UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

Utilizando um Alocador Dinâmico para Ambientes Multi-cluster através da Submissão de uma Aplicação Distribuída

Leandro Santos Grapiuna

Florianópolis - SC

Julho / 2009

Leandro Santos Grapiuna

Utilizando um Alocador Dinâmico para Ambientes *Multi-cluster* através da Submissão de uma Aplicação Distribuída

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.

Orientador

Mário Antônio Ribeiro Dantas

Banca examinadora

Antônio Augusto Medeiros Frohlich

Frank Augusto Siqueira

Lau Cheuk Lung

Luís Fernando Friedrich

SUMÁRIO

RESUMO

Uma configuração em grades computacionais permite que recursos distribuídos geograficamente sejam disponibilizados. Um usuário deste ambiente pode, então, selecionar recursos para que esses executem tarefas específicas. Entretanto, o modelo usual de seleção de recursos não contempla a dinâmica do ambiente. Assim, uma forma diferenciada de seleção possibilita ao usuário otimizar a utilização destes recursos.

Partindo desses pressupostos, o ambiente distribuído utilizado, neste trabalho, é formado por duas organizações virtuais sendo que a primeira possui três *clusters* contendo três, quatro e cinco processadores respectivamente e a segunda, dois *clusters* compostos por quatro processadores cada.

Sendo assim, foi realizado o desenvolvimento de uma aplicação cliente, implementado em Java e utilizando RMI, cuja função é formular e enviar uma requisição por recursos ao alocador dinâmico apresentado em (FERREIRA et al, 2009) e, posteriormente, submeter o ordenador, implementado em C e utilizando a API GRAS, ao recurso selecionado pelo cliente dentre os sugeridos pelo alocador. Até o presente momento, não havia uma aplicação que pudesse fazer uso deste alocador. Portanto, este trabalho dá o primeiro passo no desenvolvimento aplicações para o alocador.

Dessa forma, o presente relatório apresenta o desenvolvimento tanto do cliente quanto do ordenador paralelo, além de disponibilizar material referente à pesquisa sobre alocação dinâmica conduzida pelo LAPESD possibilitando, assim, material para estudos futuros.

ABSTRACT

Computational grids allows resources that are geographically distributed to be available. An user for this environment can, then, select those resources to perform specific tasks. However, the usual model of resource selection does not address the dynamics of the environment. Thus, a different way of selection allows the user to optimize the use of these resources.

Assuming that, the distributed environment used in this work is formed by two virtual organizations: the first has three clusters containing three, four and five respectively processors and the second, two clusters with four processors each.

So, was developed a client application, implemented in Java using RMI, whose function is to formulate and send a request for resource to dynamic allocator presented in (Ferreira et al, 2009) and, subsequently, submit the sorter, implemented in C and using the API GRAS, to the resource selected by the client among those suggested by the allocator. Until now, there wasn´t an application that could use this allocator. Therefore, this work provides the first step in developing applications for the allocator.

This way, this report presents the development of both the client and the parallel sorter, in addition to provide material on the research of dynamic allocation conducted by LAPESD allowing material for future studies.

LISTA DE FIGURAS

1 INTRODUÇÃO

1.1 Contextualização

O Laboratório de Pesquisa em Sistemas Distribuídos (LAPESD) vem realizando estudos referentes à alocação dinâmica de recursos em um ambiente *multi-cluster*. Estes estudos resultaram na proposta apresentada em (FERREIRA et al., 2009), que utiliza uma abordagem de lógica *fuzzy* sobre ontologias. O paradigma de ontologia é utilizado para que os usuários possam expressar uma requisição através de uma interface padronizada. Além disso, algoritmos de lógica *fuzzy* servem para confrontar os parâmetros utilizados com os valores encontrados dinamicamente.

1.2 Objetivos

1.2.1 Objetivos gerais

- Desenvolver uma aplicação cliente que possa se comunicar com o alocador dinâmico para a requisição de recursos que serão alocados para a submissão de uma aplicação distribuída.

1.2.2 Objetivos específicos

- Implementar um cliente para o alocador dinâmico.
- Implementar uma aplicação distribuída.
- Executar a aplicação nos recursos selecionados pelo cliente a paritr das sugestões do alocador dinâmico em um ambiente *multi-*

cluster.

1.3 Justificativa

O alocador dinâmico apresentado em (FERREIRA et al., 2009) tem a capacidade de informar a um cliente o estado do ambiente distribuído levando em consideração os requisitos mínimos de uma requisição por recursos. Entretanto, não há clientes que possam realizar a seleção baseada nestas informações. Portanto, este trabalho implementa um cliente com o propósito de submeter o ordenador paralelo que será executado no recurso selecionado.

1.4 Organização do Texto

Este trabalho está organizado da seguinte forma: no capítulo 2 é feita uma fundamentação teórica referente a sistemas distribuídos, clusters e grades computacionais. Seguindo, o capítulo 3 é destinado ao desenvolvimento da aplicação. Posteriormente, no capítulo 4 é apresentada a conclusão do trabalho e, por último, o capítulo 5 indica trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Para um melhor entendimento deste trabalho é necessário uma breve apresentação de alguns conceitos envolvidos no mesmo. Os principais conceitos são referentes à computação distribuída, agregados computacionais e grades computacionais.

2.1 Sistemas Distribuídos

Um sistema distribuído pode ser definido como um sistema que realiza tarefas em processos diferentes. Geralmente, estes processos estão distribuídos fisicamente em duas ou mais plataformas de processamento. Quando ocorre essa situação deve-se levar em consideração, não só o meio de comunicação e seus protocolos, mas também as plataformas para um melhor aproveitamento do ambiente.

Porém, a heterogeneidade desses ambientes era um problema para os desenvolvedores de aplicação, sendo assim, houve uma necessidade de padronizar protocolos e interfaces de programação. Esta padronização de serviços foi a origem do que se chama hoje de *middleware*: uma camada de serviços localizada entre o sistema operacional e a aplicação, tornando transparente a utilização das plataformas em questão. Assim, um *middleware* substitui funções não distribuídas de um sistema operacional por funções distribuídas usando a rede (Bernstein, 1996).

A figura 1 mostra o modelo de um *middleware* posicionado entre as aplicações

e as plataformas.

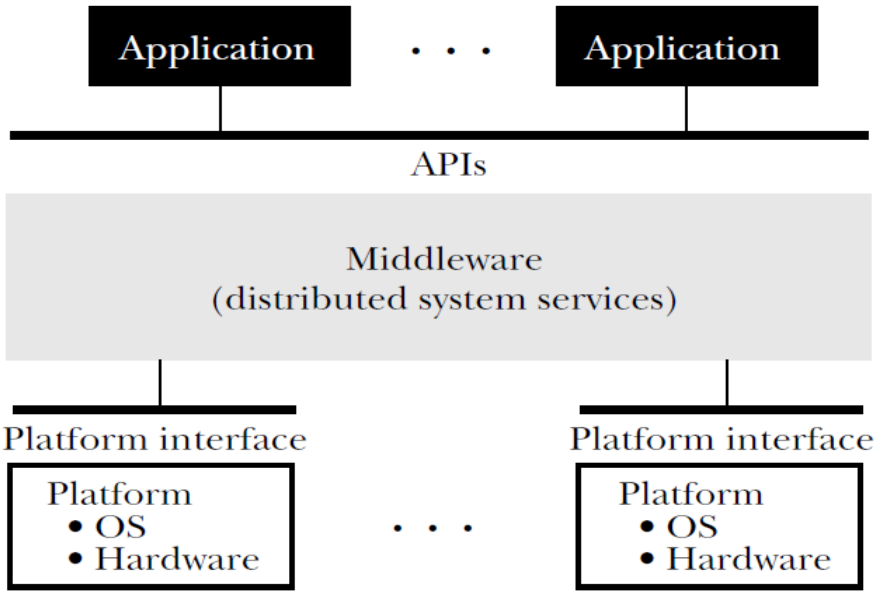

Figura 1: Middleware. (Bernstein, 1996)

2.2 Cluster

Um cluster é “uma agregação de computadores de uma forma dedicada (ou não) para a execução de aplicações específicas de uma organização” (Dantas, 2005). Em uma configuração de cluster, a opção de compor o agregado unicamente por computadores pessoais tem vantagens financeiras ao projeto por ser bem mais acessível a utilização desses computadores, conhecidos como COTS – *Commercial Off The Shelf*, do que soluções multiprocessadas.

Um agregado tem como característica a presença de um coordenador central e meios de comunicação de alta vazão e baixa latência. Isso permite que configurações de agregados de computadores sejam utilizados para um processamento paralelo eficiente.

2.3 Grades Computacionais

O conceito de grade computacional, do inglês *computational grid*, cunhado por Foster e Kesselman (1999), teve como inspiração a tecnologia de distribuição de energia elétrica utilizada por volta de 1910 nos Estados Unidos da América. Naquela época, quem necessitasse de eletricidade teria que construir e operar um gerador por conta própria, o que se assemelha ao uso cotidiano da computação. O desenvolvimento do *power grid*, uma rede de transmissão que conecta todas as usinas geradoras de uma região para maximizar a confiabilidade de entrega da energia, e da tecnologia de transmissão e distribuição associadas ao grid, viabilizou acesso confiável e de baixo custo a serviços padronizados. Portanto, o termo *grid*, em *grid computing*, é relacionado às características de serviços de um *power grid*, ou seja, uma infraestrutura que ofereça acesso a recursos de forma confiável, consistente, pervasiva e de baixo custo.

Foster e Kesselman (1999) apresentam cinco grandes classes de aplicações para o uso em grades computacionais:

- Supercomputação Distribuída – consiste em agregar ambientes multicomputados e/ou multiprocessados para solucionar problemas onde apenas um nó não seria suficiente.
- Computação de Alto-Desempenho - é usado para escalonar um grande número de tarefas independentes objetivando a solução de um único problema.
- Computação Sob Demanda – disponibiliza recursos para necessidades de curto prazo cuja propriedade não seja viável economicamente ou que não seja convenientemente alocado localmente.

- Computação Intensiva de Dados – lida com a síntese de novas informações a partir de dados mantidos em repositórios, bibliotecas digitais e bancos de dados geograficamente distribuídos, síntese essa que requer bastante processamento e comunicação.
- Computação Colaborativa – foca em disponibilizar e maximizar a integração entre pessoas.

Uma grade computacional tem como característica a inexistência de um coordenador central e a distribuição geográfica dos recursos pertencetes a organizações virtuais diferentes. Há um esforço por parte do OGF, *Open Grid Forum*, para que grades computacionais utilizem interfaces e protocolos de propósito geral abertos e padronizados.

Um ambiente *multi-cluster* é uma agregação de vários *clusters* disponíveis em um ambiente de grades computacionais.

3 PROPOSTA E DESENVOLVIMENTO

3.1 Introdução

A proposta deste trabalho está em implementar uma aplicação cliente capaz de formular uma requisição, representada por uma ontologia, submetê-la ao alocador dinâmico descrito em (FERREIRA et al., 2009) e executar o ordenador paralelo nos recursos de um ambiente *multi-cluster* simulado baseando-se em informações retornadas por este alocador dinâmico. A implementação desta aplicação cliente foi realizada utilizando a linguagem de programação Java e a tecnologia RMI (*Remote Method Invocation*), essa foi utilizada por ser a forma de comunicação implementada no alocador dinâmico. A implementação da requisição foi, inicialmente, baseada no *framework* JENA e, posteriomente, a API DOM (*Document Object Model*), isso foi possível pelo fato da requisição ser persistida em forma de arquivo OWL (*Web Ontology Language*) que é baseada na sintaxe XML. Para a configuração do ambiente sob a forma de informações semânticas foi utilizado o *software Protégé.* O ordenador paralelo foi desenvolvido utilizando a linguagem C com a ajuda da API GRAS disponibilizado pelo pacote de ferramentas SimGrid para possibilitar que esse seja executado em um ambiente distribuído e, principalmente, para realizar a sua execução em um simulador.

O restante do capítulo está organizado da seguinte forma: a sessão 3.2 fala da visão geral do sistema, a sessão 3.3 descreve o cliente e a 3.4 explica como é o formato da requisição. A aplicação que será submetida ao ambiente é apresentada na sessão 3.5. Ambiente este que é descrito na sessão 3.6. A sessão 3.7 se refere

ao simulador utilizado. E, por último, a execução dos testes é narrado na sessão 3.8.

3.2 Visão do Sistema

O sistema, como demonstrado na figura 2, funciona do seguinte modo: uma aplicação, com a intenção de submeter um *job* a um ambiente *multi-cluster,* envia uma requisição por recursos a um alocador. Este alocador, primeiro, seleciona os recursos que atendem aos requisitos do pedido comparando-os com as informações dos recursos e serviços das organizações virtuais publicadas em um repositório. Em seguida, o conjunto de recursos filtrados por esta primeira seleção serve como parâmetro para que o alocador dinâmico colete os dados do ambiente *multi-cluster*, que serão retornados ao cliente.

Resultados estes que contêm o nome do *cluster*, o número de *cpu´s* e a carga nos meios de comunicação. Desta forma, o alocador sugere os melhores recursos para a seleção, efetuada pelo cliente, para que, logo após, este possa submeter o *job*. No caso deste trabalho, a decisão foi pela menor carga nos meios de comunicação, mesmo que isto resulte em processadores ociosos.

O *job* a ser submetido é composto por um processo mestre e outros processos escravos que realizam uma ordenação em um conjunto de elementos. O resultado deste processo é retornado ao cliente.

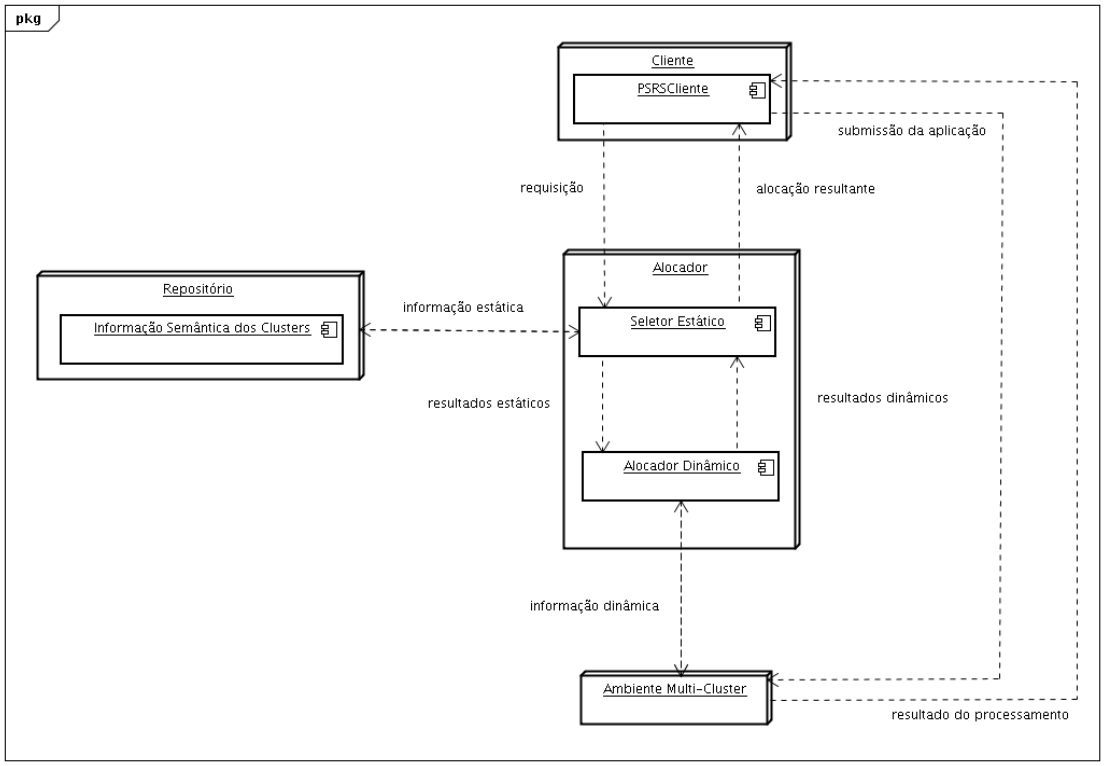

Figura 2: Visão do sistema

3.3 O Cliente

A aplicação cliente segue o diagrama de classe ilustrado na figura 3. Ela é composta por três classes: GridRequestOntology, ServerRMISelecionador e JobDispatcher. E tem dependêcia com outras duas: MSGCliente e MSGSelecionador.

