

# Implantação Automatizada de Grades Computacionais Oportunistas

FABRICIO DE SOUSA NASCIMENTO<sup>1</sup>,  
ALFREDO GOLDMAN VEL LEJBMAN (ORIENTADOR)<sup>1</sup>  
e RAPHAEL Y. DE CAMARGO (ORIENTADOR)<sup>2</sup>

<sup>1</sup> Instituto de Matemática e Estatística, Universidade de São Paulo (USP), Brazil

{fabsn, gold}@ime.usp.br

<sup>2</sup> Universidade de São Paulo (USP), Brazil

## 1. introdução

Grades computacionais [1, 2, ?camargo05:plopd5] permitem o compartilhamento de recursos computacionais distribuídos de modo a aumentar sua utilização e diminuir seu custo de aquisição e manutenção. Grades oportunistas [4, 5, ?cirne06:ourgrid] são uma classe de grades computacionais com o objetivo de utilizar recursos ociosos de máquinas compartilhadas para a execução de aplicações computacionalmente intensivas.

O cenário clássico para grades computacionais é freqüente em universidades e instituições que muitas vezes possuem milhares de computadores que permanecem ociosos na maior parte do tempo. Utilizando grades oportunistas é possível utilizar o poder computacional de todas estas máquinas praticamente sem custos adicionais.

Mas a implantação de grades oportunistas, e sistemas distribuídos em geral, é trabalhosa e sujeita a erros. Mesmo nos casos em que os nós são homogêneos e que os usuários possuem amplo acesso aos nós, o trabalho é repetitivo e tedioso. Para a instalação e configuração do sistema é preciso copiar os códigos-fontes nos nós e compilá-los, instalar todas as bibliotecas que o sistema de grade necessita, configurar corretamente o ambiente e compilar os códigos-fontes. Finalmente, a inicialização dos diferentes componentes de uma grade requer que os diferentes módulos da grade sejam inicializados em uma determinada ordem e de maneira coordenada.

Esses desafios são suficientes para justificar o largo emprego de ferramentas automatizadas visando facilitar a vida de um administrador destes sistemas. Tais programas permitem com pouco esforço configurar e implantar aplicações distribuídas em um grande número de máquinas, evitando desta forma o trabalho repetitivo e permitindo uma economia de tempo e recursos. Por outro lado, estas ferramentas possuem importantes limitações, como falta de escalabilidade e baixa flexibilidade.

Com o objetivo de permitir a implantação automática do projeto do middleware de grade InteGrade [6, 7], desenvolvemos uma ferramenta baseada no programa TakTuk [8–10]. O TakTuk, desenvolvido no INRIA, França, permite a execução paralela de comandos em nós remotos de maneira eficiente, utilizando mecanismos de adaptação automática, balanceamento dinâmico de cargas e auto replicação. O InteGrade é um middleware que permite a criação de grades computacionais oportunistas de modo a utilizar os recursos ociosos de computadores pessoais para a execução de aplicação paralelas computacionalmente intensivas.

Neste artigo apresentamos o *IG-Deployer*, que permite a implantação automatizada de uma grade oportunista baseada no InteGrade. Esta ferramenta permite a instalação, configuração e gerenciamento automatizado de uma grade InteGrade. Por ser baseado no TakTuk, que é uma ferramenta genérica para sistemas distribuídos, as idéias presentes neste artigo podem ser estendidas para outros sistemas de grades oportunistas e sistemas distribuídos em geral. Resultados experimentais mostram que a implantação de uma grade oportunista utilizando o IG-Deployer é bastante eficiente e o tempo de implantação cresce de modo logarítmico com o número de nós.

## 2. a ferramenta TakTuk

TakTuk [9, 10] é uma ferramenta para execução paralela de comandos em um conjunto potencialmente grande de nós remotos. Dentre as principais características do TakTuk estão sua adaptação automática, balanceamento dinâmico de cargas e auto replicação.

