Departamento de Ciência da Computação

Instituto de Matemática e Estatística

Universidade de São Paulo

Escalonador Dinâmico e Inter-aglomerado para Aplicações de Grades Oportunistas

Projeto de pesquisa apresentado como parte da documentação requerida para obtenção de bolsa de doutorado FAPESP

Candidato: Vinicius Gama Pinheiro - vinicius@ime.usp.br

Responsável: Prof. Alfredo Goldman - gold@ime.usp.br

São Paulo, Outubro de 2008

Resumo

Grades oportunistas são grades computacionais que aproveitam o poder computacional ocioso de recursos não-dedicados para executar aplicações distribuídas e de alto desempenho. O escalonamento de aplicações nesse tipo de grade é uma área de pesquisa promissora e ainda repleta de desafios a serem transpostos. Os desafios mais comuns são relacionados à falta de informações sobre as aplicações submetidas e ao uso de recursos heterogêneos e não-dedicados. Em ambientes com essas características, o escalonamento precisa ser dinâmico e adaptativo, isto é, os recursos devem ser alocados no momento da criação das tarefas, possibilitando que somente os recursos mais adequados no momento sejam escolhidos. Nessas grades, esses recursos ficam espalhados em diversos domínios administrativos locais, sendo compartilhados por usuários locais que devem ter prioridade sobre o uso dos mesmos. Neste trabalho, propomos a implementação de um escalonador dinâmico e inter-aglomerado para aplicações de grade oportunista. Esse escalonador deve ser modular e adaptável, permitindo diferentes heurísticas de escalonamento como *First In, First Out*, filas de processos com atribuição de prioridades, casamento de tarefas e reserva de máquinas, entre outras. O balanceamento ativo de carga entre recursos de aglomerados distintos também será um requisito obrigatório. As informações utilizadas pelos algoritmos de escalonamento serão obtidas através de uma interface com o serviço de monitoramento dos recursos da grade. Esse serviço, ainda com funcionalidades limitadas, está sendo desenvolvido por outros pesquisadores do nosso grupo. O nosso objetivo, ao final do projeto, é prover uma versão utilizável desse escalonador, que seja sensível às flutuações na disponibilidade de recursos como processadores, memórias, canais de comunicação e armazenamento em disco, e que respeite as necessidades dos donos dos recursos.

Sumário

1 Introdução

Grades oportunistas são grades computacionais que aproveitam o poder computacional ocioso de recursos não-dedicados para executar aplicações distribuídas e de alto desempenho. Esses ambientes são altamente dinâmicos, com frequente entrada e saída de nós, e devem compartilhar *hardwares* e *softwares* heterogêneos [6, 12]. O escalonamento de aplicações em ambientes de grade oportunista consiste essencialmente em determinar quando, e em qual recurso, cada tarefa deve ser executada. Em ambientes oportunistas, existe a preocupação adicional de manter a transparência da grade sob o ponto de vista dos usuários locais e donos dos recursos, que não devem sofrer com perdas de desempenho.

Como pode ser observado nos diversos trabalhos publicados sobre este assunto [5, 1, 2, 8], o escalonamento de aplicações paralelas em grades computacionais geralmente é dividido em duas etapas. A primeira etapa consiste em selecionar, a partir do conjunto de tarefas submetidas, qual ou quais serão escalonadas. Na segunda etapa, os recursos para a execução dessas tarefas devem ser alocados. Diferentes funcionalidades e mecanismos podem ser utilizados em ambas as etapas. A escolha das tarefas a serem executadas podem adotar diversas políticas como FIFO (*First-In First-Out*), minimizar o tempo de espera para execução, classificar as tarefas em filas de prioridade, priorizar as tarefas pelo tamanho (maior, menor) ou pelo tipo (paramétricas, BSP [21], MPI [13]), etc. A escolha dos recursos também podem seguir diversas estratégias como minimizar o tempo de execução, maximizar o *throughput*, reservar recursos para períodos futuros, efetuar a divisão justa de recursos entre as tarefas (mais conhecido como *fairness*), entre outros [18]. O perfil dos usuários da grade, o modelo de arquitetura adotado, os tipos de aplicações contempladas e as variações no ambiente de execução são alguns dos fatores que devem determinar como o escalonador deve se comportar. Diante de tantas possibilidades, é desejável que o escalonador seja modular e adaptável, isto é, que o seu comportamento possa ser alterado facilmente e que novos comportamentos possam ser definidos e avaliados, tanto para estudo quanto para uso.

Neste trabalho, propomos a implementação de um escalonador inter-aglomerado para grades oportunistas modular e adaptável no qual seja possível a definição de diversas políticas de escalonamento. Esse escalonador terá o respaldo de um mecanismo que analisa o padrão de uso dos recursos. Através desse mecanismo serão feitas previsões dos intervalos de ociosidade dos recursos para, então, realizar o casamento ou agendamento desses com as tarefas submetidas.

