1
Introdução

A computação ocupa posição de destaque no cenário global. Esse destaque é impulsionado pelas necessidades provenientes da sociedade, condicionada às facilidades de utilização dos computadores, e por um gradual aumento do leque de aplicações que podem ser tratados computacionalmente.

O crescimento em poder computacional e capacidade de armazenamento alcançado pelos computadores atuais é cada vez maior. Por outro lado, a quantidade de problemas complexos que estão sendo tratados por cientistas, físicos, biólogos e engenheiros também cresce em muito, exigindo uma demanda computacional que muitas vezes não é disponibilizada por uma única máquina, por mais robusta que essa seja [33, 64].

Essa realidade começou a ser transformada a partir da década de 70, quando vieram à tona os sistemas computacionais distribuídos, que surgiram como uma evolução natural para substituir os sistemas centralizados (os chamados mainframes) [19].

Na década de 80, motivadas pelo alto custo de aquisição e manutenção de máquinas paralelas, assim como pela dependência estabelecida entre o usuário e o fabricante da máquina, as comunidades envolvidas com pesquisa em sistemas distribuídos e computação paralela propuseram a utilização de sistemas distribuídos como plataforma de execução paralela [53]. Isso concretizou uma melhor relação custo/benefício para a computação paralela, pois proporcionou menor custo de implantação e maior poder computacional (em relação às arquiteturas seqüenciais) a uma ampla variedade de aplicações.

Nesse cenário, a computação em *clusters* [22] teve um papel de destaque, possibilitando o acesso a uma plataforma paralela de alto desempenho e custo mais acessível (se comparado aos supercomputadores). Contudo, apesar do baixo custo dos *clusters*, esses sistemas não são amplamente usados porque normalmente são dedicados a um número restrito de usuários, o que pode acarretar ociosidade durante grande parte do tempo. Além disso, devido à crescente demanda por poder computacional requerida por aplicações cada vez mais robustas, os *clusters* já não são capazes de atender às necessidades apresentadas por diversas áreas.

Com o advento do *grid* computacional [64, 66, 68] a custo reduzido, é possível acessar um número bem maior de recursos. O objetivo desse novo ambiente é oferecer uma plataforma suficientemente robusta capaz de executar aplicações que necessitem de um alto poder de processamento ou de armazenamento. Com essa tecnologia é possível visualizar um meta-computador composto por milhares de recursos pertencentes a organizações distintas.

Os *grids* são capazes de oferecer esse ambiente robusto ao agregar recursos distribuídos geograficamente, heterogêneos e, geralmente, pertencentes a diferentes domínios e compartilhados por múltiplos usuários. Em cada local pode haver os mais diversos recursos, tais como uma ou mais máquinas mono-processadas, *clusters*, sistemas multicomputadores com memória distribuída, sistemas multiprocessadores de memória compartilhada ou até mesmo máquinas maciçamente paralelas [66].

Entretanto, a exploração eficiente desse tipo de ambiente ainda é um desafio [122]. Dentre as principais dificuldades é possível mencionar escalabilidade, heterogeneidade em múltiplos níveis, estrutura instável do sistema porque é construída dinamicamente a partir dos recursos disponíveis, volatilidade relacionada a falhas na rede, e múltiplos domínios administrativos, cada um com sua própria política administrativa que deve ser preservada pelo sistema *grid* [67]. Dessa forma, para garantir que as aplicações possam se beneficiar do ambiente sem o conhecimento das complexidades inerentes ao *grid*, é fundamental desenvolver tecnologia que possa tornar transparente a gerência dessa plataforma.

Para resolver esse problema, uma solução adotada tem sido a implementação de um *middleware*, camada de *software* cujo objetivo é aliviar a carga do programador na hora de projetar, programar e gerenciar aplicações distribuídas, fornecendo aos desenvolvedores a imagem única de um ambiente de programação distribuído, integrado e consistente [100]. *Middlewares* facilitam a utilização eficiente e segura da infra-estrutura disponível, permitindo que aplicações aproveitem os benefícios desse ambiente sem o conhecimento das características específicas dos recursos distribuídos. Assim sendo, essa camada de *software* implanta o conceito de transparência na gerência dos recursos.

Problemas de otimização combinatória surgem, na prática, como um amplo universo de aplicações a serem tratadas pelo paralelismo em ambientes *grids*, pois freqüentemente são NP-difíceis e, conseqüentemente, consomem muito tempo de CPU. Metaheurísticas são procedimentos de alto nível que coordenam heurísticas simples, objetivando explorar eficiente e efetivamente o espaço de busca para encontrar boas soluções aproximadas (freqüentemente ótimas) para problemas de otimização combinatória. As metaheurísticas são

uma das técnicas usadas para solucionar difíceis problemas de otimização.