Desenvolvido no INRIA na França, a ferramenta é utilizada em diversos sistemas, como a grade computacional Grid'5000 [12, 13], que é composta por milhares de nós heterogêneos espalhados geograficamente no território francês. No Grid'5000, o TakTuk é utilizado para implantar o sistema operacional das máquinas e para realizar tarefas administrativas. Outro sistema que utiliza o TakTuk é o KAAPI [11], um middleware que permite a execução de aplicações paralelas baseadas em fluxos de trabalho, e que utiliza o TakTuk para distribuir as tarefas da aplicação paralela nas máquinas da Grade.

TakTuk replica comandos em um número arbitrário de máquinas e permite ao usuário descrever os comandos a serem executados em arquivos, que tem

uma sintaxe própria bastante flexível. O TakTuk é escrito em perl e por isso é portátil. Sua principal vantagem está no desempenho, pois este utiliza um algoritmo adaptativo baseado em roubo de trabalhos (*work stealing*). Além disso, o TakTuk se instala automaticamente nas máquinas em que não está instalado, sendo enviado para estas máquinas através de conexões do tipo SSH.

Sua fase de execução de comandos é composta por dois estágios. O primeiro deles consiste em uma parte não paralelizável em que são estabelecidas as conexões, usualmente através do conector ssh. A segunda parte é a execução remota de comandos, que é paralelizável.

O TakTuk dispara os comandos em árvore, ou seja, cada nó da rede em que a primeira fase foi completa pode começar a se replicar em outros nós. A ramificação padrão do TakTuk é de 10 nós, o que significa que a máquina que inicia o processo escolhe 10 máquinas para serem seus alvos, repassando as tarefas a estas máquinas e transmitindo a lista de máquinas restantes. Cada um destes nós filhos, por sua vez, repassa as tarefas para outros 10 nós, e assim por diante. Este processo é ilustrado na Figura 1 (para uma ramificação de 3 nós).

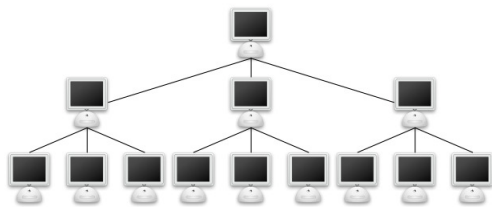


Figura 1. Topologia em árvore do TakTuk

A topologia em árvore tem então sua eficiência garantida pelo algoritmo de roubo de trabalhos. A princípio, até por volta de 100 máquinas, o desempenho do TakTuk será inferior ao da replicação simples de comandos, isso principalmente devido a sobrecarga gerada pela computação desse algoritmo. Entretanto, com um número superior a 100 máquinas o desempenho do TakTuk é bem superior ao da replicação simples [8].

## 2.1 outras ferramentas de implantação

Além do TakTuk, existem diversas outras ferramentas que permitem a implantação de sistemas distribuídos, como o Omnitty [15] e o Capistrano [14].

O Omnitty é um multiplexador de ssh que foi criado para simplificar o trabalho do administrador de sistemas Linux. Com ele é possível replicar comandos para grupos de máquinas em uma rede, desde instruções de linha de comando à aplicações que se utilizem de interface modo texto. Embora a idéia pareça simples, o Omnitty é uma ferramenta bastante poderosa e flexível. Não é preciso nenhum trabalho extra para que as rotinas preparadas para rodar em um único nó, possam simplesmente rodar em muitos nós.

Porém, o Omnitty possui algumas desvantagens que impedem seu uso como base para o *IG-Deployer*. Uma delas é que na ausência de um sistema de arquivos remotos como o NFS [16], os scripts a serem executados precisam ser replicados manualmente para todas as máquinas. Outro grande problema está no quesito plataforma, o Omnitty está fortemente ligado a sistemas UNIX, principalmente pelo fato de ser um emulador de terminais, o que torna seu uso em outros sistemas operacionais difícil, de tal modo que no tempo da escrita deste artigo, não havia sido portado ainda para sistemas não-Unix.

O Capistrano é um sistema de automatização de tarefas escrito na linguagem Ruby. Assim como o Omnitty ele replica uma série de instruções para um grupo de nós na rede. A principal diferença entretanto está ligada ao modo como as tarefas são automatizadas. No Capistrano, o usuário descreve as rotinas em um arquivo denominado *Capifile*, que guarda além de um lista das máquinas, um conjunto de comandos personalizados que encerram ações que podem ser invocadas, agendadas, para execução e até mesmo desfeitas.