1.1 Justificativa

Grades oportunistas aproveitam a ociosidade dos recursos para executar aplicações paralelas. Esses ambientes geralmente implementam algum mecanismo que detecta quando um recurso está ocioso, atribuindo-lhe uma ou mais tarefas para execução. Quando o dono do recurso requisita o uso exclusivo da sua máquina, isso possivelmente acarreta a interrupção dessas tarefas. Esse procedimento é conhecido como computação de melhor esforço (*best effort computing*) [5]. Uma das principais preocupações do modelo de computação oportunista, portanto, é evitar degradação de desempenho para os donos das máquinas compartilhadas. Por outro lado, com a computação de melhor esforço, as tarefas submetidas pelo usuário da grade ficam sujeitas a interrupções, sacrificando o tempo de execução das mesmas.

Atualmente, nosso projeto conta com um módulo que monitora os recursos de uma grade oportunista. Esse módulo, denominado LUPA (*Local User Pattern Analyzer*) ainda opera com funcionalidades limitadas (ainda não é possível ter uma visão centralizada de todos os recursos de um aglomerado, por exemplo), mas através dele é possível obter de cada recurso informações relativas ao seu padrão de uso, como uso de processador e memória. Neste projeto, propomos utilizar esse módulo através de uma interface com o nosso escalonador e, através da análise dos padrões de uso coletados, realizar o casamento entre recursos e tarefas. Dessa forma, as tarefas poderão ser executadas nos recursos mais adequados, reduzindo os riscos de interrupções.

1.2 Objetivos

O objetivo principal deste projeto é implementar um escalonador para grades oportunistas que seja modular, podendo esse, inclusive, ser utilizado como plataforma científica na investigação de algoritmos de escalonamento e análise de resultados.

Dentre os objetivos específicos, podemos citar:

1. *Agendamento de Tarefas*: O usuário poderá definir um período futuro no qual a sua aplicação deverá ser executada;

2. *Escalonamento Adaptável*: Disponibilizar diferentes algoritmos de escalonamento para a execução das aplicações, entre eles: FIFO (*First In, First Out*), FF (*First Fit*), BF (*Best Fit*), atribuição de prioridades, etc;

3. *Balanceamento de Carga*: Através de interações com o serviço de monitoramento, as tarefas poderão ser alocadas nos recursos menos ocupados;

4. *Escalonamento Inter-Aglomerado*: As aplicações poderão ser alocadas em recursos de diferentes aglomerados.

2 Trabalhos Correlatos

Alguns escalonadores encontrados na literatura possuem características semelhantes às do escalonador que propomos. Alguns deles não são voltados para ambientes de grades oportunistas. Outros adotam algoritmos de escalonamento menos complexos e limitam-se à execução de aplicações embaraçosamente paralelas. Nenhum deles utiliza informações sobre os padrões de uso dos recursos.

A seguir, serão descritos alguns desses escalonadores, destacando-se suas virtudes, fraquezas e características que os diferem do que é proposto neste projeto.

2.1 Task Scheduling Testbed

Trabalhos recentes de Boeres e Rebello [2, 3] focam na implementação de um ferramenta chamada *Task Scheduling Testbed* através da qual é possível testar diferentes algoritmos de escalonamento. Essa ferramenta utiliza o GridSim [4] para simular um ambiente de grade e oferece uma interface gráfica para a submissão de aplicações sintéticas, em forma de DAG's (*Directed Cyclic Graphs*). Através dessa interface é possível definir o comportamento do algoritmo de escalonamento através da configuração de dois escalonadores: um estático e um dinâmico. O estático atua antes da submissão, mapeando tarefas e recursos disponíveis de acordo com filas de prioridades e informações locais sobre os recursos da grade. O escalonador dinâmico consiste na realocação dos recursos através da reexecução do escalonador estático mas, nesse estágio, pode ser utilizada uma outra fila de prioridade.

Através da ferramenta também e possível submeter e monitorar aplicações do tipo MPI em grades computacionais baseadas no *Globus Toolkit*. Aplicações MPI podem ser convertidas previamente para o formato de DAG's, permitindo que diferentes estratégias de escalonamento sejam observadas antes que a submissão seja efetivada.

2.2 PBS/OpenPBS

O Portable Batch Scheduler (PBS) [14, 7] foi inicialmente desenvolvido para computadores paralelos de memória compartilhada de arquitetura SMP (Shared Memory Multiprocessor). Pode ser configurado para funcionar em diversas arquiteturas desde aglomerados de estações de trabalho heterogêneas a supercomputadores.