Cung et al. [46] mencionam que a paralelização de metaheurísticas não visa tão somente diminuir os tempos de processamento, tornando possível a resolução de problemas maiores e mais realistas, mas também possibilita encontrar soluções melhores do que as obtidas por algoritmos seqüenciais. Nesse contexto, várias metaheurísticas foram paralelizadas com diferentes técnicas, a grande maioria desenvolvida para ambientes de *clusters* [9, 10, 43, 59, 104, 121].

As pesquisas em *grids* computacionais, consideram principalmente duas classes de aplicações. A primeira, chamada de *bag-of-tasks*, é formada por aplicações compostas apenas de tarefas independentes, que caracterizam-se pelo fato de poder ativar qualquer tarefa em qualquer recurso e ordem de execução. A segunda classe, consistindo de aplicações *workflow*, é formada por aplicações em que as tarefas realizam comunicação entre si, sendo necessário ter conhecimentos sobre a relação de precedência entre as tarefas. As metaheurísticas não se enquadram diretamente em nenhuma das classes acima, pois suas tarefas podem executar de forma totalmente independente (como as aplicações *bag-of-tasks*), mas também podem realizar comunicação entre si (como as aplicações *workflow*). Nesse último caso, a comunicação pode ocorrer sem que haja uma relação previamente definida de precedência entre as tarefas. Isso significa que, ao contrário das aplicações *workflow*, a relação entre as tarefas é definida em tempo de execução e pode depender de vários parâmetros e fatores aleatórios.

O trabalho desenvolvido nesta tese focou na execução eficiente de metaheurísticas paralelas em ambientes de *grid*, pois a grande maioria das metaheurísticas paralelas foi desenvolvida para ambientes de *clusters*. A paralelização de metaheurísticas é uma estratégia cada vez mais comumente adotada para se beneficiar de um maior poder computacional. Entretanto, ainda há poucas implementações de metaheurísticas paralelas para *grids* e, dentre as implementações existentes, a grande maioria adota estratégias inadequadas de paralelização, resultando em algoritmos que não conseguem aproveitar toda a capacidade de processamento fornecida pelo *grid* porque não tratam as características específicas desse novo ambiente.

Nesse contexto, o desafio tratado nesta tese considerou não só as características apresentadas pelas metaheurísticas, mas também o fato de serem aplicações irregulares, em que a demanda computacional nem sempre é conhecida antecipadamente. Além disso, o dinamismo e a heterogeneidade pertencentes a nova plataforma também tiveram que ser tratados de maneira transparente ao usuário.

Para garantir a gerência transparente nas execuções das metaheurísticas paralelas no ambiente de *grid*, optou-se pelo desenvolvimento de metaheurísticas autonômicas [73, 97]. O termo sistema autonômico foi usado primeiramente pela IBM em 2001 [93] para referenciar aplicações que são capazes de se auto-gerenciarem e auto-adaptarem. Essas aplicações têm a habilidade de monitorarem o estado do ambiente em que estão sendo executadas e reagirem às mudanças de estado que podem ocorrer durante as suas execuções [138].

Dessa forma, o desenvolvimento de metaheurísticas paralelas autonômicas proposta nesta tese é uma idéia original, que pretende diminuir a lacuna no desenvolvimento de metaheurísticas eficientes para o ambiente de *grid*.

1.1 Contexto

Devido às características do ambiente e da aplicação citados anteriormente, vários desafios tiveram que ser superados para que as implementações paralelas das metaheurísticas pudessem ser executadas eficientemente no ambiente *grid*. Para isso, a estratégia usada foi a separação de conceitos.

Dessa forma, todos os mecanismos relacionados à gerência do ambiente como tolerância a falhas, escalonamento de tarefas, criação dinâmica e balanceamento de carga foram encapsulados em um *middleware*, chamado SGA EasyGrid [32]. As características específicas da aplicação foram adaptadas ao ambiente *grid* através de uma nova estratégia de paralelismo hierárquica e distribuída, proposta nesta tese.

O SGA EasyGrid é automaticamente embutido na aplicação, que deve ter sido desenvolvida através do MPI (*Message Passing Interface*) [114]. Essa característica garante a portabilidade da aplicação, permitindo que qualquer programa MPI existente possa ser executado em um ambiente *grid*, sem necessidade de nenhuma alteração no código do cliente.

Para isso, o ambiente *grid* deve oferecer os serviços básicos de gerenciamento, tais como autenticação, autorização, descoberta e acesso a recursos, segurança, execução das tarefas remotas e transferência de dados. O ambiente *grid* real, usado em todos os experimentos desta tese, tem adotado o Globus Toolkit [65, 75] para realizar os serviços básicos de gerenciamento do ambiente.

