Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo

Estudo comparativo de escalonadores de aplicações para grades computacionais

Projeto de pesquisa de mestrado

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

São Paulo, Janeiro de 2009

Resumo

Atualmente existem várias grades computacionais em funcionamento baseadas em diferentes middlewares para o gerenciamento de recursos. Cada um deles apresenta estratégias específicas para o escalonamento de aplicações nos recursos disponíveis.

O escalonamento pode considerar desde um ambiente mais simples, com recursos localizados em uma única rede local, no caso um aglomerado. Mas ambientes mais complexos podem ser considerados. Dois tipos de contextos estão cada vez mais comuns no escalonamento de aplicações em grades, o escalonamento inter-aglomerado e o escalonamento oportunista. No caso inter-aglomerado, permite-se que as aplicações sejam alocadas em ambientes compostos por diferentes domínios, possivelmente permitindo sua execução simultânea em mais de um domínio.

No escalonamento em ambiente oportunista utilizam-se recursos não dedicados para executar aplicações. 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, vamos estudar os diferentes escalonadores atualmente utilizados. Em seguida vamos implementá-los em simuladores de grades, assim como elaborar uma análise comparativa dos mesmos em diferentes cenários.

Sumário

1 Introdução

Grades computacionais compreendem uma complexa infra-estrutura composta por soluções integradas de hardware e software que permitem o compartilhamento de recursos distribuídos sob a responsabilidade de instituições distintas [12]. Esses ambientes são alternativas atraentes para a execução de aplicações paralelas ou distribuídas que demandam alto poder computacional, tais como mineração de dados, previsão do tempo, biologia computacional, física de partículas, processamento de imagens médicas, entre outras [1]. Essas aplicações paralelas são composta por diversas tarefas que, a depender do modelo de aplicação, podem se comunicar durante a fase de execução.

Existem diversos tipos de grades computacionais, classificadas de acordo com a sua finalidade. As grades de dados (*data grids*) são utilizadas para a pesquisa e armazenamento distribuído de grandes quantidades de dados. As grades de serviço (*service grids*) tem como objetivo a interoperabilidade e são ambientes propícios para o compartilhamento sob demanda de serviços Web entre diversas instituições. Finalmente, as grades oportunistas fazem uso da capacidade computacional ociosa de recursos não-dedicados, como as estações de trabalho encontradas em laboratórios científicos, intranets e pequenas redes locais [22, 12].

Neste trabalho nos concentraremos no estudo do escalonamento de aplicações [1] em grades computacionais. Neste contexto as principais dificuldades estão ligadas ao uso de recursos em domínios diferentes, e ao uso de recursos não dedicados.

Existem duas abordagens para realizar o escalonamento em diferentes domínios: 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.

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*

[1] Aplicações são compostas de tarefas e de aplicações com diferentes necessidades de computação e comunicação.

e *softwares* heterogêneos [7, 14]. 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 em diversos trabalhos publicados [6, 2, 3, 10], 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 pode 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 [24], MPI [15]), etc. A escolha dos recursos também pode seguir diversas estratégias como minimizar o tempo de execução, maximizar a vazão, reservar recursos para períodos futuros, efetuar a divisão justa de recursos entre as tarefas (mais conhecido como *fairness*), entre outros [20]. 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 em produção.

1.1 Objetivo

O objetivo principal deste projeto é estudar e compreender diferentes escalonadores de aplicações para grades computacionais. Além de propor uma comparação detalhada entre eles através de simulações.

2 Exemplos de escalonadores

Segue abaixo um resumo de alguns dos escalonadores que estudaremos. Está lista será incrementada durante a fase de estudo bibliográfico. Alguns deles são voltados para ambientes de grades oportunistas. Outros adotam algoritmos de escalonamento menos complexos e limitam-se à execução de aplicações embaraçosamente paralelas.

A seguir, serão descritos alguns desses escalonadores, destacando-se suas características, virtudes

e fraquezas.