O suporte para aglomerados foi adicionado posteriormente, mas ainda não contém funcionalidades importantes como, por exemplo, a submissão simultânea de tarefas em diversas máquinas. No PBS, a tarefa inicial de uma aplicação, incluindo seus scripts de gerenciamento, são executados inteiramente em um único nó. O escalonamento é realizado através de uma algoritmo que mescla FIFO com

uma regra de *First Fit*, ou seja, ele percorre a fila de tarefas e escalona a primeira que possa ser encaixada nos intervalos disponíveis dos recursos. Para evitar que tarefas longas sejam postergadas indefinidamente, o PBS possui um mecanismo que é disparado assim que o tempo de espera de uma tarefa tenha ultrapassado um limite de tempo estabelecido (o padrão é 24 horas). Esse mecanismo pára de escalonar novas tarefas até que a tarefa postergada seja iniciada. Vale ressaltar que, durante a execução desse mecanismo, mesmo que um recurso não possa executar a tarefa postergada, ele não será alocado para outras tarefas.

A despeito de sua simplicidade, o escalonador do PBS é modular e pode ser substituído por outros escalonadores, na forma de *plug-ins* que podem ser adicionados ao sistema. Uma versão modificada do PBS utiliza o Maui Scheduler. O Maui [1] opta por escalonar primeiramente as tarefas de maior prioridade (invariavelmente as tarefas maiores) e, depois, procura escalonar as tarefas de menor prioridade nos intervalos de tempo ainda disponíveis. Apesar das melhorias propiciadas por este *plugin*, o PBS/Maui ainda não possui características direcionadas aos ambientes oportunistas, como preempção de tarefas quando os recursos são requisitados pelos seus respectivos donos e alocação de recursos em aglomerados vizinhos.

2.3 SLURM

SLURM (Simple Linux Utility for Resource Management) [22] é um escalonador de código aberto para aglomerados Linux de grande e pequeno porte. Com o foco na simplicidade, esse escalonador é altamente escalável, capaz de executar aplicações paralelas em aglomerados com mais de mil nós. Através do SLURM, é possível definir requisitos para a execução de tarefas e ele também fornece ferramentas para monitoramento e cancelamento de tarefas. O seu escalonador, contudo, é bastante simples, adotando uma política de FIFO (First-In Fist-Out).

A despeito da sua simplicidade e facilidade de uso, o SLURM não fornece suporte a computação

em grade e, por ser desenvolvido somente para aglomerados Linux (com fácil adaptação para sistemas Unix), não pode ser utilizado em ambientes heterogêneos. o SLURM não faz coleta dos padrões de uso dos recursos e as informações sobre o estado dos nós da rede não são disponibilizados ao usuário, já que são utilizados somente por processos internos. Essas funções, bem como heurísticas mais avançadas de escalonamento (somente FIFO esta implementada) devem ser configuradas à parte, pois não fazem parte do sistema.

Os trabalhos submetidos para execução no SLURM são ordenados por uma fila de prioridades. Cada trabalho pode ser composto por uma ou mais tarefas. Quando um trabalho é escolhido para ser executado o SLURM aloca o conjunto de recursos necessários dentro de um aglomerado. Contudo, quando essa alocação falha, esse conjunto de recursos não é utilizado para escalonar trabalhos de menor prioridade.

2.4 Condor

Um dos sistemas pioneiros na área da computação oportunista, o projeto Condor[15, 11, 20], lançado em 1984, alavancou o interesse acadêmico na busca de soluções que permita o uso de ciclos ocioso de estações de trabalho para a execução de aplicações paralelas de alto processamento.

O casamento entre tarefas e recursos é definido através de uma linguagem própria denominada ClassAds, que flexibiliza a adoção de diferentes políticas de escalonamento. Mecanismos de tolerância a falhas (*checkpointing* e migração) e de segurança (*sandbox*) também estão presentes neste sistema. No Condor, o sistema detecta quando um usuário solicita o uso da sua máquina (através de interações com mouse e teclado) e pode migrar as tarefas que executavam nela para outro recurso.

A união entre o Condor e o projeto Globus [9] proporcionou ao Condor a infra-estrutura necessária para sua adaptação aos ambientes de grades. O Globus fornece os protocolos para comunicação segura entre os diversos aglomerados da grade enquanto o Condor cuida dos serviços de submissão, escalo-

namento, recuperação de falhas e criação de um ambiente de execução amigável. Cada aglomerado possui um gerenciador central denominado CM (Central Manager) que administra os outros nós do aglomerado e verifica sua disponibilidade, além de executar o casamento de recursos a partir das informações obtidas. Cada aglomerado possui também um ou mais nós que agem como Gateways. Os Gateways mantém informações sobre os seus Gateways vizinhos e informam ao CM do seu aglomerado sobre a disponibilidade dos recursos nos aglomerados adjacentes.