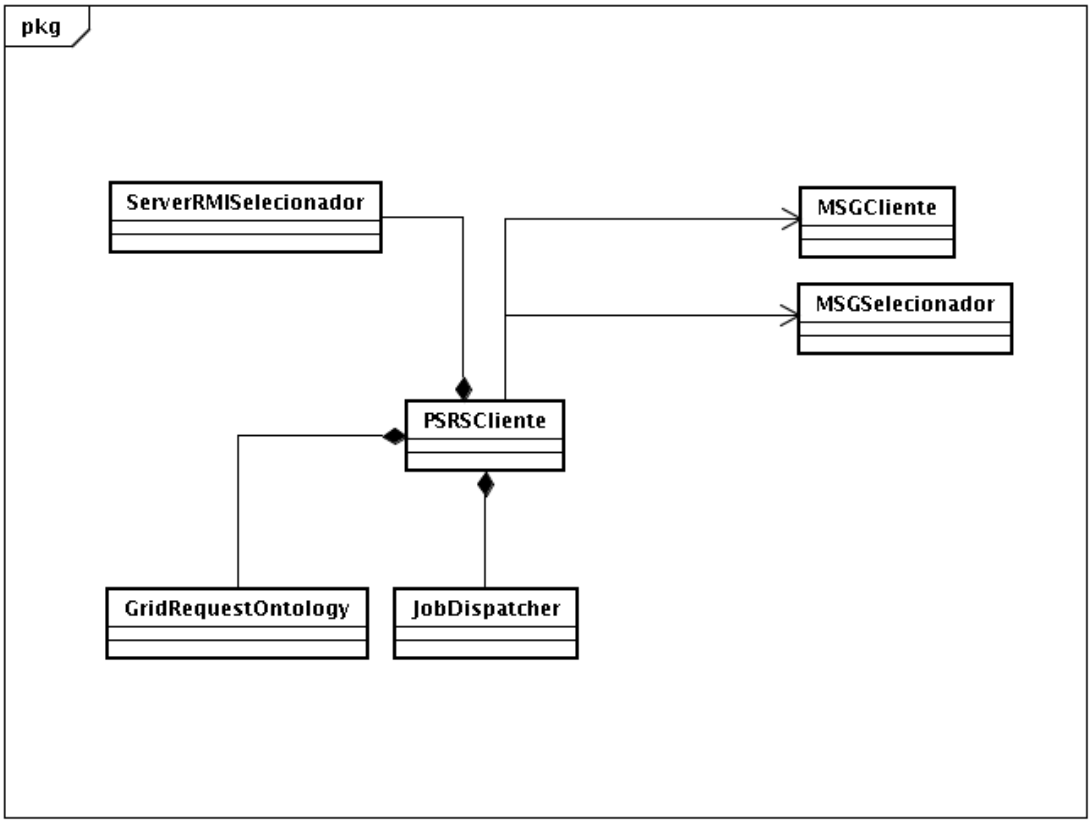

Figura 3: Diagrama de Classe

GridRequestOntology é responsável por formular a requisição em forma de ontologia. ServerRMISelecionador, onde a sua instância é um objeto remoto, é responsável pela alocação dos recursos. JobDispatcher é responsável em enviar o *job* ao recurso selecionado. O MSGCliente encapsula o pedido e o MSGSelecionador os recursos alocados, parâmetro e retorno do ServerRMISelecionador, respectivamente.

O diagrama de sequência é ilustrado na figura 4. O cliente manda uma mensagem para o GridRequestOntology informando o número mínimo de *cpu´s* como restrição. O GridRequestOntology formula a requisição e a persiste sob a forma de um arquivo OWL. A seguir, é enviada uma mensagem ao ServerRMISelecionador delegando a função de alocar os recursos. Com a lista de recursos selecionados, o cliente escolhe a melhor opção, no caso a menor carga nos meios de comunicação, e pede ao JobDispatcher que envie o *job* ao recurso

escohido.

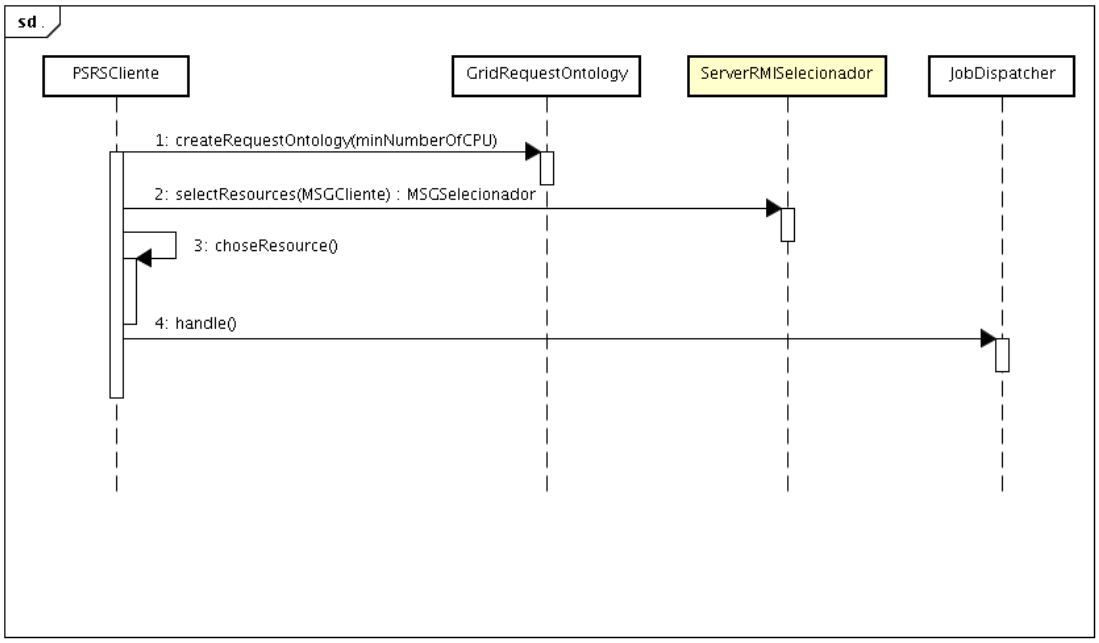

Figura 4: Diagrama de Sequência

3.4 A Requisição

A requisição (Request), como mostrado na figura 5, é composto por uma descrição do sistema (Description_Computer_System) em forma de requisitos (Requirements). Há quatro classes que os representam: requisitos para processadores (Requirements_CPU), requisitos para sistemas de arquivos (Requirements_FS), requisitos para sistema operacional (Requirements_SO) e requisitos para memória (Requirements_Memory). Neste trablalho, foi necessário apenas o requisito de processadores como restrição. Restrição que especifica o menor número de processadores presentes nos recursos.

Um arquivo OWL é uma forma de descrever classes, propriedades, restrições e indivíduos de uma ontologia utilizando a sintaxe XML/RDF (*eXtensible Markup Language* / *Resource Description Framework*).

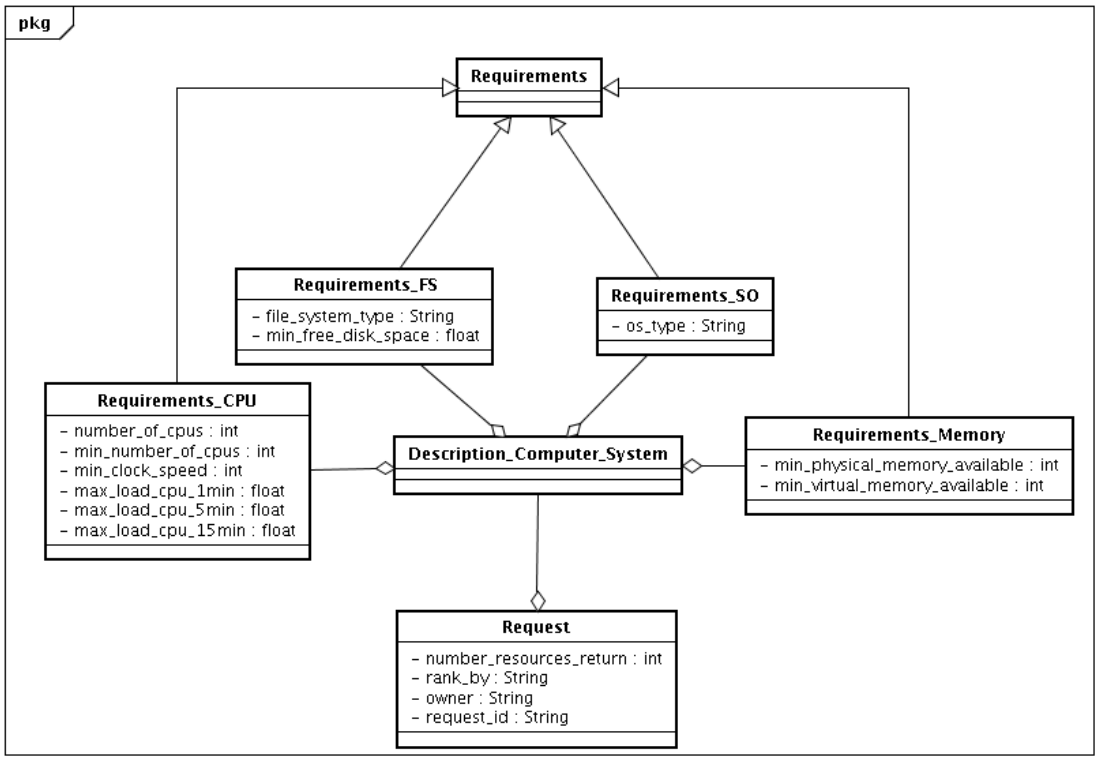

Figura 5: Requisição

3.5 O Job

A alocação dos recursos tem como propósito permitir ao cliente a possibilidade de executar algo nestes recursos, enviando um *job* ou executando um serviço disponível. Neste caso, o propósito é o envio de uma aplicação para ser executada no ambiente. Ou seja, o *job,* ou a aplicação, composta por uma ou mais tarefas, representado aqui por um processo mestre e n processos escravos, é enviado ao ambiente para ser executado.

A aplicação implementada em questão foi um ordenador de elementos baseado em um método de ordenação paralela denominado Ordenação Paralela por Amostragem Regular, ou *Parallel Sorting by Regular Sampling* (PSRS).

O PSRS funciona da seguinte forma: sendo um número e de elementos a serem ordenados, cada um dos p processos recebem e/p elementos. Cada processo, então, ordena os e/p elementos utilizando o método de ordenação *quick sort.* Com os elementos localmente ordenados, cada processo envia ao processo coordenador, este escolhido entre os processos participantes da ordenação, um conjunto de p – 1 elementos regularmente espaçados como amostra. O processo coordenador, em posse destas amostras, ordena-as, novamente por meio do *quick sort,* e escolhe as p – 1 amostras definitivas. Estas amostras, agora denominadas de pivôs, são, então, enviadas a todos os processos. Cada processo, utiliza os pivôs para particionar os elementos ordenados em p partições. Assim, cada partição é pertencente a um processo. Neste ponto, cada processo já sabe para onde deverá enviar cada partição realizando comunicações com todos os outros processos. Em seguida, como cada segmento já está ordenado, é realizado um *merge sort* de p – 1 passos. E, finalmente, os elementos são concatenados do primeiro processo ao último, uma vez que cada segmento já está ordenado e em sua posição. Este algoritmo é ilustrado por meio de um exemplo na figura 6.

Porém, o ordenador implementado apresenta algumas diferenças em relação ao ordenador supracitado. Primeirante, o processo coordenador é dedicado, não realizando a ordenação inicial dos elementos. Fica a cargo deste processo coordenador, chamado de processo mestre distribuir cada pedaço do espaço a ser ordenado aos processos escravos e receber os segmentos finais para a concatenação. Por conseguinte, o método de exclusão mútua implementado para que todos os processos escravos comuniquem entre si foi o método da barreira tendo o processo mestre como coordenador.

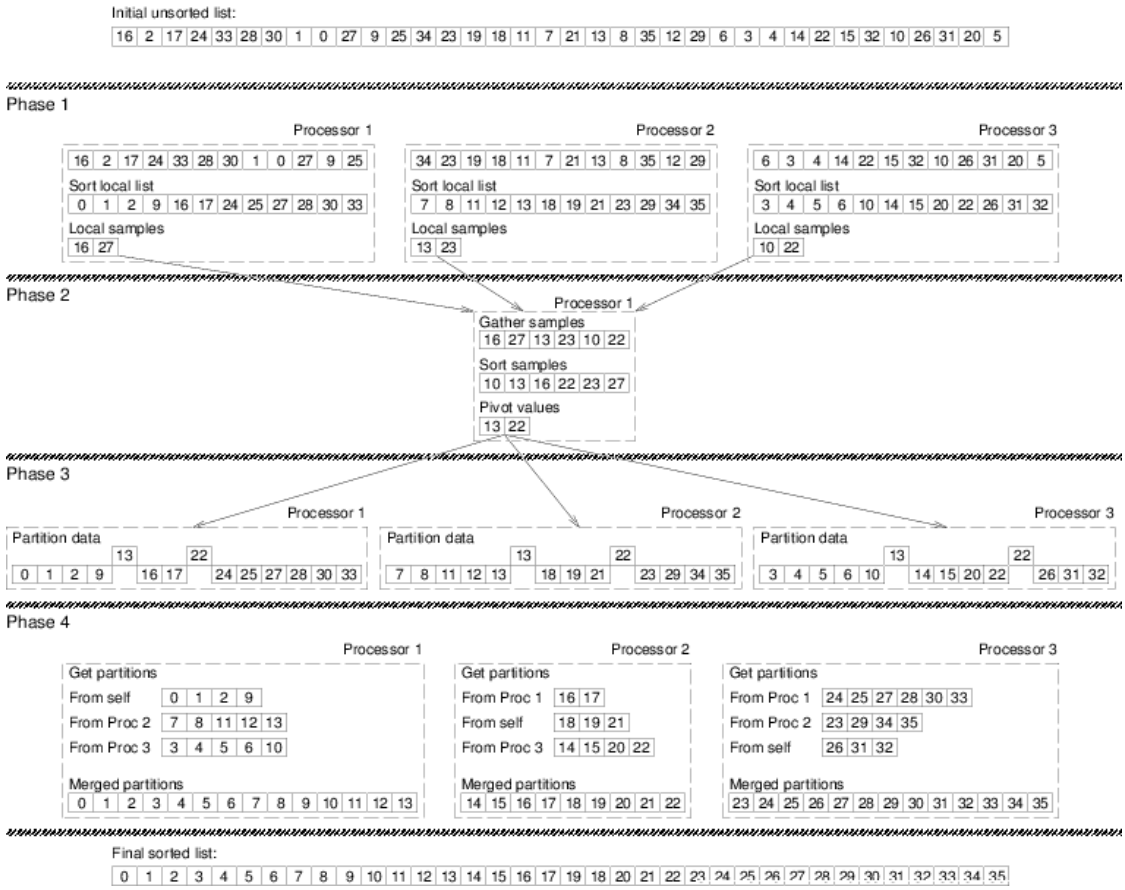

Figura 6: PSRS. fonte: http://www.usenix.org/publications/library/proceedings/coots99/full_papers/macdonald/macdonald_html/

3.6 O Ambiente

Para realizar os testes no sistema é preciso um ambiente. No trabalho apresentado em (FERREIRA et al., 2009) foi utilizado um simulador para executar o alocador dinâmico. Este simulador foi configurado para representar um ambiente que contém duas organizações virtuais, uma com três *clusters* contendo sete, cinco e três processadores respectivamente e a outra com dois *clusters* com quatro processadores cada.