Nesse sentido o Capistrano evita muito mais repetição do que o Omnitty, oferecendo um sistema mais elaborado de execuções de tarefas, organizando-as, e facilitando a sintaxe na definição de diferentes grupos para a implantação. Por essa simplicidade é que o Capistrano é largamente utilizado para implantação de sistemas *web* escritos em *Ruby on Rails* [18].

Embora ofereça muitos dos recursos necessários para implantação do *InteGrade*, o Capistrano não possui estratégias eficientes para replicação das tarefas. Conseqüentemente, quando o número de máquinas é muito grande, acaba sendo vantajoso utilizar uma ferramenta que execute a replicação de comandos de uma maneira paralela, como o TakTuk. Por estes motivos, decidimos utilizar o TakTuk como base para o desenvolvimento do *IG-Deployer*.

### 3. o InteGrade

O InteGrade [6, 7] é um middleware que permite a criação de grades computacionais oportunistas. O objetivo é utilizar os recursos ociosos de computadores pessoais para a execução de aplicação paralelas computacionalmente intensivas.

Uma grade InteGrade é organizada como uma federação de aglomerados, onde cada aglomerado é constituído por um conjunto de máquinas. A Figura 2 mostra a organização de uma grade InteGrade e os principais componentes que devem ser instanciados em cada aglomerado.

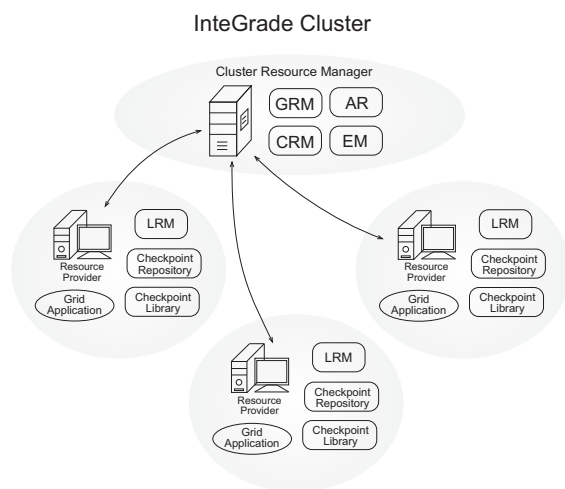


Figura 2. Arquitetura de um aglomerado InteGrade

As máquinas de cada aglomerado são divididas em duas classes: (1) máquinas *gerenciadoras de aglomerado*, que são as máquinas responsáveis por executar os módulos de gerenciamento de cada aglomerado e (2) máquinas *provedoras de recursos*, que compartilham seus recursos com a grade quando ociosas.

O *Local Resource Manager (LRM)* e o *Global Resource Manager (GRM)* realizam o gerenciamento dos recursos de cada aglomerado. Um LRM é executado em cada máquina provedora de recursos, coletando dinamicamente informações sobre o estado do nó e enviando-as periodicamente ao GRM. Além disso os LRMs controlam a execução de aplicações da grade em um nó. Já o GRM gerencia os LRMs de seu aglomerado e é responsável por receber requisições de execução de aplicações e escaloná-las nas máquinas disponíveis.

A submissão de aplicações para execução é realizada pela ferramenta *Application Submission and Control Tool (ASCT)*, que permite que usuários da grade submetam aplicações para execução, monitorem sua execução e obtenham os resultados da exe-

cução.

Outros módulos importantes do InteGrade são o *Execution Manager*, que é responsável por monitorar a execução das aplicações, a biblioteca de *checkpointing*, que provê tolerância a falhas às aplicações e as bibliotecas de programação paralela, BSPlib e MPI.

### 4. implantação automatizada do InteGrade

Um dos principais desafios na implantação do InteGrade é o grande número de módulos, escritos em linguagens de programação distintas, e que precisam ser instalados, configurados e inicializados em cada nó da grade.

No caso mais simples, em uma grade com um único aglomerado, precisamos instalar e instanciar um módulo GRM (*Global Resource Manager*) na máquina gerenciadora do aglomerado e módulos LRM (*Local Resource Manager*) nas demais máquinas do aglomerado. Além disso é preciso instanciar o ASCT (*Application Submission and Control Tool*), para permitir a submissão de aplicações para execução. Mas outros cenários exigem diversos outros módulos, como as bibliotecas de de programação paralela e de *checkpointing* e o módulo LUPA.