Nossa proposta difere do Condor já que o escalonador proposto é de propósito mais geral ao passo que o Condor, por contar com checkpointing e migração de tarefas, é mais adequado para aplicações embaraçosamente paralelas (e.g. saco de tarefas). Além disso o escalonador proposto conta com um módulo que analisa o padrão de uso dos recursos, característica ausente no sistema Condor.

2.5 OurGrid

Desenvolvido pela Universidade de Campina Grande, com o apoio da Hewlett Packard, o OurGrid [6] é o projeto de uma grade que permite que laboratórios compartilhem os ciclos ociosos de seus recursos através de um rede de favores, que promove a justa divisão do tempo de processamento entre as entidades participantes da grade. Com o objetivo de incentivar a participação do maior número possível de laboratórios em torno de uma comunidade segura e escalável, o OurGrid se baseia em uma rede ponto a ponto, além de contar com um mecanismo de sandbox baseado na máquina virtual Xen. Por outro lado, este sistema lida somente com a execução de aplicações paralelas embaraçosamente paralelas (e.g. saco de tarefas), sendo que as tarefas inicial e final rodam necessariamente na máquina do usuário.

O OurGrid possui duas opções para escalonamento: WQR e StorageAffinity [19]. O WQR (*Work-Queue with Replication*) é um escalonador simples que seleciona as tarefas segundo uma política FIFO. Assim que todas as tarefas são enviadas, o WQR escolhe uma aleatoriamente, constrói uma réplica e

escalona. Esse procedimento é repetido até que não hajam mais recursos disponíveis. Como o WQR não utiliza qualquer informação acerca das aplicações ou dos recursos, a replicação funciona como um mecanismo que procura compensar alocações más sucedidas (e.g. escalonar tarefas em recursos lentos ou sobrecarregados). Isso faz com que o WQR consuma mais recursos do que os escalonadores que utilizam informações sobre a disponibilidade dos recursos. Cientes deste problema, os desenvolvedores lançaram a segunda versão do OurGrid com uma nova opção de escalonamento: o StorageAffinity. Esse escalonador mantém informações sobre a quantidade de dados que os nós contém sobre uma determinada aplicação. Dessa forma, sempre que uma decisão de escalonamento precisa ser feita, o StorageAffinity escolhe o recurso que já contém a maior quantidade de dados necessários para o processamento. Essa abordagem é mais adequada para as aplicações do tipo saco de tarefas que processam grandes quantidades de dados, já que o tempo de transferência dos dados para as máquinas que irão processá-los representam uma sobrecarga considerável no tempo total de execução das aplicações.

A vantagem do Ourgrid é proporcionar um ambiente de grade onde aglomerados podem compartilhar recursos de forma segura e confiável, através de um mecanismo que mensura o tempo de processamento disponível para um aglomerado a partir da quantidade de recursos que ele oferece para a grade. Contudo, esse sistema não oferece suporte para a execução de aplicações que trocam mensagens entre as tarefas, como BSP e MPI. Além disso, ao optar pela simplicidade, o OurGrid não faz análise de uso dos recursos e não dispõe de heurísticas de escalonamento mais complexas.

2.6 OAR

OAR [5] é um escalonador em batch para aglomerados de grande porte. Essa escalonador utiliza ferramentas de alto nível como linguagem de programação Perl e banco de dados MySql para realizar casamento entre tarefas e recursos através de consultas SQL a um banco de dados centralizado. Ele é modular e provê heurísticas de escalonamento baseadas em filas de prioridade, agendamento e

backfilling.

O OAR investe na simplicidade e nos benefícios da linguagem SQL. Todos os dados internos sobre aplicações e recursos são armazenados em um banco de dados e o acesso a esse banco é o único meio de comunicação entre os módulos. O casamento entre recursos e o armazenamento e consulta de logs do sistema são realizados através de chamadas SQL. O controle de escalonamento e de execução das tarefas são realizados por scripts Perl, organizados em módulos. O OAR possui um módulo central cuja finalidade é gerenciar a execução das atividades implementadas nos sub-módulos (escalonamento, execução, monitoramento). Os sub-módulos notificam o módulo central sempre que realizam uma atualização no banco de dados. Essa abordagem é utilizada com a justificativa de tornar o sistema mais robusto, contudo, além de representar um ponto único de falha para o aglomerado, faz com que o sistema fique altamente dependente do desempenho proporcionado pelo sistema de banco de dados utilizado.

Para o serviço de monitoramento da grade, o OAR utiliza a ferramenta Taktuk. O Taktuk [16] é originalmente utilizado para fazer instalações remotas de aplicações paralelas em grandes aglomerados mas, dentro do OAR, essa ferramenta é utilizada para realizar tarefas administrativas nos nós dos aglomerados através de um serviço de execução remota baseado em ssh. Através do Taktuk, nós (potencialmente) falhos podem ser detectados pelo tempo de resposta dos mesmos, respeitando-se um tempo limite (*time out*) que pode ser modificado pelo administrador da grade. Todavia, apesar da sua versatilidade, o Taktuk não faz análise de padrões de uso dos recursos.