Aqui utilizamos o mesmo ambiente para realizar a alocação dinâmica e um ambiente aproximado para a submissão do *job*. Este fato ocorreu por não ter sido possível reproduzir o mesmo ambiente. Porém, estima-se que o ambiente simulado

para a submissão do *job* esteja bastante próximo do utilizado na execução do alocador.

Para as informações semânticas, a Organização Virtual A tem estas configurações:

Cluster	cpus	clock	so
Cluster_01	7	1800	Linux
Cluster_02	5	2100	Linux
Cluster_03	3	2400	Linux

Enquanto que a Organização Virtual B apresenta as seguintes:

Cluster	cpus	clock	so
Cluster_1	4	1200	Linux
Cluster_2	4	1200	Linux

As informações dos recursos e serviços das organizações virtuais são publicadas em um repositório sob a forma de ontologia denominadas informações semânticas. O ambiente é representado na figura 7.

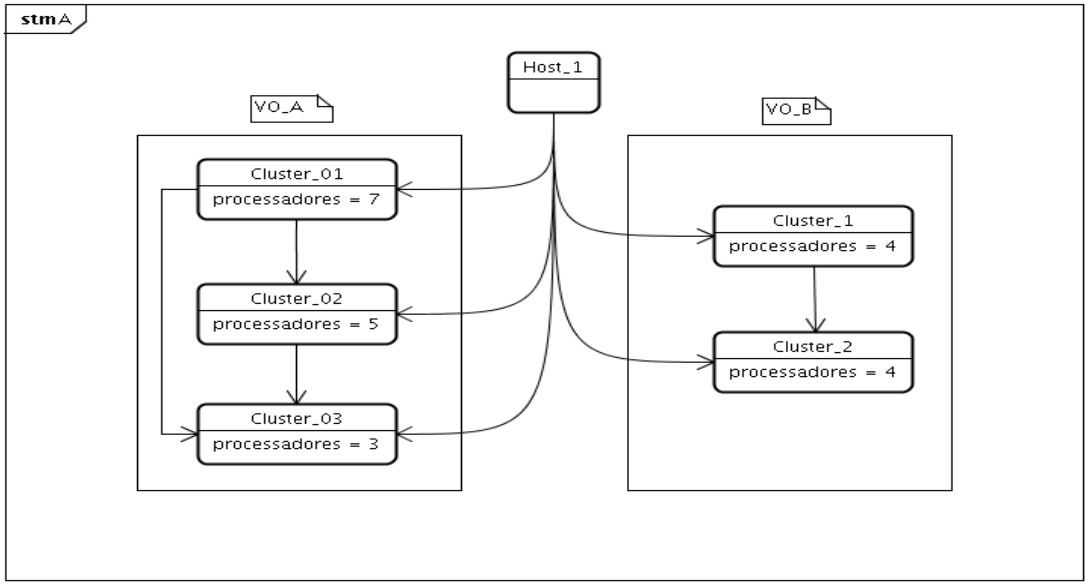

Figura 7: Ambiente multi-cluster

3.7 O Simulador

No momento em que o trabalho teve início, o alocador dinâmico não estava configurado para suportar um ambiente *multi-cluster* de teste e/ou produção. Como demonstrado em (FERREIRA et al., 2009), foi utilizado um simulador para o desenvolvimento do alocador dinâmico. Assim, este trabalho reutilizou este simulador, fazendo-se necessário a implementação de outra simulação que contemplasse o algoritmo de ordenação paralela por amostragem regular.

De acordo com (SimGrid, 2009), o SimGrid é “um *toolkit* que provê as principais funcionalidades para simulação de aplicações distribuídas em ambientes heterogêneos distribuídos. O objetivo principal do projeto é facilitar a pesquisa na área de escalonamento de aplicações distribuidas e paralelas em plataformas de computação distribuída e paralela indo de simples redes de *workstations* a grades computacionais.”

O SimGrid é formado, portanto, pelos seguintes componentes e ilustrados na figura 8:

- XBT (*eXtended Bundle of Tools*), que consiste em ferramentas básicas e de estruturas de dados.
- Surf é a camada que contém as principais funcionalidades para simular a plataforma.
- SimDag, biblioteca utilizada pelo usuário para simular escalonamento centralizado para tarefas paralelas baseadas em modelos DAG (*Direct Acyclic Graphs*).
- SMPI permite que aplicações escritas utilizando MPI (*Message Passing Interface*) possam ser simuladas sem que o código seja alterado.

- MSG (*MetaSimGrid*) recomendado apenas para o ambiente de simulação. Projetado para simular aplicações distribuídas.
- GRAS (*Grid Reality And Simulation*) permite que um mesmo código de uma aplicação distribuída seja utilizado tanto para simulação quanto para produção em um sistema real.
- AMOK (*Advanced Metacomputing Overlay Kit*) oferece serviços de alto nível baseados no GRAS.

Um aspecto referente ao GRAS que deve ser mencionado é a possibilidade de reaproveitar o código escrito tanto para gerar uma simulação quanto para gerar código para produção. Isso resprenta uma vantagem sobre os simuladores tradicionais, mesmo contra uma biblioteca oferecida no mesmo pacote: MSG.

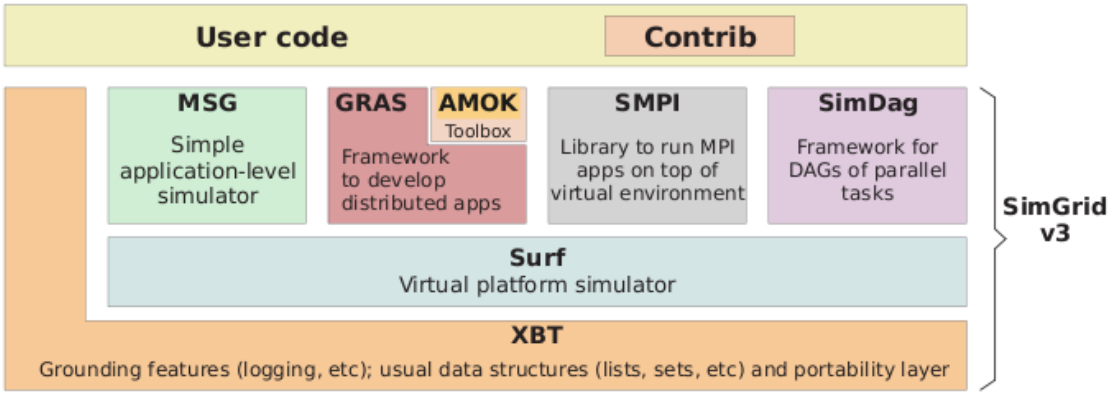

Figura 8: SimGrid

As figuras 9 e 10 mostram claramente esta diferença. Enquanto aquela representa um simulador convencional, em que um código escrito durante a pesquisa utilizando um simulador deverá ser reescrito caso o mesmo tenha o objetivo de executar em um ambiente de produção, esta mostra o esquema utilizado pelo GRAS, o mesmo código é utilizado tanto para a pesquisa quanto para a produção. Portanto, uma vez que a fase de pesquisa esteja concluída com os resultados esperados, o mesmo código estará pronto para produção, o que otimiza

esforços de equipe.

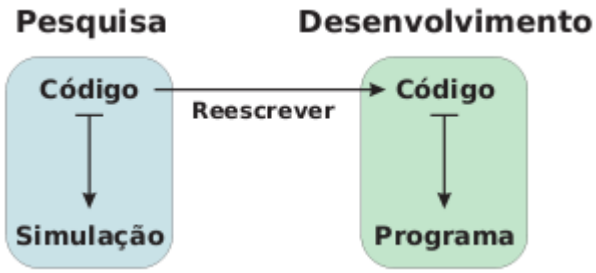

Figura 9: Simuladores Convencionais

Figura 10: GRAS

3.8 Testes

Pertinente ao que foi proposto, foram realizados alguns testes que consistiram em executar a aplicação cliente para requisitar a alocação de 4 recursos. Com o resultado do alocador foi preparado uma simulação para executar o *job* que ordenará uma lista de números inteiros contendo 1000, 2000, 3000, 4000 e 5000 elementos nestes recursos.

Primeiramente foi necessário configurar o ambiente. Para isso, dois arquivos OWL foram criados utilizando um software de autoria de ontologias, Protégé. A configuração destes arquivos segue as informações apresentadas nas tabelas da sessão 3.6. Estes arquivos representam informações semânticas dos recursos e

serviços que as organizações virtuais estão publicando e permitem ao alocador fazer uma primeira filtragem, de forma estática, sobre os recursos que serão alocados, como demostrado na figura 2.

Em seguida, foi preparado simulações para os testes de submissão. Para isso, foi reproduzido o arquivo de configuração da plataforma virtual adaptando-o às necessidades do algoritmo de ordenação paralela por amostragem regular e preparado os arquivos que representam a submissão (*deployment*). Com os arquivos de configuração e o servidor esperando por requisições do cliente, o próximo passo é executar o mesmo.

O cliente realiza uma requisição por recursos que contenham pelo menos quatro processadores. O alocador, então, retorna quais os recursos que satisfazem a requisição. No caso deste teste, foi retornado quatro clusters que atendem à requisição. Como o Cluster_03, mostrado na figura 7, tinha menos de quatro processadores ele foi eliminado já na etapa da seleção estática. O alocador, então, informa o nome do recurso, o total de processadores de cada recurso e a carga nos meios de comunicação. Estas informações são apresentadas na tabela a seguir.

recurso	processadores	Carga na comunicação
Cluster_01	7	0,018164
Cluster_02	5	0,050277
Cluster_1	4	0,050610
Cluster_2	4	37,898322

A partir disso, a aplicação cliente escolhe o Cluster_01. Esta escolha é realizada por apresentar a menor carga na comunicação. Em seguida, a mesma aplicação requisita a submissão do *job.* Esta submissão é realizada executando o simulador com a configuração adequada de alocação(*deployment*).

A escolha do critério de menor carga na comunicação resultou, nesta situação, em um recurso onde o poder de processamento de cada processador é o menor e em recursos ociosos. Assim, o *job* foi enviado para ser executado em quatro processos deixando três ociosos. A figura 11 mostra o performance do algoritmo durante a execução deste exemplo. Portanto, a submissão do *job* ao Cluster_01, que contém sete processadores, para ser executado em quatro deles com a tarefa de ordenar 1000, 2000, 3000, 4000 e 5000 elementos demorou 0.050184s, 0.051611s, 0.053141s, 0.054722s, 0.056146s respectivamente.

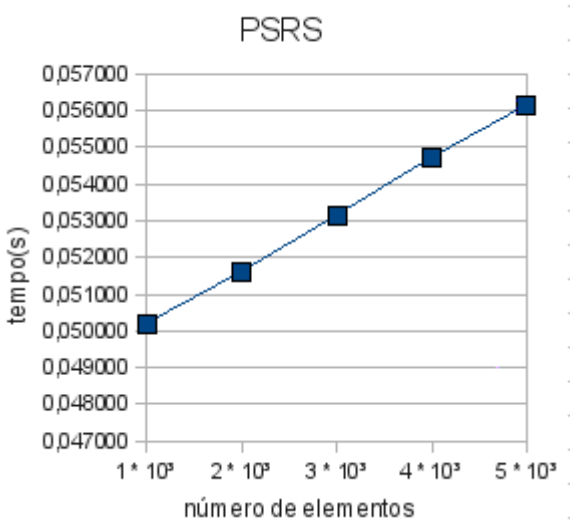

Figura 11: Tempos de execução do psrs no Cluster_01 utilizando 4 processadores

4 CONCLUSÃO

Otimizar a utilização de recursos em configurações de grades computacionais é uma área de pesquisa à qual se inclui os estudos realizados pelo LAPESD. Estudos esses que resultaram em um alocador dinâmico com a possibilidade de coletar informações do estado atual do ambiente distribuído e fazer sugestões de seleção de recursos. Porém, até então, não existia uma aplicação que utilizasse este alocador com esta finalidade. Por isso, a proposta deste artigo foi desenvolver esta aplicação.

Primeiro foi necessário a definição de uma aplicação que serviria como propósito para ser executada nos recursos selecionados, isso resultou na implementação de um ordenador baseado no algoritmo de ordenação paralela por amostragem regular. Depois foi necessário formular a requisição em forma de ontologia persistida na forma de arquivo OWL. Em seguida a aplicação cliente foi implementada para se comunicar com o alocador via RMI. Por fim, o ordenador foi submetido, via simulação, aos recursos selecionados.

Este trabalho, e todo o esforço demandado, proporcionou um grande aprendizado tanto teórico quanto prático. No início do projeto, houve uma insistência em utilizar o alocador em um ambiente real: no Unicore, um *middleware* para grades computacionais cuja principal característica é fornecer acesso a recursos distribuídos de forma transparente (MARZOLLA et al., 2007; ROMBERG, 1999). Assim, ocorram tentativas de configuração deste ambiente utilizando máquinas virtuais gerenciadas pelo VMWare Server. Em cada máquina foi

instalado uma distribuição Linux; primeiro Ubuntu, depois Fedora e novamento o Ubuntu na tentativa de instalação do Oscar, que permite aos usuários a instalação de um cluster para computação de alta desempenho do tipo Beowulf (OSCAR, 2009). Entretanto, ficou claro que as dificuldades referentes à configuração deste cluster, a sua disponibilização através do Unicore e posteriormente a adaptação às configurações do alocador dinâmico inviabilizavam o projeto dentro do prazo vigente.

Posteriormente, foi necessário um período para definir qual seria a aplicação que iria servir como propósito para o cliente. Primeiro foi analisado distribuir o método *quick sort* sequencial, mas como ele não apresentava interdependência entre os processos, esse foi descartado. Por fim, foi implementado, utilizando a API GRAS disponível pelo SimGrid para a linguagem C, um ordenador paralelo baseado no algoritmo de ordenação paralela por amostragem regular.

Com esta aplicação pronta, depois de um longo esforço, principalmente referente à configuração do ambiente simulado, foi possível realizar testes preliminares. Estes testes consistiram em comparar a execução da aplicação em recursos escolhidos entre os indicados pelo alocador e em outros arbitrários utilizando as métricas disponíveis que levavam em consideração apenas o tempo de envio dos dados. Com isso, foi possível comprovar a eficácia das sugestões do alocador dinâmico. Entretanto, para manter este relatório conciso e focado nos objetivos iniciais, optou-se por não apresentá-los no corpo do documento.

O desenvolvimento da aplicação cliente tanto quanto a configuração das informações semânticas não exigiu tanto esforço quanto ao exigido pelo simulador. Isso deve-se ao fato de não ter ocorrido nenhum problema nem erros, devido ao fato de se tratar de uma linguagem utilizada como padrão durante todo o curso e

pela boa usabilidade do *Protégé,* uma vez conhecidos os conceitos de ontologia. Após a proposta ser implementada, um conjunto de testes foi formulado. Desta vez, não para validar o alocador, mas para confirmar o funcionamento do sistema em geral. Assim, após a configuração do ambiente, a aplicação cliente foi executada para submeter uma requisição restringindo os recursos necessário a um mínimo de quatro processadores. Com as informações do alocador, o cliente escolhe aquele recurso que contém a menor carga nos meios de comunicação e, posteriormente, envia – via simulação – o ordenador paralelo para ser executado no ambiente *multi-cluster* satisfazendo, assim, os testes.