2.1 PBS/OpenPBS

O Portable Batch Scheduler (PBS) [16, 9] 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 para 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 [2] opta por escalonar primeiramente as tarefas de maior prioridade 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.

2.2 SLURM

SLURM (Simple Linux Utility for Resource Management) [25] 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 está 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.

2.3 Condor

Um dos sistemas pioneiros na área da computação oportunista, o projeto Condor [17, 13, 23], 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 [18], 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 [11] 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, escalonamento, 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.

2.4 OurGrid/MyGrid

Desenvolvido pela Universidade de Campina Grande, com o apoio da Hewlett Packard, o OurGrid [7] 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. 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.

A interação dos usuários com a grade é realizada através do módulo MyGrid [8]. O MyGrid oferece três opções de escalonamento: *Workqueue*, *Workqueue with Replication* (WQR) e *StorageAffinity* [21]. O *Workqueue* simplesmente escalona as tarefas submetidas aos recursos disponíveis em uma ordem arbitrária. O WQR (*WorkQueue with Replication*) é uma extensão do *Workqueue* sendo que, após a submissão de todas as tarefas, o escalonador passa a submeter réplicas das tarefas em execução até que não hajam mais recursos disponíveis. Como a estratégia de *Workqueue* 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 MyGrid 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 mede 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.

2.5 OAR

OAR [6] 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 [19] 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.

3 Metodologia e Resultados Esperados

Inicialmente estudaremos em detalhes os diversos ambientes de grades computacionais de forma a encontrar um conjunto representativo de escalonadores de aplicações. A partir deste conjunto será conduzido um estudo detalhado dos simuladores encontrados.

Em uma etapa posterior analisaremos diferentes simuladores de grades. Trabalhos recentes de Boeres e Rebello [3, 4] abordam a implementação de uma ferramenta chamada *Task Scheduling Testbed* através da qual é possível testar diferentes algoritmos de escalonamento. Essa ferramenta utiliza o GridSim [5] 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.

Além deste, também pretendemos estudar outros simuladores como o SimGrid (`http://simgrid.gforge.inria.fr/`) e GangSim (`http://people.cs.uchicago.edu/~cldumitr/GangSim/`. A partir da escolha de um ou mais simuladores, criaremos diferentes ambientes de execução de forma a comparar os diversos algoritmos de escalonamento.

4 Plano de Trabalho e Cronograma

	Anos e Semestres			
	2009		**2010**	
Atividades	1^o	2^o	1^o	2^o
Disciplinas obrigatórias	x	x		
Levantamento Bibliográfico	x	x		
Estudo dos diversos escalonadores		x	x	
Análise de ambientes de simulação		x	x	
Implementação dos algoritmos			x	x
Redação da dissertação				x

Tabela 1: Cronograma de atividades

Também se almeja no projeto a publicação de artigos científicos.

Referências

[1] Fran Berman, Geoffrey Fox, and Tony Hey. *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons, 2003.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] Walfredo Cirne, Daniel Paranhos, Lauro Costa, Elizeu Santos-Neto, Francisco Brasileiro, Jacques Sauvé, Fabrício A. B. Silva, Carla O. Barros, and Cirano Silveira. Running bag-of-tasks applications on computational grids: the mygrid approach. *Parallel Processing, 2003. Proceedings. 2003 International Conference on*, pages 407–416, Oct. 2003.

[9] Altair Engineering. Portable batch scheduler. `http://www.pbsgridworks.com`. Last accessed on 26 Out, 2008.

[10] 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.

[11] 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.

[12] Ian Foster and Carl Kesselman. *Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2003.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] Miron Livny, Jim Basney, Rajesh Raman, and Todd Tannenbaum. Mechanisms for high throughput computing. *SPEEDUP*, 11, 1997.

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

[20] 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.

[21] 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.

[22] Andrew S. Tanenbaum and Maarten Van Steen. *Distributed Systems: Principles and Paradigms*. Prentice-Hall, 2th edition, 2002.

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

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

[25] 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.