrio entre as cargas de processamento dos computadores.

5. Trabalhos Relacionados

O Condor [10] é um sistema que disponibiliza computadores baseado na ociosidade dos mesmos. Esse sistema objetiva maximizar a utilização dos recursos com a menor interferência possível entre as tarefas escalonadas e as atividades do usuário do computador. Nesse contexto, ele se propõe a oferecer grande quantidade de processamento, com desempenho uniforme, a médio e longo prazo. O Condor possui uma idéia bem estabelecida e vem sendo adaptado as evoluções dos sistemas distribuídos, onde sua versão Condor-G [7] interage com o Globus para seu uso em grades de computadores.

Em um escopo diferente do Cadeo, o Condor não foi originalmente concebido para a programação paralela e sim para possibilitar uma melhor utilização dos recursos. O Condor possui dois níveis de escalonamento, onde o primeiro distribui tarefas seqüenciais e independentes entre computadores e o segundo escalona a execução das tarefas nos recursos. O sistema implementa pontos de verificação (*checkpoints*) que evitam a perda do processamento já realizado quando um computador deixa de estar ocioso. O Cadeo apresenta algumas idéias semelhantes ao Condor por acreditar-se que estas se enquadram satisfatoriamente em seu contexto voltado a programação paralela.

O MyGrid [4], assim como o Cadeo, é voltado a aplicações paralelas do tipo sacola de tarefas. No MyGrid as tarefas são distribuídas entre os computadores pertencentes a grade de um usuário, a qual é composta por todos os computadores que o usuário tem acesso. A principal diferença entre o MyGrid e o Cadeo está na estrutura de escalonamento dos sistemas. O MyGrid oferece escalonamento de tarefas através de um algoritmo de fila de trabalho com replicação (*Work Queue with Replication - WQS*). Nesse algoritmo, as tarefas são replicadas em diferentes computadores buscando contornar a hetero-

geneidade de computadores e tarefas e a dinamicidade de computadores. A replicação ajuda a contornar tais problemas oferecendo um desempenho razoável, mas não resolve o problema completamente [13].

6. Conclusão

Este artigo apresentou o projeto e protótipo do sistema Cadeo. Ele possibilita a utilização de computadores de sistemas distribuídos baseado na ociosidade dos mesmos. A principal característica do Cadeo é o gerenciamento, de forma totalmente transparente às aplicações paralelas, de uma plataforma de execução dinâmica. Dessa forma, o Cadeo permite a implementação de aplicações paralelas seguindo um modelo semelhante ao de aplicações para memória compartilhada.

Um protótipo, em Java, do Cadeo foi implementado objetivando conceber uma estrutura básica e funcional, capaz de suportar adaptações futuras. Não houve preocupação em incorporar os melhores algoritmos para implementar as políticas de decisão do sistema. Por outro lado, a versão implementada segue estritamente o modelo projetado, já permitindo escrever aplicações que tiram proveito da transparência e do assincronismo na invocação de tarefas.

Os resultados obtidos na avaliação do protótipo do Cadeo comprovaram que a sobrecarga gerada pela utilização do Cadeo é muito pequena quando comparada ao uso exclusivo do ProActive. Para tarefas de granularidade 100, a aplicação com o Cadeo levou 272,0 segundos contra 267,8 segundos da aplicação com ProActive. Também pode-se constatar que o sistema gerencia de forma satisfatória os computadores da plataforma de execução dinâmica. Em ambientes dinâmicos, o desempenho das aplicações passou a agregar a sobrecarga decorrente da migração de objetos em Java, refletindo a decisão de projeto de migrar tarefas de computadores que deixam de estar disponíveis.

Por ser a primeira implementação do protótipo do Cadeo e pela grande variedade de aspectos que envolvem o sistema, a seguir têm-se algumas das questões a serem tratadas em trabalhos futuros: Implementação distribuída e localização automática do módulo alocador; Possibilitar a implementação de aplicações no modelo dividir para conquistar; Pesquisar políticas de escalonamento e balanceamento de cargas que melhor se enquadram no contexto do sistema; Investigar alternativas ao uso da migração de objetos a fim de eliminar sua sobrecarga no sistema.