Com a conclusão deste trabalho é disponibilizado material que poderá ser reutilizado em estudos futuros. Sendo assim, o labor tornou-se um documento de extrema valia em relação ao que foi proposto.

5 TRABALHOS FUTUROS

Para trabalhos futuros, é recomendado a sintonia da formulação da requisição aos avanços do alocador dinâmico, a adequação do sistema a um ambiente real de computação em grade, bem como a utilização de um serviço de submissão de *jobs* do ambiente.

Uma outra abordagem, referente ainda ao alocador dinâmico, seria o desenvolvimento de uma aplicação interativa para que o usuário possa formular a requisição sem depender de ferramentas de autoria de ontologias como o *Protégé*.

6 ANEXOS

6.1 Artigo

Utilizando um Alocador Dinâmico para Ambientes *Multi-cluster* através da Submissão de uma Aplicação Distribuída

Leandro S. Grapiuna

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina
Caixa Postal 15.064 CEP 88040-900 Campus Universitário – Florianópolis, SC – Brasil

grapiuna@inf.ufsc.br

Abstract. *In computational grids, a different way for selection and use of available resources can optimize the efficiency of use of a distributed environment. The dynamic allocator presented in [Ferreira et al., 2009] tells the client the environment state taking into account the requirements of a request for resources. This article, then, describes the development of this client, who has the responsibility to make the selection of resources from a request to the allocator, and the submission of a distributed application to a multi-cluster environment.*

Resumo. *Em grades computacionais, uma forma diferenciada de seleção e utilização dos recursos disponíveis otimiza a eficiência de utilização do ambiente distribuído. O alocador dinâmico apresentado em [FERREIRA et al., 2009] informa ao cliente o estado do ambiente levando em consideração os requisitos mínimos de uma requisição por recursos. Este artigo, então, descreve o desenvolvimento deste cliente, que tem a responsabilidade de efetuar a seleção de recursos a partir da requisição ao alocador, assim como a submissão de uma aplicação distribuída ao ambiente multi-cluster.*

1. Introdução

Uma configuração em grades computacionais permite que recursos distribuídos geograficamente sejam disponibilizados. Um usuário deste ambiente pode, então, selecionar recursos para que esses executem tarefas específicas. Entretanto, o modelo usual de seleção de recursos, o qual é baseado apenas em informações estáticas dos mesmos, não contempla a dinâmica do ambiente. Assim, uma forma diferenciada de seleção de recursos que possa ser baseada em informações referentes ao estado do ambiente possibilita otimizar a eficiência na utilização do mesmo.

O ambiente distribuído utilizado caracteriza uma configuração de grade computacional composto principalmente por *clusters.* Este ambiente é formado por duas organizações virtuais. Uma possuindo três *clusters* contendo três, quatro e cinco processadores respectivamente. A outra, dois *clusters* compostos por quatro processadores cada. Ambas organizações virtuais publicam os recursos disponíveis em forma de informações semânticas, ou seja, na forma de ontologia.

O alocador dinâmico apresentado em [FERREIRA et al., 2009] consegue adquirir informações referentes ao estado do ambiente distribuído. Informações essas disponibilizadas a uma aplicação cliente que decidirá, entre as opções mostradas, a que lhe é mais adequada. Porém, até o presente momento, não existia uma deste tipo para este alocador. O presente artigo, então, descreve o desenvolvimento desta aplicação assim como a implementação de um ordenador paralelo que será executado no ambiente *multi-cluster* através do simulador SimGrid.

A sequência do documento está organizada da seguinte forma. A seção 2 é destinada ao

desenvolvimento da proposta. Na seção 3 é relatado os testes provenientes da execução da aplicação. E a conclusão é feita na seção 4.

2. Desenvolvimento

A condução do projeto foi realizado da seguinte forma. Primeiro foi desenvolvido uma aplicação que seria executada no ambiente *multi-cluster*. Esta aplicação trata-se de uma implementação baseada no algoritmo de ordenação paralela por amostragem regular. Para isso, foi utilizado uma API na linguagem de programação C para a implementação de aplicações distribuídas de larga escala e sua execução em um ambiente simulado. Esta API é disponibilizada pelo SimGrid. E, posteriormente, o desenvolvimento de uma aplicação cliente para formular e enviar uma requisição por recursos ao alocador dinâmico e submeter uma aplicação distribuída, no caso o ordenador, ao recurso selecionado. Em seguida é apresentado o ambiente e o desenvolvimento tanto do cliente quanto do ordenador paralelo.

2.1. Visão do Sistema

O sistema, como demonstrado na figura 1, funciona do seguinte modo: uma aplicação, com a intenção de submeter um *job* a um ambiente *multi-cluster,* envia uma requisição por recursos a um alocador. Este alocador, primeiro, seleciona os recursos que atendem aos requisitos do pedido comparando-os com as informações dos recursos e serviços das organizações virtuais publicadas em um repositório. Em seguida, o conjunto de recursos filtrados por esta primeira seleção serve como parâmetro para que o alocador dinâmico colete os dados do ambiente *multi-cluster*, que serão retornados ao cliente.

Resultados estes que contêm o nome do *cluster*, o número de *cpu´s* e a carga nos meios de comunicação. Desta forma, o alocador sugere os melhores recursos para a seleção, efetuada pelo cliente, para que, logo após, este possa submeter o *job*. No caso deste trabalho, a decisão foi pela menor carga nos meios de comunicação, mesmo que isto resulte em processadores ociosos.

O *job* a ser submetido é composto por um processo mestre e outros processos escravos que realizam uma ordenação em um conjunto de elementos. O resultado deste processo é retornado ao cliente.

2.2. O Cliente

A aplicação cliente segue o diagrama de classe ilustrado na figura 2. Ela é composta por três classes: GridRequestOntology, ServerRMISelecionador e JobDispatcher. E tem dependêcia com outras duas: MSGCliente e MSGSelecionador.

GridRequestOntology é responsável por formular a requisição em forma de ontologia. ServerRMISelecionador, onde a sua instância é um objeto remoto, é responsável pela alocação dos recursos. JobDispatcher é responsável em enviar o *job* ao recurso selecionado. O MSGCliente encapsula o pedido e o MSGSelecionador os recursos alocados, parâmetro e retorno do ServerRMISelecionador, respectivamente.

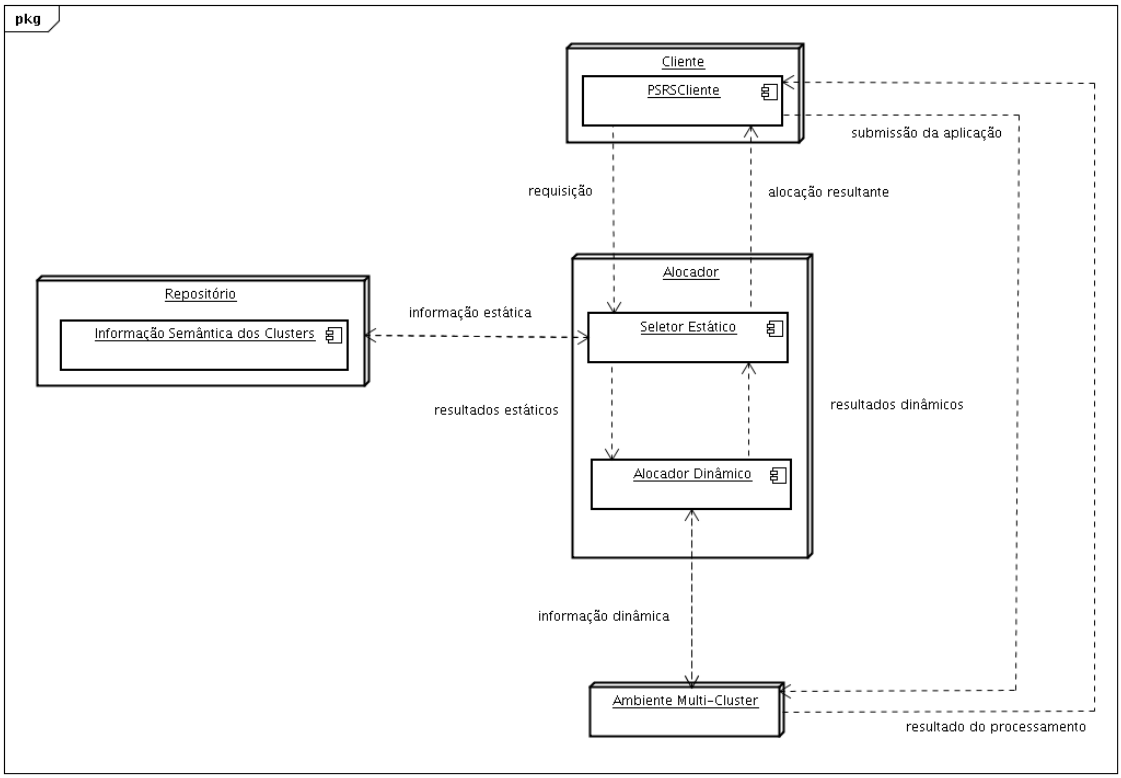

Figura 12: Visão do Sistema

O diagrama de sequência é ilustrado na figura 3. O cliente manda uma mensagem para o GridRequestOntology informando o número mínimo de *cpu´s* como restrição. O GridRequestOntology formula a requisição e a persiste sob a forma de um arquivo OWL. A seguir, é enviada uma mensagem ao ServerRMISelecionador delegando a função de alocar os recursos. Com a lista de recursos selecionados, o cliente escolhe a melhor opção, no caso a menor carga nos meios de comunicação, e pede ao JobDispatcher que envie o *job* ao recurso escohido.

2.3. A Requisição

A requisição (Request), como mostrado na figura 5, é composto por uma descrição do sistema (Description_Computer_System) em forma de requisitos (Requirements). Há quatro classes que os representam: requisitos para processadores (Requirements_CPU), requisitos para sistemas de arquivos (Requirements_FS), requisitos para sistema operacional (Requirements_SO) e requisitos para memória (Requirements_Memory). Neste trablalho, foi necessário apenas o requisito de processadores como restrição. Restrição que especifica o menor número de processadores presentes nos recursos.

Um arquivo OWL é uma forma de descrever classes, propriedades, restrições e indivíduos de uma ontologia utilizando a sintaxe XML/RDF (*eXtensible Markup Language* / *Resource Description Framework*).

