

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](mailto:vinicius@ime.usp.br)

Responsável: Prof. Alfredo Goldman - [gold@ime.usp.br](mailto: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 que, nesses domínios, esses recursos podem ser agrupados em aglomerados, como laboratórios científicos e intranets. 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 FIFO (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.




# Sumário

<b>1</b>	<b>Introdução</b>	<b>4</b>
1.1	Justificativa . . . . .	5
1.2	Objetivos . . . . .	6
<b>2</b>	<b>Trabalhos Correlatos</b>	<b>7</b>
2.1	PBS . . . . .	7
2.2	OAR . . . . .	7
2.3	SLURM . . . . .	8
<b>3</b>	<b>Metodologia e Resultados Esperados</b>	<b>8</b>
3.1	Agendamento de Tarefas . . . . .	9
3.2	Escalonamento de Aplicações Inter-aglomerado . . . . .	9
3.3	Módulo de Escalonamento e Adição de Algoritmos . . . . .	10
3.4	Resultados Esperados . . . . .	11
<b>4</b>	<b>Plano de Trabalho e Cronograma</b>	<b>11</b>

# 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 [6]. 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 [2]. 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*) focam na 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 [9, 6].

**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 freqüente entrada e saída de nós, e devem compartilhar *hardwares* e *softwares* heterogêneos [5, 7]. O escalonamento de aplicações em ambientes de ~~grade oportunista~~ consiste essencialmente em determinar quando, e em qual recurso, cada tarefa deve ser executada. Diferentes funcionalidades e mecanismos podem ser utilizados pelo escalonador, como filas de processos com atribuição de prioridades, casamento de tarefas e reserva de máquinas, entre outros. Diversas políticas de escalonamento podem ser adotadas. Minimizar o tempo médio de execução, maximizar o *throughput* de tarefas executadas, reduzir o tempo máximo de espera ou efetuar a divisão justa de

recursos entre as tarefas (mais conhecido como *fairness*) são alguns exemplos conhecidos [8]. 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 para grades oportunistas modular e adaptável, que se ajuste ao diversos cenários de disponibilidade de recursos, isto é, que seja sensível às alterações nas variáveis do ambiente de execução, como quantidade de processadores livres, largura de banda disponível, espaço em disco, etc.

## 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*) [4]. 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 indesejadas.

## 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*), *first-fit*, *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. O *Distributed Queuing System* (DQS), o *Load Sharing Facility* (LSF), o *Portable Batch Scheduler* (PBS), o OAR, o SLURM e o *LoadLeveler* da IBM são alguns deles [1], mas nenhum deles foram projetados para escalonamento em ambientes opornistas. A seguir, serão descritos alguns desses escalonadores.

### 2.1 PBS

O *Portable Batch Scheduler* (PBS) foi inicialmente desenvolvido para computadores paralelos de memória compartilhada de arquitetura SMP (*Shared Memory Multiprocessor*). O suporte para aglomerados foi adicionado posteriormente, mas ainda não contém funcionalidades importantes como, por exemplo, a submissão simultânea de tarefas em diversas máquinas. Por enquanto, essa função passa obrigatoriamente por um nó central. O escalonamento é realizado através de um algoritmo que mescla *First-In First-Out* 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. Uma versão modificada utiliza o *Maui Scheduler*. Esse escalonador 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 [3].

### 2.2 OAR

O OAR [4] é um escalonador em batch para aglomerados de grande porte e que 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 baseado em filas de prioridade, agendamento e backfilling. Através

de uma extensão, o OAR provê 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 estava sendo executavam nessa máquina, e as que dependem destas, são interrompidas. O OAR também possui um módulo de monitoramento da grade, denominado **TakTuk**. Todavia, esse módulo não faz análise de padrões de uso dos recursos.