Referências

[1] M. Baker and R. Buyya. Cluster computing at a glance. In R. Buyya, editor, *High Performance Cluster Computing*, volume 1, Architectures and Systems, pages 3–47. Prentice Hall PTR, Upper Saddle River, NJ, 1999. Chap. 1.

[2] D. Caromel, L. Henrio, and B. Serpette. Asynchronous and deterministic objects. In *Proceedings of the 31st ACM Symposium on Principles of Programming Languages*, pages 123–134. ACM Press, 2004.

[3] D. Caromel, W. Klauser, and J. Vayssiere. Towards seamless computing and metacomputing in java. In G. C. Fox, editor, *Concurrency Practice and Experience*, volume 10, pages 1043–1061. Wiley & Sons, Ltd., 1998.

[4] W. Cirne, D. Paranhos, L. Costa, E. Santos-Neto, F. Brasileiro, J. Sauvé, F. A. B. da Silva, C. O. Barros, and C. Silveira. Running bag-of-tasks applications on computational grids: The MyGrid approach. In *International Conference on Parallel Processing*, Kaohsiung, Taiwan, ROC, 2003.

[5] K. E. K. Falkner, P. D. Coddington, and M. J. Oudshoorn. Implementing Asynchronous Remote Method Invocation in Java. In *Parallel and Real-Time Systems (PART'99)*, pages 22–34, Melbourne, Australia, 1999.

[6] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a Future Computing Infrastructure*. MORGAN-KAUFMANN, San Francisco, California, 1999.

[7] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-g: A computation management agent for multi-institutional grids. In *10th IEEE International Symposium High Performance Distributed Computing*, pages 55–63, San Francisco, USA, 2001. IEEE Comp. Society Press.

[8] V. Getov, G. von Laszewski, M. Philippsen, and I. Foster. Multiparadigm communications in Java for grid computing. *Communications of the ACM*, 44(10):118–125, Oct. 2001.

[9] I. H. Kazi, H. H. Chen, B. Stanley, and D. J. Lilja. Techniques for obtaining high performance in java programs. *ACM Computing Survey*, 32(3):213–240, 2000.

[10] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor : A hunter of idle workstations. In *8th International Conference on Distributed Computing Systems*, pages 104–111, Washington, D.C., USA, June 1988. IEEE Computer Society Press.

[11] M. Lobosco, C. L. de Amorim, and O. Loques. Java for high-performance network-based computing. *Concurrency and Computation: Practice and Experience*, 14(1):1–31, 2002.

[12] J. Maassen, R. V. Nieuwpoort, R. Veldema, H. Bal, T. Kielmann, C. Jacobs, and R. Hofman. Effi cient Java RMI for parallel programming. *ACM Transactions on Programming Languages and Systems*, 23(6):747–775, Nov. 2001.

[13] D. Paranhos, W. Cirne, and F. Brasileiro. Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids. In *Euro-Par 2003: International Conference on Parallel and Distributed Computing*, Klagenfurt, Austria, 2003.

[14] R. R. Raje, J. I. Williams, and M. Boyles. Asynchronous Remote Method Invocation (ARMI) mechanism for Java. *Concurrency: Practice and Experience*, 9(11):1207–1211, 1997.

[15] A. E. Schaeffer Filho, L. C. da Silva, A. C. Yamin, I. Augustin, and C. F. R. Geyer. PerDiS: Um modelo para descoberta de recursos na arquitetura ISAM. In *VI Workshop de Comunicação Sem Fio e Computação Móvel*, pages 98–107, Fortaleza, Brasil, Outubro 2004.

[16] R. V. van Nieuwpoort, T. Kielmann, and H. E. Bal. Satin: Effi cient Parallel Divide-and-Conquer in Java. In *Euro-Par 2000 Parallel Processing*, Lecture Notes in Computer Science, pages 690–699, Munich, Germany, 2000. Springer.