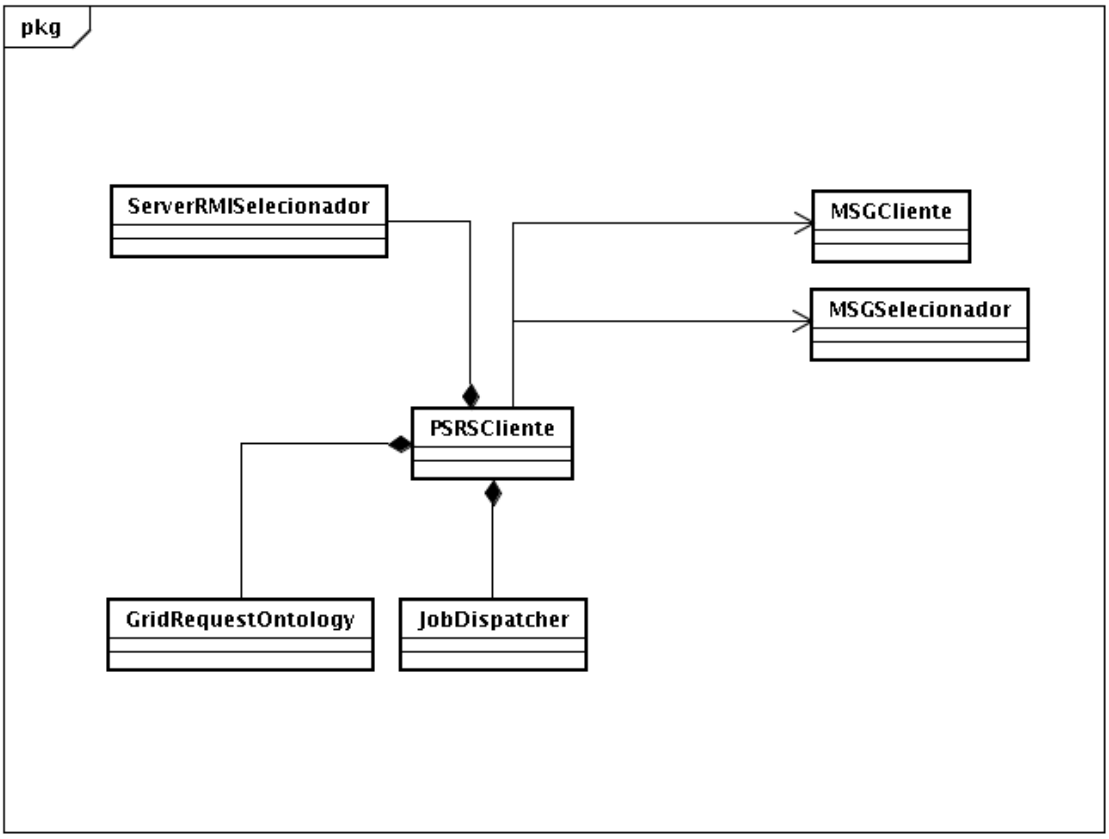

Figura 13: Diagrama de Classe

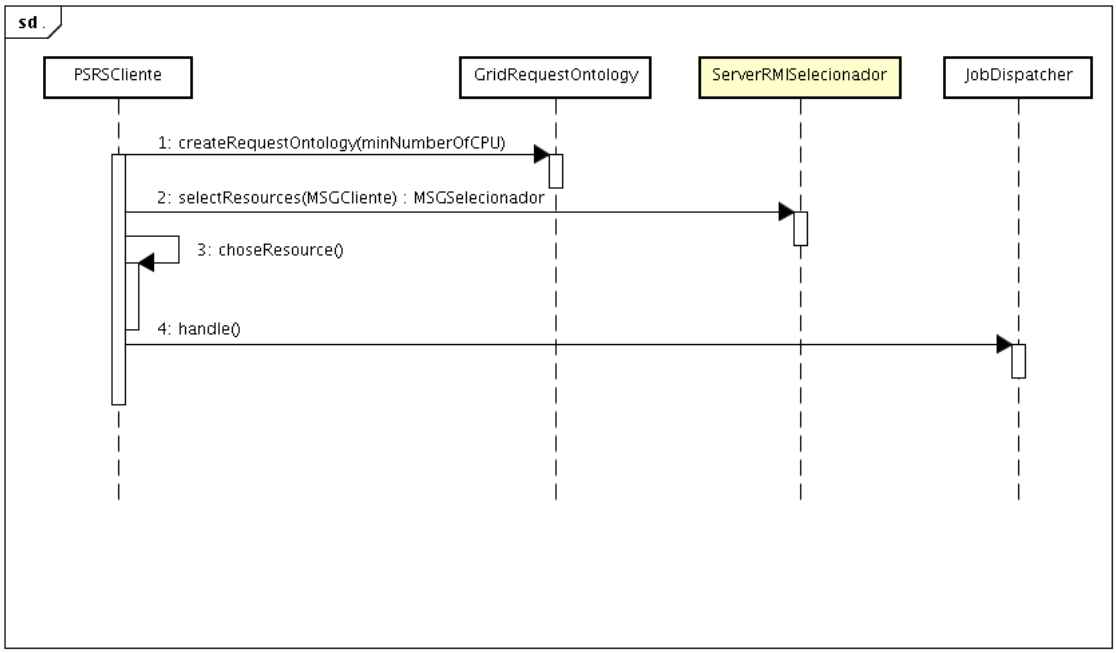

Figura 14: Diagrama de Sequência

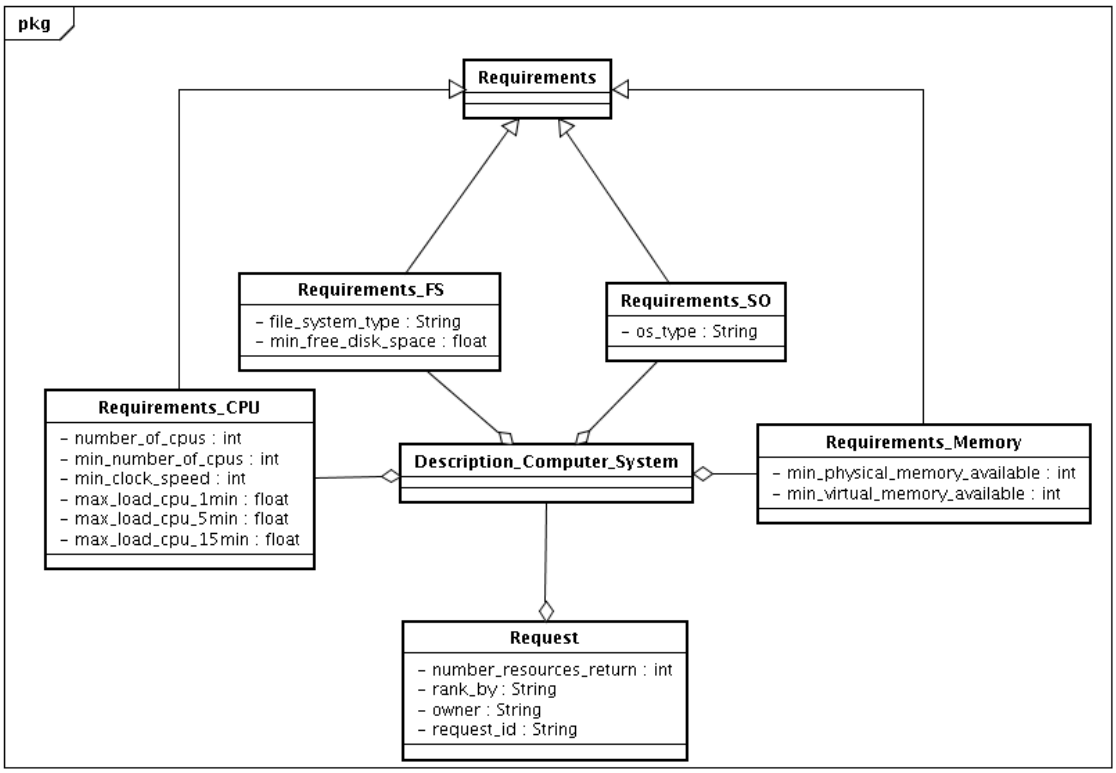

Figura 15: Requisição

2.4. O Job

A alocação dos recursos tem como propósito permitir ao cliente a possibilidade de executar algo nestes recursos, enviando um *job* ou executando um serviço disponível. Neste caso, o propósito é o envio de uma aplicação para ser executada no ambiente. Ou seja, o *job,* ou a aplicação, composta por uma ou mais tarefas, representado aqui por um processo mestre e n processos escravos, é enviado ao ambiente para ser executado.

A aplicação implementada em questão foi um ordenador de elementos baseado em um método de ordenação paralela denominado Ordenação Paralela por Amostragem Regular, ou *Parallel Sorting by Regular Sampling* (PSRS).

O PSRS funciona da seguinte forma: sendo um número e de elementos a serem ordenados, cada um dos p processos recebem e/p elementos. Cada processo, então, ordena os e/p elementos utilizando o método de ordenação *quick sort.* Com os elementos localmente ordenados, cada processo envia ao processo coordenador, este escolhido entre os processos participantes da ordenação, um conjunto de p – 1 elementos regularmente espaçados como amostra. O processo coordenador, em posse destas amostras, ordena-as, novamente por meio do *quick sort,* e escolhe as p – 1 amostras definitivas. Estas amostras, agora denominadas de pivôs, são, então, enviadas a todos os processos. Cada processo, utiliza os pivôs para particionar os elementos ordenados em p partições. Assim, cada partição é pertencente a um processo. Neste ponto, cada processo já sabe para onde deverá enviar cada partição realizando comunicações com todos os outros processos. Em seguida, como cada segmento já está ordenado, é realizado um *merge sort* de p – 1 passos. E, finalmente, os elementos são concatenados do primeiro processo ao último, uma vez que cada segmento já está ordenado e em sua posição. Este algoritmo é ilustrado por meio de um exemplo na figura 5.

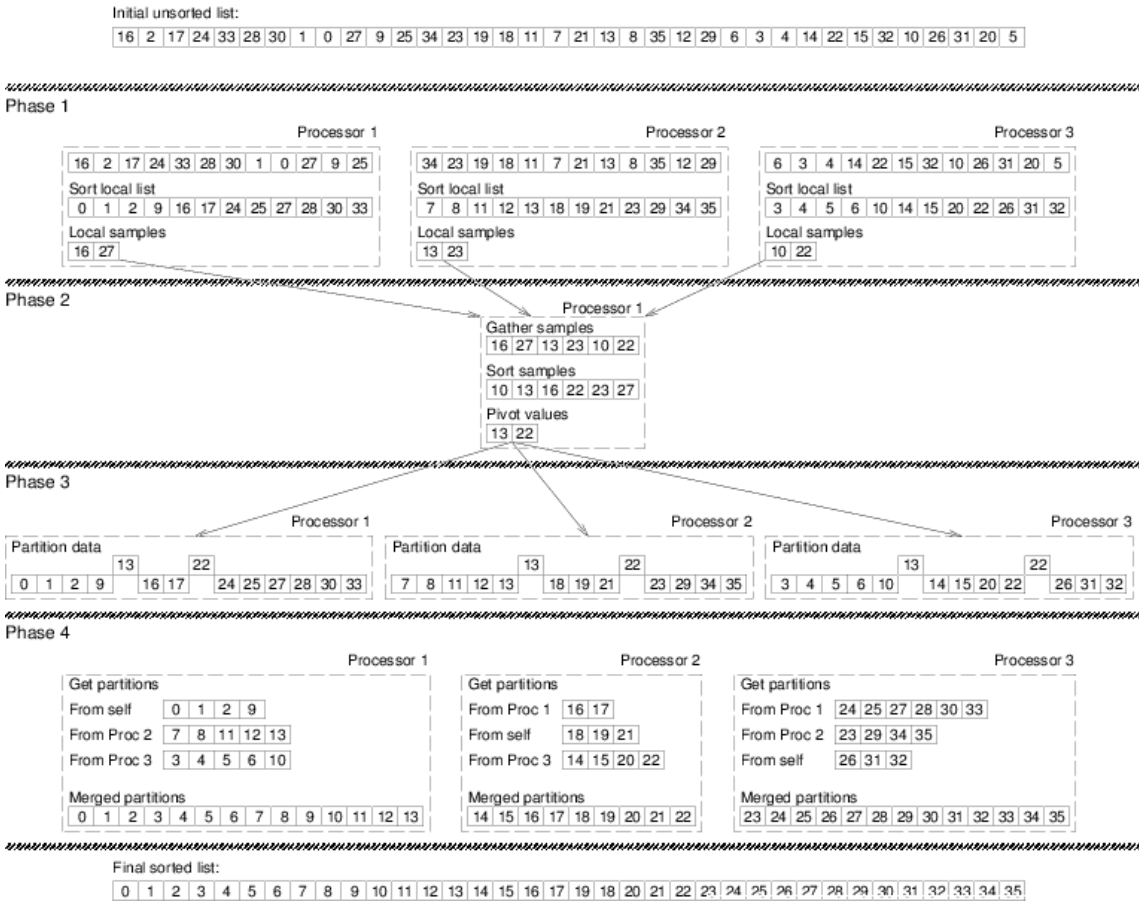

Figura 16: PSRS

Porém, o ordenador implementado apresenta algumas diferenças em relação ao ordenador supracitado. Primeirante, o processo coordenador é dedicado, não realizando a ordenação inicial dos elementos. Fica a cargo deste processo coordenador, chamado de processo mestre distribuir cada pedaço do espaço a ser ordenado aos processos escravos e receber os segmentos finais para a concatenação. Por conseguinte, o método de exclusão mútua implementado para que todos os processos escravos comuniquem entre si foi o método da barreira tendo o processo mestre como coordenador.

2.5. O Ambiente

Para realizar os testes no sistema é preciso um ambiente. No trabalho apresentado em (FERREIRA et al., 2009) foi utilizado um simulador para executar o alocador dinâmico. Este simulador foi configurado para representar um ambiente que contém duas organizações virtuais, uma com três *clusters* contendo sete, cinco e três processadores respectivamente e a outra com dois *clusters* com quatro processadores cada.

Aqui utilizamos o mesmo ambiente para realizar a alocação dinâmica e um ambiente aproximado para a submissão do *job*. Este fato ocorreu por não ter sido possível reproduzir o mesmo ambiente. Porém, estima-se que o ambiente simulado para a submissão do *job* esteja bastante próximo do utilizado na execução do alocador.

Para as informações semânticas, a Organização Virtual A tem estas configurações:

cluster	cpus	clock	so
Cluster_01	7	1800	Linux
Cluster_02	5	2100	Linux
Cluster_03	3	2400	Linux

Enquanto que a Organização Virtual B apresenta as seguintes:

cluster	cpus	clock	so
Cluster_1	4	1200	Linux
Cluster_2	4	1200	Linux

As informações dos recursos e serviços das organizações virtuais são publicadas em um repositório sob a forma de ontologia denominadas informações semânticas. O ambiente é representado na figura 6.

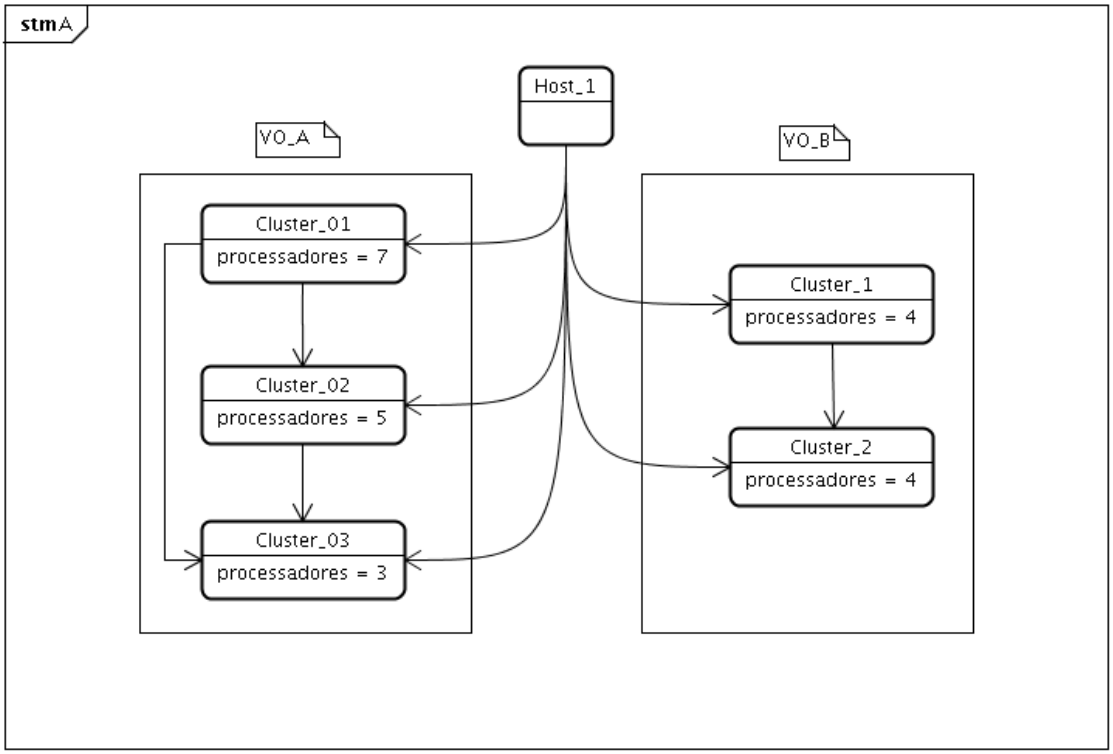

Figura 17: Ambiente multi-cluster

2.6. O Simulador

No momento em que o trabalho teve início, o alocador dinâmico não estava configurado para suportar um ambiente *multi-cluster* de teste e/ou produção. Como demonstrado em [FERREIRA et al., 2009], foi utilizado um simulador para o desenvolvimento do alocador dinâmico. Assim, este trabalho reutilizou este simulador, fazendo-se necessário a implementação de outra simulação que contemplasse o algoritmo de ordenação paralela por amostragem regular.

De acordo com [SimGrid, 2009], o SimGrid é "um *toolkit* que provê as principais funcionalidades para simulação de aplicações distribuídas em ambientes heterogêneos distribuídos. O objetivo principal do projeto é facilitar a pesquisa na área de escalonamento de aplicações distribuidas e paralelas em plataformas de computação distribuída e paralela indo de simples redes de *workstations* a grades computacionais."

O SimGrid é formado, portanto, pelos seguintes componentes e ilustrados na figura 7:

- XBT (*eXtended Bundle of Tools*), que consiste em ferramentas básicas e de estruturas de dados.
- Surf é a camada que contém as principais funcionalidades para simular a plataforma.
- SimDag, biblioteca utilizada pelo usuário para simular escalonamento centralizado para tarefas paralelas baseadas em modelos DAG (*Direct Acyclic Graphs*).
- SMPI permite que aplicações escritas utilizando MPI (*Message Passing Interface*) possam ser simuladas sem que o código seja alterado.
- MSG (*MetaSimGrid*) recomendado apenas para o ambiente de simulação. Projetado para simular aplicações distribuídas.
- GRAS (*Grid Reality And Simulation*) permite que um mesmo código de uma aplicação distribuída seja utilizado tanto para simulação quanto para produção em um sistema real.
- AMOK (*Advanced Metacomputing Overlay Kit*) oferece serviços de alto nível baseados no GRAS.

Um aspecto referente ao GRAS que deve ser mencionado é a possibilidade de reaproveitar o código escrito tanto para gerar uma simulação quanto para gerar código para produção. Isso resprenta uma vantagem sobre os simuladores tradicionais, mesmo contra uma biblioteca oferecida no mesmo pacote: MSG.

As figuras 8 e 9 mostram claramente esta diferença. Enquanto aquela representa um simulador convencional, em que um código escrito durante a pesquisa utilizando um simulador deverá ser reescrito caso o mesmo tenha o objetivo de executar em um ambiente de produção, esta mostra o esquema utilizado pelo GRAS, o mesmo código é utilizado tanto para a pesquisa quanto para a produção. Portanto, uma vez que a fase de pesquisa esteja concluída com os resultados esperados, o mesmo código estará pronto para produção, o que otimiza esforços de equipe.