Através de uma extensão, o OAR prove suporte para computação em grade, sendo que o gerenciamento das tarefas paralelas adota a política do melhor esforço, isto é, assim que uma máquina é requisitada pelo seu dono, todas as tarefas que estavam sendo executadas nessa máquina, e as que dependem destas, são interrompidas. Um dos objetivos do escalonador que propomos é justamente evitar que isso ocorra através da obtenção de informações sobre o padrão de uso dos recursos. Através

desse serviço, serão escolhidos os recursos mais adequados para a execução das tarefas, isto é, aqueles que provavelmente (pela análise do seu histórico de uso) estarão livres pelo período de tempo necessário para a execução.

3 Metodologia e Resultados Esperados

Este projeto será realizado com os recursos do projeto InteGrade [12]. O projeto InteGrade mantém um conjunto de aglomerados, compostos por máquinas de professores e estudantes, dispersas em diversos laboratórios nas dependências do Instituto de Matemática e Estatística da Universidade São Paulo. Alguns desses aglomerados já executam o *middleware* de grade oportunista Integrade, mas a versão atual dispõe somente de um escalonador simplificado que executa um algoritmo Round-Robin para selecionar os recursos. Nesta seção, serão descritos a arquitetura do InteGrade e as modificações propostas para esse *middleware*, dentre outras atividades.

3.1 Arquitetura do InteGrade

O projeto InteGrade consiste no desenvolvimento de um *middleware* de grade que aproveita o poder computacional ocioso das estações de trabalho. Este projeto é mantido pelo Instituto de Matemática e Estatística da Universidade São Paulo (IME/USP), em conjunto com diversas instituições de vários estados: Departamento de Computação e Estatística da Universidade Federal do Mato Grosso do Sul (DCT/UFMS), Departamento de Informática da Pontifícia Universidade Católica do Rio de Janeiro (DI/PUC-Rio), Instituto de Informática da Universidade Federal da Goiás (INF/UFG) e o Departamento de Informática da Universidade Federal do Maranhão (DEINF/UFMA).

O *middleware* InteGrade é baseado em CORBA, um padrão para sistemas de objetos distribuídos. O serviços de nomeação e comunicação do InteGrade são exportados como interfaces CORBA IDL que são acessíveis por uma grande variedade de linguagens de programação e sistemas operacionais.

A arquitetura do InteGrade é organizada através de aglomerados em uma estrutura hierárquica. Dentro de um aglomerado cada nó pode assumir diferentes papéis, definidos a partir dos componentes que este hospeda. O *Cluster Manager* é o nó responsável por gerenciar o aglomerado e realizar a comunicação com outros aglomerados. Um nó do tipo *Resource Provider* exporta parte dos seus recursos, deixando-os disponíveis para os usuários da grade. Um nó do tipo *User Node* é aquele que pertence a um usuário da grade que submete aplicações ao ambiente de execução. Na figura 1, podemos ver tanto a estrutura interna dos aglomerados quando a estrutura em árvore que define a hierarquia inter-aglomerados, na qual cada *Cluster Manager* possui um canal de comunicação com outro *Cluster Manager* "pai", à exceção do que está no topo da árvore.