Outro desafio são as dependências do InteGrade com relação a outras bibliotecas e programas. O *IG-Deployer* precisa instalar e configurar todas estas dependências, que incluem as implementações de CORBA em Java (JacORB) e em Lua (OiL). Além disso, é preciso instalar a linguagem de programação Lua e algumas bibliotecas de desenvolvimento, como a OpenSSL. Estas dependências precisam ser empacotadas, distribuídas, desempacotadas e compiladas em seu local de destino, respeitando diferenças de plataformas, que atualmente são as arquiteturas i386 e x86\_64 do Linux. Além disso, os módulos do InteGrade precisam ser configurados corretamente para utilizar estas bibliotecas.

#### 4.1 iG-Deployer

O sistema de implantação utiliza scripts em Perl que preparam o TakTuk para realizar as tarefas de implantação da grade oportunista. O IG-Deployer permite que o usuário defina o caminho de instalação do InteGrade e suas dependências, a máquina que será usada como gerenciador do aglomerado e a lista de máquinas provedoras de recursos. Estas especificações são escritas utilizando o formato YAML ("YAML Ain't a Markup Language") [17]. Optamos por simplificar ao máximo a sintaxe do arquivo de configuração para facilitar a configuração

do IG-Deployer.

Um ponto onde o IG-Deployer auxilia bastante é no tratamento de erros durante a implantação. Caso ocorra algum erro no momento da compilação dos pacotes ou da inicialização da grade, o IG-Deployer continua a instalação ou inicialização, mas notifica o usuário em quais máquinas o erro ocorreu. Para tal, o IG-Deployer captura os códigos de erros gerados pelos processos de compilação ou inicialização e mostra os erros de forma amigável para o usuário.

A seqüência de implantação é definida em 4 principais estágios: empacotamento, distribuição, instalação e inicialização. Além disso, outras três tarefas estão disponíveis: limpeza, finalização e verificação. Estas tarefas estão definidas abaixo:

### **empacotamento (*Pack*)**

O estágio de empacotamento é o primeiro estágio do processo de implantação. Ele é executado apenas na máquina que lança a implantação e gera dois pacotes comprimidos contendo todos os arquivos necessários para a instalação do InteGrade. O objetivo deste estágio é melhorar o desempenho da transferência de arquivos, pois ele diminui o número de arquivos copiados e o tamanho total destes arquivos.

Os dois pacotes gerados são: (1) o de dependências, que contém os ORBs CORBA JacORB e OiL, a linguagem de programação Lua e as bibliotecas que o InteGrade utiliza e (2) o de instalação, que contém o InteGrade e todos os seus módulos.

### **distribuição (*Deploy*)**

O processo de distribuição é iniciado logo ao fim do processo de empacotamento, e inicia o envio dos pacotes para as máquinas clientes. Cada máquina recebe todos os módulos do InteGrade e suas dependências empacotadas. O processo de distribuição é feito através da transferência de arquivos para todas as máquinas.

### **instalação (*Install*)**

Ao finalizar a transferência dos pacotes, a fase de instalação é iniciada. Esta fase é composta pelas seguintes tarefas: desempacotar e organizar as dependências nos diretórios adequados, compilar as dependências e configurar e compilar os módulos do InteGrade. Esta é a fase mais demorada do processo, e pode levar vários minutos dependendo da configuração da máquina.

Para configurar os módulos do InteGrade, o IG-Deployer precisa gerar arquivos de configuração específicos para cada máquina. Por exemplo, é neces-

sário definir, para cada LRM, a localização do módulo GRM e das bibliotecas Java.

### **inicialização (*Start*)**

No final do processo de implantação, o IG-Deployer inicia os módulos do InteGrade. Esta inicialização deve ser feita de modo coordenado, pois existem dependências entre os módulos, como o serviço de nomes, o GRM e os LRMs, e estes módulos podem estar em máquinas distintas.