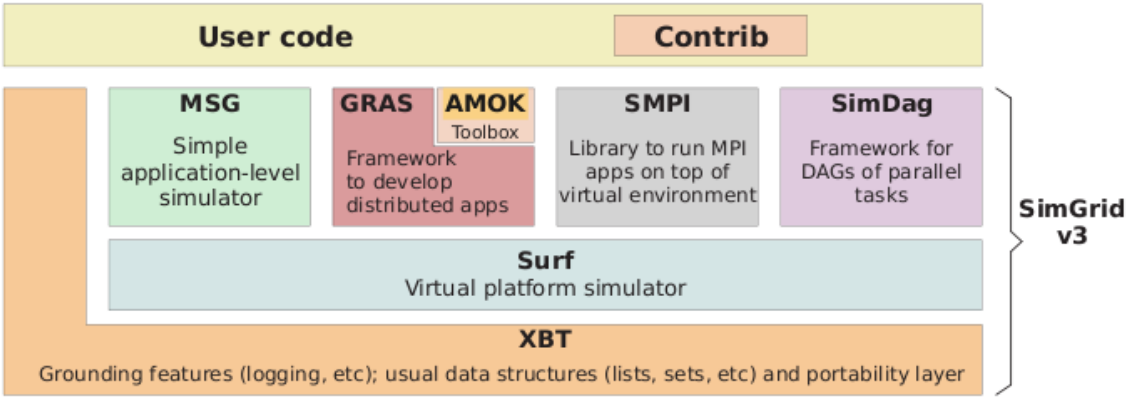

Figura 18: SimGrid

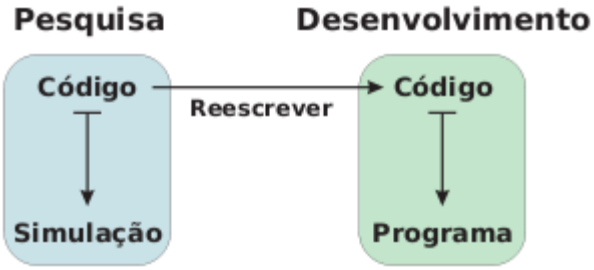

Figura 19: Simuladores Convencionais

Figura 20: GRAS

3. Testes

Pertinente ao que foi proposto, foram realizados alguns testes que consistiram em executar a aplicação cliente para requisitar a alocação de 4 recursos. Com o resultado do alocador foi preparado uma simulação para executar o *job* que ordenará uma lista de números inteiros contendo 1000, 2000, 3000, 4000 e 5000 elementos nestes recursos.

Primeiramente foi necessário configurar o ambiente. Para isso, dois arquivos OWL foram criados utilizando um software de autoria de ontologias, Protégé. A configuração destes arquivos segue as informações apresentadas nas tabelas da sessão 3.5. Estes arquivos representam informações semânticas dos recursos e serviços que as organizações virtuais estão publicando e permitem ao alocador fazer uma primeira filtragem, de forma estática, sobre os recursos que serão alocados, como demostrado na figura 1.

Em seguida, foi preparado simulações para os testes de submissão. Para isso, foi reproduzido o arquivo de configuração da plataforma virtual adaptando-o às necessidades do algoritmo de ordenação paralela por amostragem regular e preparado os arquivos que representam a submissão (*deployment*). Com os arquivos de configuração e o servidor esperando por requisições do cliente, o próximo passo é executar o mesmo.

O cliente realiza uma requisição por recursos que contenham pelo menos quatro processadores. O alocador, então, retorna quais os recursos que satisfazem a requisição. No caso deste teste, foi retornado quatro clusters que atendem à requisição. Como o Cluster_03, mostrado na figura 6, tinha menos de quatro processadores ele foi eliminado já na etapa da seleção estática. O alocador, então, informa o nome do recurso, o total de processadores de cada recurso e a carga nos meios de comunicação. Estas informações são apresentadas na tabela a seguir.

recurso	processadores	carga na comunicação
Cluster_01	7	0,018164
Cluster_02	5	0,050277
Cluster_1	4	0,050610
Cluster_2	4	37,898322

A partir disso, a aplicação cliente escolhe o Cluster_01. Esta escolha é realizada por apresentar a menor carga na comunicação. Em seguida, a mesma aplicação requisita a submissão do *job*. Esta submissão é realizada executando o simulador com a configuração adequada de alocação(*deployment*).

A escolha do critério de menor carga na comunicação resultou, nesta situação, em um recurso onde o poder de processamento de cada processador é o menor e em recursos ociosos. Assim, o *job* foi enviado para ser executado em quatro processos deixando três ociosos. A figura 10 mostra o performance do algoritmo durante a execução deste exemplo. Portanto, a submissão do *job* ao Cluster_01, que contém sete processadores, para ser executado em quatro deles com a tarefa de ordenar 1000, 2000, 3000, 4000 e 5000 elementos demorou 0.050184s, 0.051611s, 0.053141s, 0.054722s, 0.056146s respectivamente.

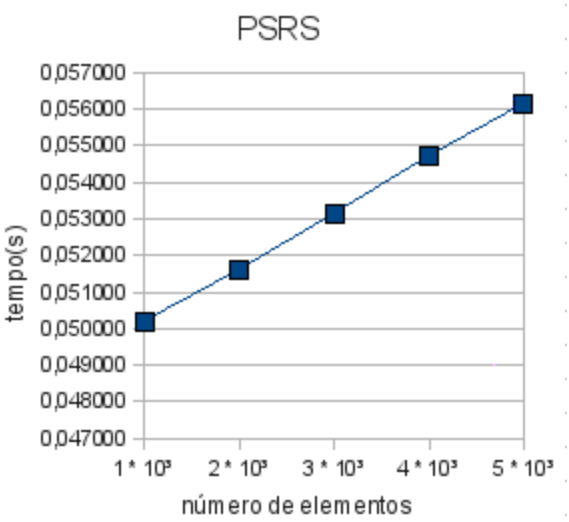

Figura 21: Tempos de execução do psrs no Cluster_01 utilizando 4 processadores.

4. Conclusões

Otimizar a utilização de recursos em configurações de grades computacionais é uma área de pesquisa à qual se inclui os estudos realizados pelo LAPESD. Estudos esses que resultaram em um alocador dinâmico com a possibilidade de coletar informações do estado atual do ambiente distribuído e fazer sugestões de seleção de recursos. Porém, até então, não existia uma aplicação que utilizasse este alocador com esta finalidade. Por isso, a proposta deste artigo foi desenvolver esta aplicação.

Primeiro foi necessário a definição de uma aplicação que serviria como propósito para ser executada nos recursos selecionados, isso resultou na implementação de um ordenador baseado no algoritmo de ordenação paralela por amostragem regular. Depois foi necessário formular a requisição em forma de ontologia persistida na forma de arquivo OWL. Em seguida a aplicação cliente foi implementada para se comunicar com o alocador via RMI. Por fim, o ordenador foi submetido, via simulação, aos recursos selecionados.

Esta trabalho representa um primeiro passo na utilização deste alocador, abrindo caminho para o desenvolvimento de outras aplicações. Enquanto que aqui foi utilizado a heurística de menor carga nos meios de comunicação para selecionar os recursos, outras heurísticas podem ser estudadas.

Referências

Ferreira, D. J.; Silva, A. P. C.; Dantas, M. A. R.; Qin, J.; Bauer, M. A. Toward Resource Management in Multi-Cluster Grid Configurations through an Ontology-Fuzzy Approach. 2009

SimGrid, http://simgrid.gforge.inria.fr/, disponível em Junho de 2009.