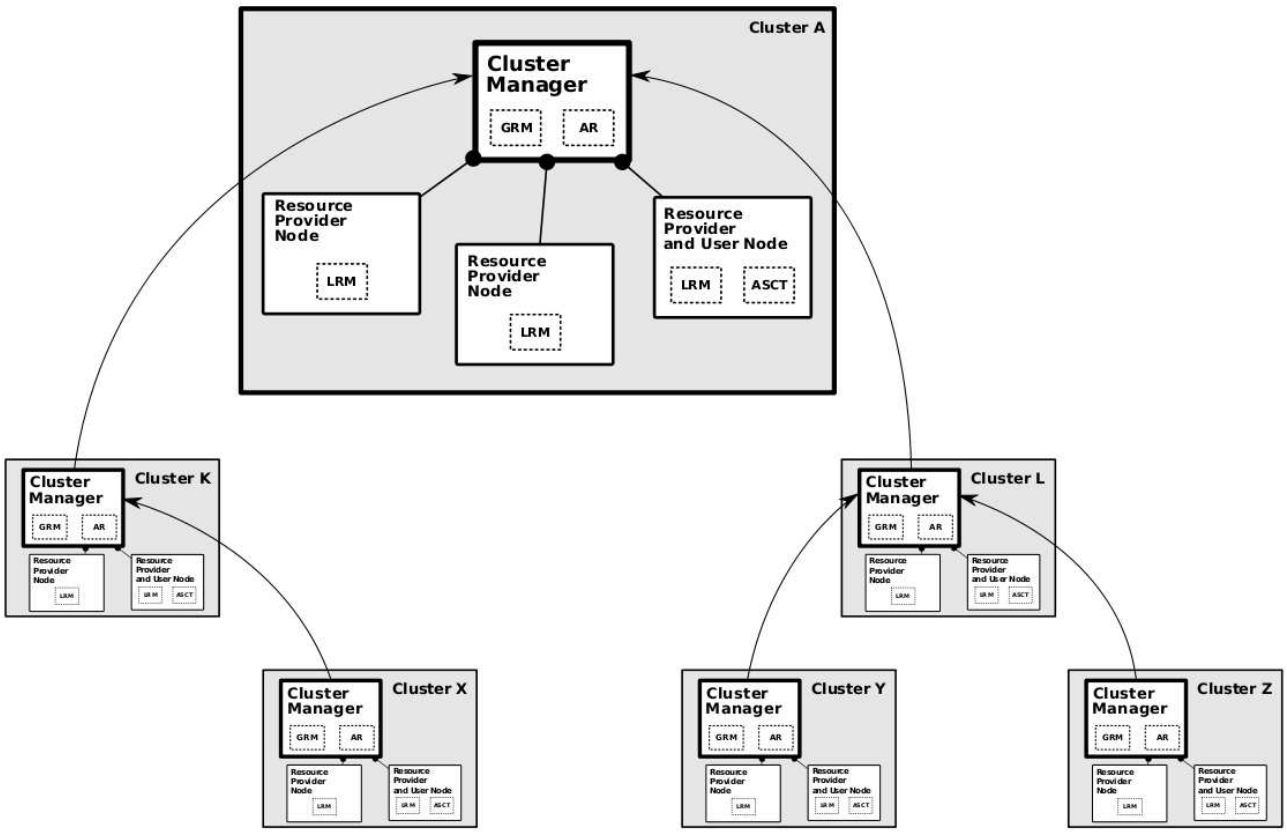

Figura 1: Arquitetura do InteGrade

O *GRM (Global Resource Manager)* é o componente principal da grade e é executado em nós do tipo Cluster Manager. Esse componente mantém uma lista sobre os *LRM*s ativos e pode escalonar

aplicações entre estes. O *LRM (Local Resource Manager)* é executado em cada nó do tipo *Resource Provider* e carrega todo o ambiente necessário para a execução das aplicações. O *AR (Application Repository)* provê um repositório centralizado para o armazenamento dos binários das aplicações submetidos à grade. Por fim, o *ASCT (Application Submission and Control Tool)* é executado nos nós dos usuários (tipo *User Node*) e fornece uma interface pela qual o usuário pode submeter aplicações à grade e visualizar os resultados finais. Além desses, em breve, será incorporado o componente *LUPA (Local Usage Pattern Analyzer)*, que será executado junto ao LRM para coletar informações locais sobre utilização de memória, CPU e disco. O InteGrade ainda contém outros componentes para armazenamento distribuído de dados, mas o funcionamento destes está fora do escopo deste trabalho.

3.2 Agendamento de Tarefas

O agendamento de tarefas para submissão futura será implementado através da inserção dessa funcionalidade no módulo de submissão do *middleware* InteGrade, o ASCT. Esse módulo já dispõe de uma interface para a submissão dos diversos modelos de aplicações contempladas pelo InteGrade: regulares, paramétricas, BSP (*Bulk Synchronous Parallel*) e MPI (*Message Passing Interface*). A modificação proposta consiste em adicionar em sua interface as opções de dia e hora para a execução da aplicação. Quando o usuário fizer a submissão de uma aplicação, o gerenciador do aglomerado (no InteGrade, representado pelo GRM) armazenará o binário da aplicação no repositório e as informações de execução em uma tabela de um banco de dados simplificado.

3.3 Escalonamento de Aplicações Inter-aglomerado

Existem duas abordagens para realizar o escalonamento inter-aglomerado: entre aplicações e entre tarefas. Essas duas abordagens diferenciam-se pela granularidade, isto é, pela unidade de trabalho que é escalonada entre os aglomerados. A primeira é mais simples pois requer apenas que a requisição de execução seja encaminhada de um aglomerado para outro. Nessa abordagem, todas as tarefas de uma

aplicação sempre estarão confinadas em apenas um aglomerado. A segunda abordagem consiste em escalonar as tarefas de uma mesma aplicação em recursos de aglomerados distintos. Essa abordagem é mais complexa visto que, para implementá-la, é preciso lidar com problemas de comunicação que podem ocorrer devido à imprevisibilidade temporal da rede que conecta os aglomerados da grade. Processos que se comunicam entre si, como os encontrados nos modelo BSP e MPI, poderiam sofrer lentidão com os eventuais atrasos nos canais de comunicação. Dessa forma, pela sua simplicidade, em nosso projeto adotaremos inicialmente o escalonamento inter-aglomerado entre aplicações.