### **finalização (*Stop*)**

O processo de finalização termina a execução do InteGrade em um determinado conjunto de nós. Para tal, o IG-Deployer se conecta com as máquinas de modo hierárquico e finaliza os processos do InteGrade em cada máquina.

### **limpeza (*Clean*)**

O processo de limpeza remove todos os arquivos instalados e termina todos os processos do InteGrade em execução em um determinado conjunto de nós. Como as dependências e módulos do InteGrade são colocados em caminhos pré-definidos, esta limpeza se dá pelo termino da execução dos processos seguido da remoção dos arquivos.

### **verificação (*Check*)**

Verifica a sintaxe do arquivo YAML, bem como a disponibilidade das máquinas identificadas no mesmo. Este módulo é chamado antes da implantação, mas pode ser usado quando o usuário deseja verificar se uma implantação pode ser realizada.

## **5. experimentos**

O principal objetivo de utilizar o TakTuk é permitir a automatização do processo de implantação do middleware InteGrade. Mas devemos nos preocupar com o seu desempenho, pois uma implantação em dezenas de nós pode demorar horas se não for realizada de modo eficiente. Realizamos experimentos para medir o tempo de implantação do InteGrade utilizando diferentes números de nós.

### **5.1 cenário**

Para esses experimentos utilizamos 32 máquinas pertencentes a uma rede usadas por alunos da graduação em Ciência da Computação do Instituto de Matemática e Estatística da Universidade de São Paulo.

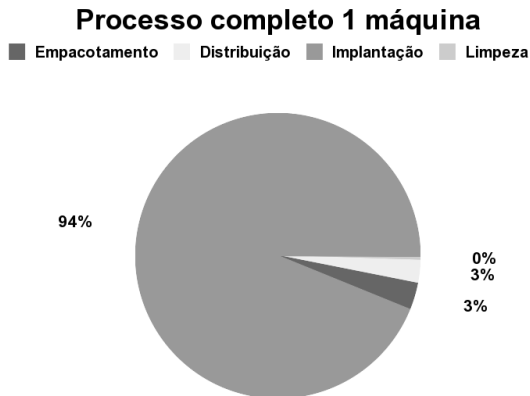


Figura 3. Divisão do tempo do processo de implantação em uma máquina

Estas máquinas são heterogêneas, com arquitetura de 32 e 64 bits, estão distribuídas em 4 diferentes laboratórios, e executam sistema operacional Linux. Todas as máquinas possuem pelo menos 1GB de memória RAM e processadores de velocidade igual ou superior a 2GHz.

Realizamos os experimentos com grupos de 1, 2, 4, 8, 16 e 32 máquinas, repetindo cada execução 20 vezes. Executamos os experimentos durante o período noturno e matinal, em que há pouco ou nenhum uso por parte dos alunos.

## 5.2 implantação em uma máquina

Para avaliar a contribuição de cada estágio no tempo total do processo de implantação, medimos o tempo deste processo em uma única máquina. Na Figura 3, vemos os resultados obtidos para 20 repetições da implantação.

Podemos perceber que o estágio mais longo é o da instalação, consumindo em média 244.15 segundos, o que corresponde a 94% do tempo total. Isto ocorre porque é preciso compilar todas as dependências e módulos do InteGrade. O processo de Distribuição consome apenas 3% do tempo total, pois envolve a transferência de 25,6 MB entre duas máquinas utilizando uma rede local de 100Mbps. O processo de empacotamento também consome 3% do tempo total, que é o tempo necessário para gerar o pacote que será transferido. Finalmente, o processo de limpeza, consome 0.7% do tempo total, o que dá em média 1.85s, que corresponde aproximadamente ao tempo de conexão através de SSH, pois a remoção de arquivos é bastante rápida.