6.2 Código Fonte

```
package app_client;


import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.rmi.RemoteException;
import java.util.ArrayList;
import java.util.List;
import java.util.StringTokenizer;

import utilidade.MSGCliente;
import utilidade.MSGSelecionador;
import utilidade.ServerRMISelecionador;
import cliente.ClienteFactory;

public class PSRSClient {

	private GridRequestOntology ontology;
	private ServerRMISelecionador server;
	private JobDispatcher dispatcher;

	private String requisicao;
	private String dynamics;

	private List<Resource> resources;
	private Resource chosenOne;

	public PSRSClient(int numberOfCPUs) {
		// gera uma requisiÃ§Ã£o na forma de ontologia.
		this.ontology = new GridRequestOntology("requisicao/grid_request.owl");
		// gera um arquivo chamado request.owl
		this.requisicao = ontology.generate(Integer.toString(numberOfCPUs));
	}

	public static void main(String args[]) {
		PSRSClient cliente = new PSRSClient(4);
		cliente.connect(args[0], args[1]);
		cliente.enviaPedido(cliente.getRequisicao());
		cliente.submeteAplicacao();
		cliente.imprimeResultado();
	}

	public void connect(String host, String port) {
		this.server = ClienteFactory.getServer(host + ":" + port);
	}

	public String getRequisicao() {
		return this.requisicao;
	}

	private MSGSelecionador selecionarRecursos(MSGCliente msg) {
		MSGSelecionador msgSelecionador = null;
		try {
```

```
                msgSelecionador = server.selectResources(msg);
        } catch (RemoteException e) {
                e.printStackTrace();
        }
        return msgSelecionador;
}

private String getConteudoPedido(File arquivo) {
        FileReader reader;
        boolean ok = true;
        String linha = "";
        BufferedReader leitor;

        try {
                reader = new FileReader(arquivo);
                leitor = new BufferedReader(reader, 1 * 512 * 512);
                String aux = "";
                while (ok) {
                        aux = leitor.readLine();
                        if (aux != null) {
                                linha = linha + aux + "\n";
                        } else {
                                ok = false;
                        }
                } // while */
        } catch (IOException io_error) {
                System.out.println(io_error);
        }
        return linha;
}

public void enviaPedido(String fileName) {
        File arq = new File(fileName);
        String pedido = this.getConteudoPedido(arq);

        System.out.println("Nome do arquivo do Pedido: " + arq.getName());

        MSGCliente msg = new MSGCliente();
        msg.setPedido(pedido);
        msg.setNomeArquivo(arq.getName());

        MSGSelecionador msgSel = this.selecionarRecursos(msg);
        this.dynamics = msgSel.getDynamicAnswer();

        this.resources = new ArrayList<Resource>();
        StringTokenizer list = new StringTokenizer(this.dynamics, "\n");
        int i = 0;
        while (list.hasMoreTokens()) {
                StringTokenizer resource = new StringTokenizer(list.nextToken()
                                .replace("%", ""), " ");
                resource.nextToken();
                String cluster = resource.nextToken();
                resource.nextToken();
                resource.nextToken();
                String cpu = resource.nextToken();
                resource.nextToken();
                resource.nextToken();
                String link = resource.nextToken();

                this.resources.add(new Resource(cluster, cpu, link));
```

```
                        // System.out.println(list.nextToken().replace("\n", ""));
                }

                choseResource();
        }

        public void choseResource() {
                System.out.println("choosing...");
                int min_idx = 0;
                double min_wld = resources.get(0).getLinkWorkLoad();
                for (int i = 1; i < resources.size() - 1; i++) {
                        min_wld = Math.min(min_wld, resources.get(i).getLinkWorkLoad());
                }
                for (int i = 0; i < resources.size() - 1; i++) {
                        if (resources.get(i).getLinkWorkLoad() == min_wld) {
                                min_idx = i;
                                break;
                        }
                }

                this.chosenOne = resources.get(min_idx);

                /*
                 * System.out.println("min: " + min_wld); System.out.println("idx: " +
                 * min_idx); System.out.println(dynamics);
                 */
        }

        public void submeteAplicacao() {
                dispatcher = new JobDispatcher(chosenOne);
                dispatcher.handle();
        }

        public void imprimeResultado() {
                System.out.println("A execuÃ§Ã£o do ordenador demorou "
                                + dispatcher.getTime() + " segundos");
        }
}


package app_client;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.xml.sax.SAXException;

import com.hp.hpl.jena.ontology.Individual;
import com.hp.hpl.jena.ontology.OntClass;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.OntProperty;
```

```
import com.hp.hpl.jena.ontology.ProfileRegistry;
import com.hp.hpl.jena.ontology.Restriction;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.vocabulary.XSD;
import com.sun.org.apache.xml.internal.serialize.XMLSerializer;

public class GridRequestOntology {
	private Document ontDocument = null;

	private Element minNumberOfCPULiteral;
	private Element ownerLiteral;
	private Element requestIDLiteral;

	public GridRequestOntology() {

	}

	public GridRequestOntology(int deprecated) {

		// namespace
		String requestNS = "http://www.owl-ontologies.com/Grid_Requests.owl#";
		// String xmlbase = "http://www.owl-ontologies.com/Grid_Requests.owl";

		// cria o modelo (grafo)
		OntModel model = ModelFactory
				.createOntologyModel(ProfileRegistry.OWL_LANG);
		model.setNsPrefix("", requestNS);
		// model.getWriter("RDF/XML-ABBREV").setProperty("xmlbase", xmlbase);
		// model.getWriter("RDF/XML-ABBREV").setProperty("relativeURIs", "");

		// cria classes
		OntClass descritionComputerSystem = model.createClass(requestNS
				+ "Description_Computer_System");
		OntClass request = model.createClass(requestNS + "Request");
		OntClass requirements = model.createClass(requestNS + "Requirements");

		// cria sub-classes
		OntClass requirementsCPU = model.createClass(requestNS
				+ "Requirements_CPU");
		OntClass requirementsFS = model.createClass(requestNS
				+ "Requirements_FS");
		OntClass requirementsMemory = model.createClass(requestNS
				+ "Requirements_Memory");
		OntClass requirementsOS = model.createClass(requestNS
				+ "Requirements_OS");

		// assossia sub-classes
		requirements.addSubClass(requirementsCPU);
		requirements.addSubClass(requirementsFS);
		requirements.addSubClass(requirementsMemory);
		requirements.addSubClass(requirementsOS);

		// cria atributos
		OntProperty requiredComputerSystem = model
				.createObjectProperty(requestNS + "description_computer_system");
		OntProperty requiredCPU = model.createObjectProperty(requestNS
				+ "required_cpu");
		OntProperty requiredFS = model.createObjectProperty(requestNS
				+ "required_fs");
		OntProperty requiredMemory = model.createObjectProperty(requestNS
```

```
                + "required_memory");
OntProperty requiredOS = model.createObjectProperty(requestNS
                + "required_os");

requiredComputerSystem.addDomain(request);
requiredComputerSystem.addRange(descritionComputerSystem);
requiredCPU.addDomain(descritionComputerSystem);
requiredCPU.addRange(requirementsCPU);
requiredFS.addDomain(descritionComputerSystem);
requiredFS.addRange(requirementsFS);
requiredMemory.addDomain(descritionComputerSystem);
requiredMemory.addRange(requirementsMemory);
requiredOS.addDomain(descritionComputerSystem);
requiredOS.addRange(requirementsOS);

// cria atributos literais
OntProperty fileSystemType = model.createDatatypeProperty(requestNS
                + "file_system_type");
OntProperty maxLoadCPU15min = model.createDatatypeProperty(requestNS
                + "max_load_cpu_15min");
OntProperty maxLoadCPU1min = model.createDatatypeProperty(requestNS
                + "max_load_cpu_1min");
OntProperty maxLoadCPU5min = model.createDatatypeProperty(requestNS
                + "max_load_cpu_5min");
OntProperty minClockSpeed = model.createDatatypeProperty(requestNS
                + "min_clock_speed");
OntProperty minFreeDisckSpace = model.createDatatypeProperty(requestNS
                + "min_free_disk_space");
OntProperty minNumberOfCPUs = model.createDatatypeProperty(requestNS
                + "min_number_of_cpus");
OntProperty minPhysicalMemoryAvailable = model
                .createDatatypeProperty(requestNS
                                + "min_physical_memory_available");
OntProperty minVirtualMemoryAvailable = model
                .createDatatypeProperty(requestNS
                                + "min_virtual_memory_available");
OntProperty numberOfCPUs = model.createDatatypeProperty(requestNS
                + "number_of_cpus");
OntProperty numberResourcesReturn = model
                .createDatatypeProperty(requestNS + "number_resources_return");
OntProperty osType = model
                .createDatatypeProperty(requestNS + "os_type");
OntProperty owner = model.createDatatypeProperty(requestNS + "owner");
OntProperty rankBy = model
                .createDatatypeProperty(requestNS + "rank_by");
OntProperty requestId = model.createDatatypeProperty(requestNS
                + "request_id");

fileSystemType.addDomain(requirementsFS);
fileSystemType.addRange(XSD.xstring);
maxLoadCPU15min.addDomain(requirementsCPU);
maxLoadCPU15min.addRange(XSD.xfloat);
maxLoadCPU1min.addDomain(requirementsCPU);
maxLoadCPU1min.addRange(XSD.xfloat);
maxLoadCPU5min.addDomain(requirementsCPU);
maxLoadCPU5min.addRange(XSD.xfloat);
minClockSpeed.addDomain(requirementsCPU);
minClockSpeed.addRange(XSD.xint);
minFreeDisckSpace.addDomain(requirementsFS);
minFreeDisckSpace.addRange(XSD.xfloat);
```

```
minNumberOfCPUs.addDomain(requirementsCPU);
minNumberOfCPUs.addRange(XSD.xint);
minPhysicalMemoryAvailable.addDomain(requirementsMemory);
minPhysicalMemoryAvailable.addRange(XSD.xint);
minVirtualMemoryAvailable.addDomain(requirementsMemory);
minVirtualMemoryAvailable.addRange(XSD.xint);
numberOfCPUs.addDomain(requirementsCPU);
numberOfCPUs.addRange(XSD.xint);
numberResourcesReturn.addDomain(request);
numberResourcesReturn.addRange(XSD.xint);
osType.addDomain(requirementsOS);
osType.addRange(XSD.xstring);
owner.addDomain(request);
owner.addRange(XSD.xstring);
rankBy.addDomain(request);
rankBy.addRange(XSD.xstring);// TODO tem que utilizar o DataRange?
requestId.addDomain(request);
requestId.addRange(XSD.xstring);
requestId.convertToFunctionalProperty();

// restriÃ§Ãµes para a classe Description_Computer_System
Restriction restrictionCPU = model.createRestriction(requiredCPU);
Restriction restrictionFS = model.createRestriction(requiredFS);
Restriction restrictionMemory = model.createRestriction(requiredMemory);
Restriction restrictionOS = model.createRestriction(requiredOS);

restrictionCPU.convertToMaxCardinalityRestriction(1);
restrictionFS.convertToMaxCardinalityRestriction(1);
restrictionMemory.convertToMaxCardinalityRestriction(1);
restrictionOS.convertToMaxCardinalityRestriction(1);

restrictionCPU.addSubClass(descritionComputerSystem);
restrictionFS.addSubClass(descritionComputerSystem);
restrictionMemory.addSubClass(descritionComputerSystem);
restrictionOS.addSubClass(descritionComputerSystem);

// restriÃ§Ãµes para a classe Request
Restriction restrictionRequestId = model.createRestriction(requestId);
Restriction restrictionOwner = model.createRestriction(owner);
Restriction restrictionRankBy = model.createRestriction(rankBy);
Restriction restrictionNumberResourcesReturn = model
                    .createRestriction(numberResourcesReturn);
Restriction restrictionDescriptionComputerSystem = model
                    .createRestriction(requiredComputerSystem);

restrictionRequestId.convertToCardinalityRestriction(1);
restrictionOwner.convertToCardinalityRestriction(1);
restrictionRankBy.convertToMaxCardinalityRestriction(1);
restrictionNumberResourcesReturn.convertToMaxCardinalityRestriction(1);
restrictionDescriptionComputerSystem
                    .convertToMaxCardinalityRestriction(1);

restrictionRequestId.addSubClass(request);
restrictionOwner.addSubClass(request);
restrictionRankBy.addSubClass(request);
restrictionNumberResourcesReturn.addSubClass(request);
restrictionDescriptionComputerSystem.addSubClass(request);

// RestriÃ§Ãµes para a classe Requirements_CPU
Restriction restrictionMaxLoadCPU15Min = model
```

```
                .createRestriction(maxLoadCPU15min);
Restriction restrictionMaxLoadCPU1Min = model
                .createRestriction(maxLoadCPU1min);
Restriction restrictionMaxLoadCPU5Min = model
                .createRestriction(maxLoadCPU5min);
Restriction restrictionMinClockSpeed = model
                .createRestriction(minClockSpeed);
Restriction restrictionMinNumberOfCPUs = model
                .createRestriction(minNumberOfCPUs);
Restriction restrictionNumberOfCPUs = model
                .createRestriction(numberOfCPUs);

restrictionMaxLoadCPU15Min.convertToMaxCardinalityRestriction(1);
restrictionMaxLoadCPU1Min.convertToMaxCardinalityRestriction(1);
restrictionMaxLoadCPU5Min.convertToMaxCardinalityRestriction(1);
restrictionMinClockSpeed.convertToMaxCardinalityRestriction(1);
restrictionMinNumberOfCPUs.convertToMaxCardinalityRestriction(1);
restrictionNumberOfCPUs.convertToMaxCardinalityRestriction(1);

restrictionMaxLoadCPU15Min.addSubClass(requirementsCPU);
restrictionMaxLoadCPU1Min.addSubClass(requirementsCPU);
restrictionMaxLoadCPU5Min.addSubClass(requirementsCPU);
restrictionMinClockSpeed.addSubClass(requirementsCPU);
restrictionMinNumberOfCPUs.addSubClass(requirementsCPU);
restrictionNumberOfCPUs.addSubClass(requirementsCPU);

// RestriÃ§Ãµes para a classe Requirements_FS
Restriction restrictionFileSystemType = model
                .createRestriction(fileSystemType);
Restriction restrictionMinFreeDisckSpace = model
                .createRestriction(minFreeDisckSpace);

restrictionFileSystemType.convertToMaxCardinalityRestriction(1);
restrictionMinFreeDisckSpace.convertToMaxCardinalityRestriction(1);

restrictionFileSystemType.addSubClass(requirementsFS);
restrictionMinFreeDisckSpace.addSubClass(requirementsFS);

// RestriÃ§Ãµes para a classe Requirements_Memory
Restriction restrictionMinPhysicalMemoryAvailable = model
                .createRestriction(minPhysicalMemoryAvailable);
Restriction restrictionMinVirtualMemoryAvailable = model
                .createRestriction(minVirtualMemoryAvailable);

restrictionMinPhysicalMemoryAvailable
                .convertToMaxCardinalityRestriction(1);
restrictionMinVirtualMemoryAvailable
                .convertToMaxCardinalityRestriction(1);

restrictionMinPhysicalMemoryAvailable.addSubClass(requirementsMemory);
restrictionMinVirtualMemoryAvailable.addSubClass(requirementsMemory);

// RestriÃ§Ãµes para a classe Requirements_OS
Restriction restrictionOSType = model.createRestriction(osType);
restrictionOSType.convertToMaxCardinalityRestriction(1);

restrictionOSType.addSubClass(requirementsOS);

// criaÃ§Ã£o de indivÃduos seria interessante se for parametrizado.
// Criando indivÃduos
```

```
		// IndivÃduo para Requirements_CPU
		Individual individualRequirements_CPU = requirementsCPU
				.createIndividual(requestNS + "Requirements_CPU_2");
		individualRequirements_CPU.addProperty(minClockSpeed, 1800);
		individualRequirements_CPU.addProperty(minNumberOfCPUs, 2);// TODO foco

				// do

				// trabalho

		// IndivÃduo para Requirements_FS
		Individual individualRequirements_FS = requirementsFS
				.createIndividual(requestNS + "Requirements_FS_2");
		individualRequirements_FS.addProperty(minFreeDisckSpace, 20.0);

		// IndivÃduo para Requirements_Memory
		Individual individualRequirements_Memory = requirementsMemory
				.createIndividual(requestNS + "Requirements_Memory_2");
		individualRequirements_Memory.addProperty(minPhysicalMemoryAvailable,
				190);

		// IndivÃduo para Requirements_OS
		Individual individualRequirements_OS = requirementsOS
				.createIndividual(requestNS + "Requirements_OS_2");
		individualRequirements_OS.addProperty(osType, "Linux");

		// IndivÃduo para Description Computer System.
		Individual individualDescriptionComputerSystem = descritionComputerSystem
				.createIndividual(requestNS + "Description_Computer_System_2");
		individualDescriptionComputerSystem.addProperty(requiredCPU,
				individualRequirements_CPU);
		individualDescriptionComputerSystem.addProperty(requiredFS,
				individualRequirements_FS);
		individualDescriptionComputerSystem.addProperty(requiredMemory,
				individualRequirements_Memory);
		individualDescriptionComputerSystem.addProperty(requiredOS,
				individualRequirements_OS);

		// IndivÃduo para Request.
		Individual individualRequest = request.createIndividual(requestNS
				+ "Request_5");
		individualRequest.addProperty(requestId, "Request_5");
		individualRequest.addProperty(owner, "parra");
		individualRequest.addProperty(requiredComputerSystem,
				individualDescriptionComputerSystem);

		try {
			FileWriter file = new FileWriter("request.owl");
			// file.write("<?xml version=\"1.0\"?>\n");
			model.write(file, "RDF/XML-ABBREV");
			// model.write(System.out, "RDF/XML-ABBREV");
			file.write("\n");
			file.close();
		} catch (IOException e) {
			// Auto-generated catch block
			e.printStackTrace();
		}
}

public GridRequestOntology(String owlFileName) {
```

```
		File owlFile = new File(owlFileName);
		// File owlFile = new File("pedido_5.owl");
		DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
		factory.setValidating(false);
		factory.setNamespaceAware(true);
		DocumentBuilder builder = null;
		try {
			builder = factory.newDocumentBuilder();
			ontDocument = builder.parse(owlFile);
		} catch (ParserConfigurationException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (SAXException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

}

public String generate(String minNumberOfCPUProperty) {
		// Element root = ontDocument.getDocumentElement();
		// Element requirementsCPUIndividualElement = (Element)
		// root.getElementsByTagName("Requirements_CPU").item(0);
		// Element minNumberOfCPUsInstanceElement = (Element)
		// requirementsCPUIndividualElement.getElementsByTagName("min_number_of_cpus").item(0);
		// minNumberOfCPUsInstanceElement.setTextContent("5");/**Ã ▪ o bixo!!*/

		minNumberOfCPULiteral = (Element) ((Element) ontDocument
				.getDocumentElement().getElementsByTagName("Requirements_CPU")
				.item(0)).getElementsByTagName("min_number_of_cpus").item(0);

		minNumberOfCPULiteral.setTextContent(minNumberOfCPUProperty);

		ownerLiteral = (Element) ((Element) ontDocument.getDocumentElement()
				.getElementsByTagName("Request").item(0)).getElementsByTagName(
				"owner").item(0);

		ownerLiteral.setTextContent("parra");

		ownerLiteral = (Element) ((Element) ontDocument.getDocumentElement()
				.getElementsByTagName("Request").item(0)).getElementsByTagName(
				"request_id").item(0);

		ownerLiteral.setTextContent("request_2");

		XMLSerializer serializer = new XMLSerializer();
		try {
			serializer.setOutputByteStream(new FileOutputStream(
					"requisicao/request.owl"));
			serializer.serialize(ontDocument);
		} catch (FileNotFoundException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
```

```
                return "requisicao/request.owl";
        }
}

package app_client;

import java.util.StringTokenizer;

public class JobDispatcher {
        private Resource resource;
        private double time;

        public JobDispatcher(Resource chosenOne) {
                this.resource = chosenOne;
        }

        public void handle() {
                String line, out;
                out = "";

                if (resource.getName().equals("Cluster_01")) {
                        String envp[] = { "GRAS_ROOT=$HOME",
                                        "LD_LIBRARY_PATH=$HOME/lib/:$LD_LIBRARY_PATH" };