A versão inicial do SGA EasyGrid foi projetada para trabalhar apenas com aplicações *bag-of-tasks*, que caracterizam-se pela total independência entre as tarefas. Para gerenciar aplicações como metaheurísticas, fez-se necessário desenvolver, durante esta tese, uma nova versão do SGA EasyGrid, chamada

SGA metaEasyGrid.

O SGA metaEasyGrid está preparado para tratar aplicações irregulares, que apresentem dinamismo da demanda computacional. O número de tarefas a ser criado não precisa ser conhecido antecipadamente. Além disso, com esse *middleware* é possível desenvolver a aplicação paralela autonômica sem que os detalhes específicos de como a aplicação fará para reagir diante das mudanças do estado do ambiente de execução tenham que ser tratados pelo desenvolvedor.

1.2 Contribuição

O objetivo desta tese foi definir e avaliar uma estratégia original para a paralelização de metaheurísticas, que proporcionasse otimização na execução, em conseqüência de uma melhor utilização da capacidade total de processamento instalada no ambiente *grid*.

Este trabalho propõe uma estratégia alternativa de paralelização que adota listas de armazenamento estruturadas de maneira hierárquica e distribuída, objetivando garantir a comunicação entre os processos durante a execução sem acarretar perda de desempenho, mesmo na comunicação com máquinas geograficamente distantes. A descrição detalhada dessa nova estratégia, assim como os benéficos alcançados com a mesma, são apresentados no decorrer deste trabalho.

Com a estratégia de paralelização proposta, esta tese contribui também com duas implementações paralelas de metaheurísticas para o problema do torneio com viagens espelhado (mTTP, do inglês *mirrored Traveling Tournament Problem*) e o problema da árvore geradora de custo mínimo com restrição de diâmetro.

Outra contribuição foi o desenvolvimento do *middleware* SGA metaEasyGrid. Essa nova versão garante a criação dinâmica de tarefas, sem que o número total de processos sejam conhecidos antes da execução da aplicação. O SGA metaEasyGrid não tem sua utilização restrita ao gerenciamento de metaheurísticas, podendo ser usado para gerenciar a execução de qualquer aplicação paralela que caracterize-se pelo paralelismo irregular.

Por último, a integração do *middleware* SGA metaEasyGrid com a estratégia de paralelização proposta contribui para o desenvolvimento de metaheurísticas paralelas autonômicas de maneira transparente para o desenvolvedor. Isso é possível porque o *middleware* SGA metaEasyGrid é embutido automaticamente na aplicação em tempo de compilação, assim o desenvolvedor não precisa conhecer nenhum detalhe do ambiente de execução *grid*.

1.3 Organização

Além da introdução, esta tese é composta por mais sete capítulos. Inicialmente, no Capítulo 2, são destacadas as principais características de uma metaheurística e os dois problemas NP-difíceis usados nesta tese para efeito de validação. Em seguida, no Capítulo 3, são abordados alguns pontos relevantes a serem considerados na paralelização de metaheurísticas, destacando-se as ferramentas a serem usadas, a plataforma de execução e a estratégia de paralelização adotada no algoritmo.

O Capítulo 4 traz uma revisão bibliográfica do ambiente *grid*. Devido à sua importância para esta tese, dá-se ênfase ao sistema de gerenciamento de aplicação EasyGrid. Logo a seguir, objetivando analisar o comportamento das estratégias tradicionais de paralelização em ambientes distribuídos, quatro implementações paralelas para o mTTP foram implementadas no Capítulo 5.

As maiores contribuições deste trabalho estão descritas no Capítulo 6. Inicialmente, é apresentada a estratégia de paralelização hierárquica distribuída proposta para metaheurísticas em ambientes computacionais *grid*. Em seguida, são descritos os detalhes do *middleware* metaEasyGrid proposto para gerenciar transparentemente a execução das aplicações paralelas no ambiente. A fim de validar a integração entre a estratégia hierárquica distribuída e o metaEasyGrid, as próximas seções deste capítulo apresentam as implementações paralelas para o mTTP e o problema da árvore geradora de custo mínimo com restrição de diâmetros.

O Capítulo 7 traz uma análise detalhada dos resultados computacionais obtidos pelas implementações paralelas desenvolvidas com a estratégia de paralelização proposta no capítulo anterior.

Para finalizar, o último capítulo destaca as vantagens oferecidas pela nova estratégia de paralelização para metaheurísticas. Em seguida, são apresentados trabalhos futuros a serem desenvolvidos como extensões ou conseqüências desta tese.