Para fazer com que as aplicações possam ser encaminhadas de um aglomerado para outro, será necessário alterar o funcionamento do gerenciador do aglomerado. Como mencionado anteriormente, no InteGrade, esse papel é desempenhado pelo GRM. Ele possui a função de se comunicar com seus aglomerados adjacentes, que consistem em um (ou nenhum) aglomerado pai e vários aglomerados filhos. Na implementação atual, quando não há recursos suficientes no aglomerado para que uma aplicação seja executada, a submissão é recusada. No escalonador que propomos, a execução poderia ser agendada para um momento futuro ou então repassada para um dos aglomerados adjacentes.

Atualmente, o serviço de monitoramento da grade, função exercida pelo módulo LUPA, opera individualmente em cada um dos recursos. Futuramente, esse módulo será estendido ao gerenciador do aglomerado para que, através de uma única consulta ao GRM, seja possível obter informações sobre todos os recursos do aglomerado. Nosso projeto almeja implementar uma interface para esse módulo de modo que cada aglomerado realize consultas aos aglomerados adjacentes. O objetivo dessa interface, portanto, é fazer com que cada aglomerado enxergue o seu vizinho como um conjunto de recursos. Este seria o primeiro passo para que recursos em aglomerados vizinhos também sejam utilizados nas decisões de escalonamento.

3.4 Módulo de Escalonamento e Adição de Algoritmos

Como mencionado na seção 1, no momento da alocação das tarefas, diversos algoritmos podem ser utilizados. Esses algoritmos podem utilizar informações fornecidas pelo LUPA para realizar casamento entre recursos e um conjunto de tarefas. Neste projeto, planejamos implementar alguns desses algoritmos como *First Fit*, *Best Fit* e FIFO. Mas, com o objetivo de flexibilizar a alocação de recursos, novos algoritmos poderão ser definidos e adicionados à lista dos pré-existentes. Para viabilizar este passo, o algoritmo de escalonamento do GRM será modularizado para que, então, outros algoritmos sejam adicionados como opções.

Durante todas as intervenções no código do InteGrade, utilizaremos o ambiente de desenvolvimento integrado Eclipse como ferramenta principal. Como ferramentas auxiliares, serão utilizados alguns diagramas UML (de sequência, de interação e de classe), simuladores de eventos discretos (e.g. GridSim [4]) e outros aplicativos (i.e. Apache Ant [10], JConsole [17], etc). Para realizar os testes práticos das diversas modificações propostas (i.e. módulos e algoritmos de escalonamento), utilizaremos os recursos e laboratórios do projeto, em especial, as máquinas do Laboratório de Computação Paralela e Distribuída do Instituto de Matemática e Estatística da Universidade São Paulo. Nesses recursos, as versões modificadas do *middleware* poderão ser instaladas e avaliadas em um ambiente real.

3.5 Resultados Esperados

Ao final do projeto, espera-se que o escalonador proposto esteja totalmente integrado ao *middleware* do projeto InteGrade, sendo possível, portanto, escalonar e agendar a execução de aplicações entre os diversos aglomerados que compõe o ambiente de grade. Através de testes e da avaliação dos resultados, pretende-se configurar o escalonador com um comportamento padrão que seja mais adequado para a maioria dos casos de submissão.

4 Plano de Trabalho e Cronograma

	Anos e Semestres					
	2009		**2010**		**2011**	
Atividades	1^o	2^o	1^o	2^o	1^o	2^o
Levantamento Bibliográfico	x					
Agendamento de Aplicações		x				
Escalonamento Inter-Aglomerado		x	x			
Algoritmos de Escalonamento				x		
Análise de Desempenho					x	
Redação da Tese				x	x	x
Qualificação				x		
Defesa						x

Tabela 1: Cronograma de atividades

1. *Levantamento Bibliográfico*: Leitura de artigos e trabalhos correlatos;

2. *Agendamento de Aplicações*: Modificações no módulo de submissão para o agendamento de aplicações;

3. *Escalonamento Inter-Aglomerado*: Consiste em implementar a interface entre os gerenciadores dos aglomerados. Nesse estágio, o gerenciador do aglomerado já deve ser capaz de se comunicar com o módulo de monitoramento dos recursos da grade;

4. *Algoritmos de Escalonamento*: Modularização do mecanismo de escalonamento e implementação dos algoritmos de escalonamento;

5. *Análise de Desempenho*: Testes, simulações e interpretação dos resultados;

6. *Redação da Tese*: Escrita da tese de doutorado;

7. *Qualificação*: Exame de qualificação;

8. *Defesa*: Defesa da tese de doutorado;

Paralelamente à essas atividades, o pesquisador também se dedicará a outras atividades do programa de doutorado, como disciplinas obrigatórias, seminários e exames admissionais. O pesquisador também almeja a publicação de artigos científicos em eventos de caráter nacional e internacional.