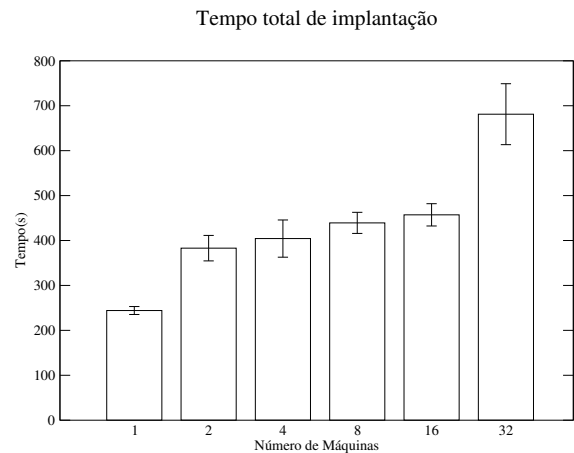


Figura 4. Tempo total de implantação do InteGrade em um aglomerado de máquinas

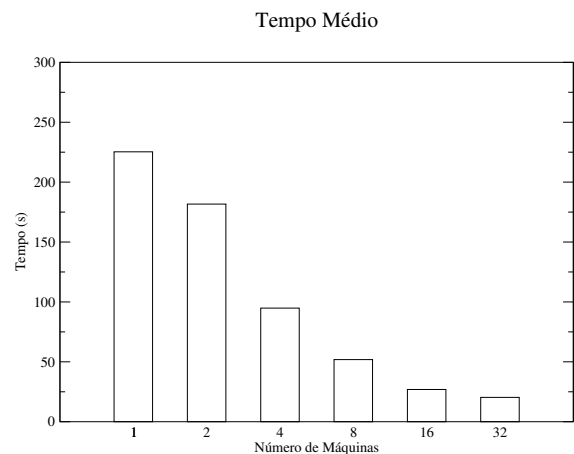


Figura 5. Tempo médio por máquina

## 5.3 implantação em aglomerados

Realizamos o processo de implantação em grupos de 1, 2, 4, 8, 16 e 32 máquinas de modo a avaliar como o IG-Deployer se comporta à medida que aumentamos o número de nós da grade.

Como podemos visualizar na Figura 4, o tempo total de implantação aumenta de modo logarítmico com relação ao número de máquinas, o que é justificado pelo fato do TakTuk executar as tarefas paralelamente.

Por exemplo, para 2 máquinas, foram necessários em média 645 segundos, enquanto para 32 máquinas, foram necessários apenas 2681 segundos. Conseqüentemente, aumentamos o número de máquinas em 16 vezes, mas o tempo de execução aumentou em pouco mais de 4 vezes.

Já na Figura 5, vemos o tempo médio gasto para instalação em cada máquina, isto é, o valor do tempo

total de implantação dividido pelo número de máquinas. Podemos verificar que o tempo médio cai à medida que aumentamos o número de nós no aglomerado. Este resultado era esperado, pois o processo de implantação é realizado simultaneamente em um maior número de máquinas.

Finalmente, de acordo com os autores do TakTuk [8], este possui melhor desempenho para configurações de mais de 100 máquinas. Deste modo, no caso de aglomerados com mais máquinas do que o utilizado em nossos experimentos, esperamos um desempenho ainda melhor do IG-Deployer.

## 6. conclusões

A instalação e implantação de grades computacionais oportunistas é uma tarefa demorada, repetitiva e sujeita a erros. Utilizando a ferramenta IG-Deployer, mostramos que é possível instalar e implantar uma grade oportunista de maneira simples e eficiente. Em particular, vimos que à medida que aumentamos o número de máquinas, o tempo total de implantação cresceu de forma logarítmica, o que permite que a implantação seja realizada em um grande número de máquinas. Este desempenho foi obtido devido às características do TakTuk, que realiza as tarefas de implantação paralelamente nas máquinas do aglomerado.

Estamos implementando otimizações para melhorar o desempenho da implantação. Por exemplo, a transferência é atualmente realizada utilizando uma conexão SSH, o que requer que todos os dados transmitidos sejam criptografados. Além disso, o TakTuk não é otimizado para distribuir arquivos, uma vez que a ferramenta foi desenvolvida para obter a difusão de execução ótima.

Finalmente, iremos desenvolver outras funcionalidades para o IG-Deployer, como a capacidade de implantar múltiplos aglomerados e uma maior automatização do processo de implantação.