## 2.3 SLURM

O SLURM (*Simple Linux Utility for Resource Management*) [10] é um escalonador de código aberto para aglomerados Linux de grande e pequeno porte. Com o foco na simplicidade, esse escalonador é altamente escalável, podendo controlar a execução de aplicação 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 submissão e monitoramento. O seu escalonador, contudo, é bastante simples, adotando uma política de FIFO (*First-In First-Out*). Apesar da sua escalabilidade, o SLURM não fornece suporte a computação em grade e, por ser desenvolvido somente para aglomerados Linux, não escalona tarefas em ambientes heterogêneos.

## 3 Metodologia e Resultados Esperados

Este projeto será realizado com os recursos do projeto InteGrade [7]. 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 as modificações propostas para o projeto InteGrade, e outras atividades no âmbito deste trabalho.



### 3.1 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 (*Application Submission and Control Tool*). Esse módulo já dispõe de uma interface para a submissão dos diversos modelos de aplicações contempladas pelo InteGrade: regulares, paramétricas, *Bulk Synchronous Parallel* (BSP) e MPI (*Message Passing Interface*). A modificação proposta consiste em adicionar na interface a opção de dia e hora para a execução da aplicação. Quando o usuário submeter a aplicação, o gerenciador do aglomerado (no InteGrade, representado pelo GRM ou *Global Resource Manager*) armazenará o binário da aplicação em um repositório e as informações de execução em uma tabela de um banco de dados simplificado. As informações das aplicações agendadas serão consultadas periodicamente, para que, nas datas planejadas, as aplicações sejam executadas.

### 3.2 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 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 modelos BSP e MPI, poderiam sofrer lentidão com os eventuais atrasos nos canais de comunicação. Dessa forma, pela sua simplicidade, em nosso projeto adotaremos o escalonamento inter-aglomerado entre aplicações.

Com o intuito de fazer com que aplicações possam ser encaminhadas de um aglomerado para outro, será necessário alterar o funcionamento do gerenciador do aglomerado. Como mencionado anteriormente, no InteGrade esse módulo é o 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. Na implementação que propomos, a submissão seria 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, possamos obter informações sobre todos os recursos do aglomerado. Dessa maneira, será possível que cada aglomerado realize consultas aos aglomerados adjacentes e escolha o mais adequado para encaminhar as aplicações que ele não pôde executar. Para que essas consultas sejam realizadas, é preciso implementar uma interface entre os gerenciadores dos aglomerados. O objetivo dessa interface, portanto, é fazer com que cada aglomerado enxergue o seu vizinho como um conjunto de recursos.

### 3.3 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) e outros aplicativos (i.e. Apache Ant, JConsole, 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 (LCPD) do Instituto de Matemática e Estatística da Universidade São Paulo. Nesses recursos, as versões modificadas do *middleware* poderão ser instaladas e avaliadas.

### 3.4 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 dotar o escalonador de 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 <sup>o</sup>	2 <sup>o</sup>	1 <sup>o</sup>	2 <sup>o</sup>	1 <sup>o</sup>	2 <sup>o</sup>
Levantamento Bibliográfico	x					
Agendamento de Aplicações		x				
Escalaonamento 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] Mark Baker, Geoffrey Fox, and Hon Yau. Cluster computing review. Technical Report CRPC-TR95623, Syracuse University, Northeast Parallel Architectures Center, November 1995.
- [2] Fran Berman, Geoffrey Fox, and Tony Hey. *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons, 2003.

- [3] 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.
- [4] 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.
- [5] 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.
- [6] Ian Foster and Carl Kesselman. *Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2003.
- [7] Andrei Goldchleger, Fabio Kon, Alfredo Goldman, Marcelo Finger, and Germano Capistrano Bezerra. Integrate: object-oriented grid middleware leveraging the idle computing power of desktop machines. *Concurrency - Practice and Experience*, 16(5):449–459, 2004.
- [8] 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.
- [9] Andrew S. Tanenbaum and Maarten Van Steen. *Distributed Systems: Principles and Paradigms*. Prentice-Hall, 2th edition, 2002.
- [10] 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.