Referências

[1] Brett Bode, David M. Halstead, Ricky Kendall, Zhou Lei, and David Jackson. The portable batch scheduler and the maui scheduler on linux clusters. In *ALS'00: Proceedings of the 4th annual Linux Showcase & Conference*, pages 27–27, Berkeley, CA, USA, 2000. USENIX Association.

[2] Cristina Boeres, José Viterbo Filho, and Vinod E. F. Rebello. A cluster-based strategy for scheduling task on heterogeneous processors. *Computer Architecture and High Performance Computing, Symposium on*, 0:214–221, 2004.

[3] Cristina Boeres and Vinod E. F. Rebello. Easygrid: towards a framework for the automatic grid enabling of legacy mpi applications: Research articles. *Concurr. Comput. : Pract. Exper.*, 16(5):425–432, 2004.

[4] Rajkumar Buyya and M. Manzur Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *CoRR*, cs.DC/0203019, 2002.

[5] Nicolas Capit, Georges Da Costa, Yiannis Georgiou, Guillaume Huard, Cyrille Martin, Grégory Mounie, Pierre Neyron, and Olivier Richard. A batch scheduler with high level components. In *CCGRID '05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2*, pages 776–783, Washington, DC, USA, 2005. IEEE Computer Society.

[6] Waldredo Cirne, Francisco Brasileiro, Nazareno Andrade, Lauro B. Costa, Alisson Andrade, Reynaldo Novaes, and Miranda Mowbray. Labs of the world, unite!!! *Journal of Grid Computing*, 4(3):225–246, September 2006.

[7] Altair Engineering. Portable batch scheduler. Last accessed on 26 Out, 2008.

[8] Gilles Fedak, Cecile Germain, Vincent Neri, and Franck Cappello. Xtremweb: A generic global computing system. In *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, page 582, Washington, DC, USA, 2001. IEEE Computer Society.

[9] Ian Foster. Globus toolkit version 4: Software for service-oriented systems. In *NPC 2005: IFIP International Conf. on Network and Parallel Computing*, pages 2–13, Beijing, China, 2005.

[10] The Apache Software Foundation. The apache ant software. http://ant.apache.org. Last accessed on 27 Out, 2008.

[11] James Frey, Todd Tannenbaum, Miron Livny, Ian Foster, and Steven Tuecke. Condor-g: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3), 2002.

[12] Andrei Goldchleger, Fabio Kon, Alfredo Goldman, Marcelo Finger, and Germano Capistrano Bezerra. Integrade: object-oriented grid middleware leveraging the idle computing power of desktop machines. *Concurrency - Practice and Experience*, 16(5):449–459, 2004.

[13] William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir. *MPI: The Complete Reference*. MIT Press, 2th edition, 1998.

[14] Robert L. Henderson. Job scheduling under the portable batch system. In *IPPS '95: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 279–294, London, UK, 1995. Springer-Verlag.

[15] Michael Litzkow, Miron Livny, and Matt Mutka. Condor - A hunter of idle workstations. In *Proc. of the 8th Int. Conf. of Distributed Computing Systems (ICDCS)*, pages 104–111, June 1988.

[16] Cyrille Martin and Olivier Richard. Algorithme de vol de travail appliqué au déploiement d'applications parallèles. In *Soumis RenPar'15*, 2003.

[17] Sun Microsystems. Using jconsole to monitor applications. http://java.sun.com/developer/technicalArticles/J2SE/jconsole.htn Last accessed on 27 Out, 2008.

[18] Grégory Mounié Pierre-François Dutot, Lionel Eyraud and Denis Trystram. Scheduling on large scale distributed platforms: From models to implementations. *International Journal of Foundations of Computer Science (IJFCS)*, 16(2):217–237, 2005.

[19] Elizeu Santos-Neto, Walfredo Cirne, Francisco Brasileiro, and Aliando Lima. Exploiting replication and data reuse to efficiently schedule data-intensive applications on grids. In *Job Scheduling Strategies for Parallel Processing*, LNCS Series. Springer Berlin/Heidelberg, May 2005.

[20] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: The condor experience. concurrency and computation: Practice and experience. 17:2–4, 2005.

[21] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, 1990.

[22] Andy B. Yoo, Morris A. Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *Job Scheduling Strategies for Parallel Processing*, pages 44–60. Springer Berlin / Heidelberg, 2003.