## Referências

- [1] Ian Foster and Carl Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure* (2003).
- [2] Fran Berman and Geoffrey Fox and Tony Hey, *Grid Computing: Making the Global Infrastructure a Reality*, John Wiley & Sons, 2003.
- [3] Raphael Y. de Camargo and Andrei Goldchleger and Márcio Carneiro and Fabio Kon, *Grid: An Architectural Pattern*, The 11th Conference on Pattern Languages of Programs (PLoP'2004), 2004.
- [4] Michael and Livny Litzkow Miron and Mutka, *Condor - A Hunter of Idle Workstations*, ICDCS '88: Proceedings of the 8th International Conference of Distributed Computing Systems, 1988, pp. 104-111.
- [5] Andrei Goldchleger and Fabio Kon and Alfredo Goldman and Marcelo Finger, *InteGrade: Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines*, ACM/IFIP/USENIX 3rd International Workshop on Middleware for Grid Computing, 2003, pp. 232-234.
- [6] Andrei Goldchleger and Fabio Kon and Alfredo Goldman and Marcelo Finger and Germano C. Bezerra, *InteGrade: Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines*, Concurrency and Computation: Practice and Experience **16** (2004), 449-459.
- [7] Site do projeto InteGrade, <http://www.integrade.org.br> (2007).
- [8] Martin Cyrille, *Déploiement et contrôle d'applications parallèles sur grappes de grandes tailles* (2003).
- [9] O. Richard C. Martin and G. Huard, *Déploiement adaptatif d'applications parallèles*, Techniques et Sciences Informatiques **24** (2005), no. 5.
- [10] Guillaume Huard and Cyrille Martin, <http://taktuk.gforge.inria.fr/>, 2008.
- [11] Thierry Gautier and Xavier Besson and Laurent Pigeon, *KAAPI: A thread scheduling runtime system for data flow computations on cluster of multi-processors*, PASCO '07: Proceedings of the 2007 international workshop on Parallel symbolic computation, 2007, pp. 15-23, DOI <http://doi.acm.org/10.1145/1278177.1278182>, (to appear in print).
- [12] F. Cappello and E. Caron and M. Dayde and F. Desprez and Y. Jegou and P. Primet and E. Jeannot and S. Lanteri and J. Leduc and N. Melab and G. Mornet and R. Namyst and B. Quetier and O. Richard, *Grid'5000: A Large Scale and Highly Reconfigurable Grid Experimental Testbed*, GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing, 2005, pp. 99-106, DOI <http://dx.doi.org/10.1109/GRID.2005.1542730>, (to appear in print).
- [13] Grid'5000 site, <http://www.grid5000.fr> (2008).
- [14] Capistrano site, <http://www.capify.org/> (2006).
- [15] Bruno Takahashi de Oliveira. Omnitty SSH Multiplexer site, <http://omnitty.sourceforge.net/> (2004).
- [16] Brian Pawlowski and Spencer Shepler and Carl Beame and Brent Callaghan and Michael Eisler and David Noveck and David Robinson and Robert Thurlow, *The NFS Version 4 Protocol* ("2000"), no. "Technical Report 3085".
- [17] Oren Ben-Kiki and Clark Evans and Ingy dot Net, *YAML 1.2 Specification* (2008). Available at <http://www.yaml.org/spec/1.2>.
- [18] Dave Thomas and David Hansson and Leon Breedt and Mike Clark (Author) and James Duncan Davidson and Justin Gehtland and Andreas Schwarz, *Agile Web Development with Rails, 2nd Edition*, Pragmatic Bookshelf; 2 edition, 2006.
- [19] F. Cappello and E. Caron and M. Dayde and F. Desprez and Y. Jegou and P. Primet and E. Jeannot and S. Lanteri and J. Leduc and N. Melab and G. Mornet and R. Namyst and B. Quetier and O. Richard, *Grid'5000: A Large Scale and Highly Reconfigurable Grid Experimental Testbed*, GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing, 2005, pp. 99-106, DOI <http://dx.doi.org/10.1109/GRID.2005.1542730>, (to appear in print).
- [20] Grid'5000 site, <http://www.grid5000.fr> (2008).

- [21] Thierry Gautier and Xavier Besseron and Laurent Pigeon, *KAAPI: A thread scheduling runtime system for data flow computations on cluster of multi-processors*, PASC0 '07: Proceedings of the 2007 international workshop on Parallel symbolic computation, 2007, pp. 15–23, DOI <http://doi.acm.org/10.1145/1278177.1278182>, (to appear in print).