                        // "export GRAS_ROOT=$HOME; export LD_LIBRARY_PATH=$HOME/lib/:
$LD_LIBRARY_PATH";
                        String currentDirectory = "/home/leandro/jee/gras-app/quick_sort/4/";
                        String command = "psrs_simulator platform.xml deploy.xml";
                        try {
                                // Process p = Runtime.getRuntime().exec(command, envp,
                                // new File(currentDirectory));
                                // p.waitFor();
                                // System.out.println(p.exitValue());

                                // BufferedReader input = new BufferedReader(
                                // new InputStreamReader(p.getInputStream()));

                                /*
                                 * while (true) { line = input.readLine(); if (line != null) {
                                 * System.out.println(line); } else { break; } }
                                 */
                                /*
                                 * if (line != null && Integer.parseInt(line.trim()) == 1) {
                                 * while ((line = input.readLine()) != null) { int idx =
                                 * line.indexOf(" "); out = out + "..." + line.substring(0, idx)
                                 * + "  "; line = line.substring(idx + 1); idx =
                                 * line.indexOf(" "); out = out + "..." + line.substring(0, idx)
                                 * + "  "; } input.close(); }
                                 */
                                StringTokenizer result = new StringTokenizer(processReturn(),
                                                "\n");
                                while (result.hasMoreElements()) {
                                        line = result.nextToken();
                                        if (line.contains("Cluster_01:")
                                                        && line.contains("end time")) {
                                                line = line.replace("[", "");
                                                line = line.replace("]", "");
                                                result = new StringTokenizer(line, " ");
                                                result.nextToken();// cluster : function
                                                this.time = Double.parseDouble(result.nextToken());// time.
```

```
                                        }
                                }

                        } catch (Exception e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                        // catch (InterruptedException e) {
                        // // TODO Auto-generated catch block
                        // e.printStackTrace();
                        // }
                }
        }

        public String processReturn() {
                String out = "[Cluster_01:master:(1) 0.000000] [PSRS/INFO] begin time: 0.000000\n"
                                + "[Cluster_01_2:slave:(2) 0.000000] [PSRS/INFO] begin time: 0.000000s\n"
                                + "[Cluster_01_3:slave:(3) 0.000000] [PSRS/INFO] begin time: 0.000000s\n"
                                + "[Cluster_01_4:slave:(4) 0.000000] [PSRS/INFO] begin time: 0.000000s\n"
                                + "[Cluster_01_2:slave:(2) 0.052897] [PSRS/INFO] end time: 0.052897s\n"
                                + "[Cluster_01_2:slave:(2) 0.052897] [PSRS/INFO] elapse time: 0.052897s\n"
                                + "[Cluster_01_2:slave:(2) 0.052897] [gras/INFO] Exiting GRAS\n"
                                + "[Cluster_01_3:slave:(3) 0.054873] [PSRS/INFO] end time: 0.054873s\n"
                                + "[Cluster_01_3:slave:(3) 0.054873] [PSRS/INFO] elapse time: 0.054873s\n"
                                + "[Cluster_01_3:slave:(3) 0.054873] [gras/INFO] Exiting GRAS\n"
                                + "[Cluster_01:master:(1) 0.056146] [PSRS/INFO] end time: 0.056146s\n"
                                + "[Cluster_01:master:(1) 0.056146] [PSRS/INFO] elapse time: 0.056146s\n"
                                + "[Cluster_01:master:(1) 0.056146] [gras/INFO] Exiting GRAS\n"
                                + "[Cluster_01_4:slave:(4) 0.056146] [PSRS/INFO] end time: 0.056146s\n"
                                + "[Cluster_01_4:slave:(4) 0.056146] [PSRS/INFO] elapse time: 0.056146s\n"
                                + "[Cluster_01_4:slave:(4) 0.056146] [gras/INFO] Exiting GRAS\n";
                return out;
        }

        public double getTime() {
                return time;
        }
}


package app_client;

public class Resource {
        private String name;
        private int numberOfCPUs;
        private double linkWorkLoad;

        public Resource(String name, String numberOfCPUs, String linkWorkLoad) {
                this.name = name;
                this.numberOfCPUs = Integer.parseInt(numberOfCPUs);
                this.linkWorkLoad = Double.parseDouble(linkWorkLoad);
        }

        public String getName() {
                return this.name;
        }

        public int getNumberOfCPUs() {
                return numberOfCPUs;
        }
```

```
	public double getLinkWorkLoad() {
		return linkWorkLoad;
	}
}
```

```
#include <stdio.h>
#include <time.h>
#include <math.h>
#include <gras.h>

#define true  1;
#define false 0;

#define N_ESCRAVOS 3;

GRAS_DEFINE_TYPE(s_array, struct s_array {
        int size;
        int length;
        int *element GRAS_ANNOTE(size, length);
});

typedef struct s_array array_t;

XBT_LOG_NEW_DEFAULT_CATEGORY(PSRS,"PSRS");

void message_declaration(void);

//=============================================================//
//============================MESTRE===========================//
//=============================================================//

typedef struct{
        array_t sockets;
        array_t *sub_list; // local_elements que serão enviados na primeira fase
        array_t samples;
        array_t pivots;
        int done;

//   int process_in_CS;
  xbt_dynar_t waiting_queue;

        array_t *sorted_list;
        int received;
} master_data_t;


void merge(int dst[], int *src1, int length_src1, int *src2, int length_src2);
void build(int array[], int length);
void split(int dst[], int *src, int begin, int end);
int register_slave_cb();
array_t join(array_t src1, array_t src2);

//-----
void sort(int array[], int length);
int  divide(int array[], int inicio, int fim);
void quicksort(int array[], int inicio, int fim);
void swap(int array[], int i, int j);

//-----
int streql(char *str1, char *str2){
        while((*str1 == *str2) && (*str1)){
                str1++;
                str2++;
  }
```

```
  return((*str1 == NULL) && (*str2 == NULL));
}

//=========Parallel Sorting by Regular Sampling================//

//============================FASE 1============================//
// distribuir uniformemente n elementos a p processadores.
// em cada processador ordenar os (n/p) elementos por quick sort.
// selecionar em cada p processador (p - 1) amostras
// nas posições (n/p)/p e enviar ao coordenador.
//============================MUTEX=============================//
/*
int server_request_cb(gras_msg_cb_ctx_t ctx, void *payload) {
  master_data_t *globals=(master_data_t*)gras_userdata_get();
  gras_socket_t s = gras_msg_cb_ctx_from(ctx);


  if (globals->process_in_CS) {
    xbt_dynar_push(globals->waiting_queue, &s);
  } else {
                globals->process_in_CS = 1;
                gras_msg_send(s, "grant", NULL);
  }
  return 0;
}

int server_release_cb(gras_msg_cb_ctx_t ctx, void *payload) {
  master_data_t *globals=(master_data_t*)gras_userdata_get();

  if (xbt_dynar_length(globals->waiting_queue)) {
    gras_socket_t s;
    xbt_dynar_pop(globals->waiting_queue, &s);

    gras_msg_send(s, "grant", NULL);
  } else {
    globals->process_in_CS = 0;
  }

  return 0;
}
*/

// barreira.
int request_cb(gras_msg_cb_ctx_t ctx, void *payload){
        master_data_t *globals = (master_data_t*)gras_userdata_get();
        gras_socket_t s = gras_msg_cb_ctx_from(ctx);

        xbt_dynar_push(globals->waiting_queue, &s);

//      INFO0("POE NA FILA DE ESPERA");
//      INFO1("FILA: %d", xbt_dynar_length(globals->waiting_queue));

        if(xbt_dynar_length(globals->waiting_queue) == globals->sockets.size){
                int i = 1;
                while (xbt_dynar_length(globals->waiting_queue)) {
                        DEBUG1("release: %d", i);
                        i++;
                //gras_socket_t s;
                xbt_dynar_pop(globals->waiting_queue, &s);
```

```
                gras_msg_send(s, "grant", NULL);
                }
        }

        return 0;
}
//==========================================================//
//==========================================================//
int register_slave_cb(gras_msg_cb_ctx_t ctx, void *payload){
        master_data_t *slaves = (master_data_t *)gras_userdata_get();
        if(slaves->sockets.size < slaves->sockets.length){
                slaves->sockets.element[slaves->sockets.size] = gras_msg_cb_ctx_from(ctx);
                VERB2("Slave %s at port %d registered.",
                                gras_socket_peer_name(slaves->sockets.element[slaves->sockets.size]),
                                gras_socket_peer_port(slaves->sockets.element[slaves->sockets.size]));
                slaves->sockets.size++;
        }else{
                VERB0("Registering refused");
        }
        return 0;
}

int quick_sort_return_cb(gras_msg_cb_ctx_t ctx, void *payload){
        INFO0("rpc return cb");
        return 0;
}

/*TODO FASE 2*/
// final da primeira fase. início da segunda
// receive_samples_cb
int local_samples_cb(gras_msg_cb_ctx_t ctx, void *payload){
        array_t *list = (array_t*)payload;

        master_data_t *slaves = (master_data_t *)gras_userdata_get();

        int i;
        for(i = 0; i < list->size; i++){
                slaves->samples.element[slaves->samples.size] = list->element[i];
                slaves->samples.size++;
        }


        if(slaves->samples.size == slaves->samples.length)   {
                slaves->done = true;
        }

        xbt_free(list->element);
/*
        //=============
        int j = 0;
        INFO1("printing %d elements", list->size);
        printf("[");
        while(j < list->size -1){
                printf("%d, ", list->element[j]);
                j++;
        }
        printf("%d", list->element[j]);
        printf("]\n");
  //=============
*/
```

```
	return 0;
}

int finish_cb(gras_msg_cb_ctx_t ctx, void *payload){
	VERB0("finish");
	master_data_t *slaves = (master_data_t *)gras_userdata_get();
//	INFO1("received from %s", gras_socket_peer_name(gras_msg_cb_ctx_from(ctx)));

	int i;
	for(i = 0; i < slaves->sockets.size; i++){
/*		INFO2("socket[%s] == ctx[%s]",
				gras_socket_peer_name(slaves->sockets.element[i]),
				gras_socket_peer_name(gras_msg_cb_ctx_from(ctx)));
*/		if(streql(gras_socket_peer_name(slaves->sockets.element[i]),
gras_socket_peer_name(gras_msg_cb_ctx_from(ctx)))){
			break;
		}
	}

	array_t *remote_list = (array_t *)payload;
	slaves->sorted_list[i].length = remote_list->length;
	slaves->sorted_list[i].size = 0;
	slaves->sorted_list[i].element = xbt_malloc0(sizeof(int) * slaves->sorted_list[i].length);

	while(slaves->sorted_list[i].size < remote_list->size){
		slaves->sorted_list[i].element[slaves->sorted_list[i].size] = remote_list->element[slaves-
>sorted_list[i].size];
		slaves->sorted_list[i].size++;
	}

	slaves->received++;

	if(slaves->received == slaves->sockets.size){
		slaves->done = true;
	}

	return 0;
}

void print(array_t list){
	int i = 0;
//	INFO1("printing %d elements", list.size);
	printf("[");
	while(i < list.size -1){
		printf("%d, ", list.element[i]);
		i++;
	}
	printf("%d", list.element[i]);
	printf("]\n");
}

/*master can use internal timers to be sure that slaves are alive.*/
int master(int argc, char *argv[]) {
//   int n_slaves = 3;
	gras_socket_t mysock;	/* socket on which I listen */
  master_data_t *slaves;

	double time_begin, time_end;

	gras_init(&argc,argv);
```

```
        VERB0("Init GRAS.");

        time_begin = gras_os_time();
        INFO1("begin time: %f", time_begin);


        mysock = gras_socket_server(atoi(argv[1]));
        VERB1("Master on port %d.", gras_socket_my_port(mysock));

        slaves = gras_userdata_new(master_data_t);
  slaves->sockets.length  = N_ESCRAVOS;
        slaves->sockets.size    = 0;
        slaves->sockets.element = xbt_malloc0(sizeof(int) * slaves->sockets.length);

        // Fila da barreira.
        slaves->waiting_queue = xbt_dynar_new(sizeof(gras_socket_t), NULL);

        message_declaration();

        gras_cb_register("register",&register_slave_cb);
        gras_cb_register("local_samples",&local_samples_cb);
        gras_cb_register("finish",&finish_cb);

//   gras_cb_register("wait",&wait_cb);


  gras_cb_register("request",&request_cb);
//   gras_cb_register("release",&server_release_cb);


        VERB0("Master waiting for slaves.");
        int i;

        int s = N_ESCRAVOS;
        for(i = 0; i < s; i++){
                gras_msg_handle(-1);
        }
//      gras_msg_handleall(3);// espera * (unidade de medidas) por todas as mensagens.

        int n_slaves = slaves->sockets.size;// todos os escravos registrados!
        VERB1("Slaves registered: %d", n_slaves);

//      INFO1("Inicializa 'semáforo': %d.", 0);// semaforo.
//      slaves->process_in_CS = 0;
//      slaves->waiting_queue = xbt_dynar_new(sizeof(gras_socket_t), NULL);
        srand(0);
        // -- gera lista para teste
        // 1 * 10^6
        // 2 * 10^6
        // 3 * 10^6
        // 4 * 10^6 + 1
        // 5 * 10^6

        /*|*/     array_t list;                           /*|*/
        /*|*/list.length = 5 * pow(10, 3);                    /*|*/
        /*|*/list.size   = list.length;                   /*|*/
        /*|*/list.element = xbt_malloc0(sizeof(int) * list.length);/*|*/
  /*|*/                                   /*|*/
        /*|*/build(list.element, list.length);              /*|*/
        /*|*/VERB1("Gerado uma lista de %d elementos.", list.size);/*|*/
```

```
// /*|*/print(list);                              /*|*/

        VERB0("============BEGIN FASE 1============");
        int slice_size = (int)floor(list.length / n_slaves);
        DEBUG2("Dividindo a lista: %d X %d", n_slaves, slice_size);
//============================FASE 1============================//

        array_t slices[n_slaves];
        slaves->sub_list = &slices;

        for( i = 0; i < n_slaves - 1; i++){
                slaves->sub_list[i].length   = slice_size;
                slaves->sub_list[i].size     = slaves->sub_list[i].length;
                slaves->sub_list[i].element  = xbt_malloc0(sizeof(int) * slaves->sub_list[i].length);

                split(slaves->sub_list[i].element, list.element, i * slice_size, (i + 1) * slice_size - 1);
        }

        slaves->sub_list[i].length   = list.length - (i * slice_size);// tail, pode não ser partes iguais.
        slaves->sub_list[i].size     = slaves->sub_list[i].length;
        slaves->sub_list[i].element  = xbt_malloc0(sizeof(int) * slaves->sub_list[i].length);

        split(slaves->sub_list[i].element, list.element, i * slice_size, list.length - 1);

        xbt_free(list.element);

        VERB0("Lista dividida.");
        DEBUG2("Dividida a lista em %d sublistas \n \t\t\t\t\t\t de %d elementos, aproximadamente.",
                        slaves->sockets.size, slices[0].size);

        int n_samples = n_slaves - 1;// amostras individuais

        // prepara para receber as amostras.
        slaves->samples.size = 0;
        slaves->samples.length  = n_samples * n_slaves;
        slaves->samples.element = xbt_malloc0(sizeof(int) *             slaves->samples.length);

        VERB1("Master informa número de amostras necessárias para cada slave: %d.", n_samples);
        VERB0("Master envia a um pedaço da lista para cada slave.");

// INICIO.
        for(i = 0; i < n_slaves; i++){
                gras_msg_send(slaves->sockets.element[i],"set_n_samples", &n_samples);
                gras_msg_send(slaves->sockets.element[i],"initial_sort", &slaves->sub_list[i]);// (dst, msg, payload)
                xbt_free(slaves->sub_list[i].element);
        }
        VERB2("Master espera por %d x %d amostras.", n_samples, n_slaves);

        while(!slaves->done){
                gras_msg_handle(-1);
        }

        slaves->done = false;

        VERB0("============END FASE 1============");
        VERB0("============BEGIN FASE 2============");

        VERB0("Sublistas enviadas aos escravos \n \t\t\t\t\t\t para processamento distribuído.");
//============================FASE 2============================//
        // recebe as amostras.(recebe no callback)
```

```
	// ordena-as.
	VERB0("Amostras recebidas:");
//	print(slaves->samples);
	sort(slaves->samples.element, slaves->samples.length);
	VERB0("Amostras ordenadas:");
//	print(slaves->samples);
	VERB0("pivots ordenados:");
	//**********

	// seleciona pivots
	slaves->pivots.length   = n_samples;
	slaves->pivots.size     = slaves->pivots.length;
	slaves->pivots.element  = xbt_malloc0(sizeof(int) * slaves->pivots.length);

  // colhe amostras
	// seleciona pivôs
	int position = floor(slaves->samples.length/(n_samples + 1));

	VERB0("Seleciona pivôs globais.");
	for(i = 0; i < n_samples; i++){
		slaves->pivots.element[i] = slaves->samples.element[1 + (i * position)];
	}
	xbt_free(slaves->samples.element);

//	print(slaves->pivots);
	VERB0("Envia todos os sockets para os slaves.");
	VERB0("Envia os pivôs para os slaves.");
	VERB0("============END FASE 2============");
	VERB0("============BEGIN FASE 3============");
	DEBUG1("done: %d", slaves->done);

//	INFO1("array size: %d", slaves->sockets.size);

	slaves->sorted_list = xbt_malloc0(sizeof(array_t) * slaves->sockets.size);
	slaves->received = 0;

	for(i = 0; i < n_slaves; i++){
		gras_msg_send(slaves->sockets.element[i],"set_n_processors", &slaves->sockets);
		gras_msg_send(slaves->sockets.element[i],"pivots", &slaves->pivots);// (dst, msg, payload)
//		gras_msg_handle(-1);
	}

	xbt_free(slaves->pivots.element);

	for(i = 0; i < n_slaves; i++){
		gras_msg_handle(-1);
	}

	// barreira...
	while(!slaves->done){
		gras_msg_handle(-1);
	}

	VERB0("============END FASE 4============");
	VERB0("============BEGIN FASE 5============");
	//*****


	for(i = 0; i < slaves->sockets.size; i++){
//		print(slaves->sorted_list[i]);
```

```
	}

//========================== ============================//
/*
	INFO0("ZAMBAS");
//	gras_os_sleep(5);
	// merging slaves->sub_list[i]´s
	array_t sorted;
	sorted.size = sorted.length = 0; //TODO
	for(i = 0; i < slaves->sockets.size; i++){
		sorted = join(sorted, slaves->sub_list[i]);
	}

	print(sorted);
*/
/*	while(!slaves->done){
		gras_msg_handle(-1);
	}
*/

	for(i = 0; i < n_slaves; i++){
		gras_msg_send((gras_socket_t)slaves->sockets.element[i],"kill", NULL);// (dst, msg, payload)
	}

	int size = 0;
	for(i = 0; i < n_slaves; i++){
		size += slaves->sorted_list[i].size;
	}

	array_t final_list;
	final_list.length = size;
	final_list.size = size;
	final_list.element = xbt_malloc0(sizeof(int) * size);

	int j, k;
	for(i = 0,		k = 0; i < n_slaves; i++){
		for(j = 0; j < slaves->sorted_list[i].size; j++, k++){
			final_list.element[k] = slaves->sorted_list[i].element[j];
		}
		xbt_free(slaves->sorted_list[i].element);
	}
	xbt_free(slaves->sorted_list);

	time_end = gras_os_time();
	INFO1("end time: %fs", time_end);
	INFO1("elapse time: %fs", time_end - time_begin);

//	print(final_list);
	gras_exit();
	return 0;
}
void message_declaration(void) {
	gras_msgtype_declare("register", NULL);
	gras_msgtype_declare("set_n_samples", gras_datadesc_by_name("int"));
	gras_msgtype_declare("initial_sort", gras_datadesc_by_symbol(s_array));
//	gras_msgtype_declare("initial_sort", gras_datadesc_by_symbol(s_phase_one_data));// local_list, n_samples
	gras_msgtype_declare("kill", NULL);
	gras_msgtype_declare_rpc("quick_sort", gras_datadesc_by_symbol(s_array),
gras_datadesc_by_symbol(s_array));
	gras_msgtype_declare("local_samples", gras_datadesc_by_symbol(s_array));
```

```
	gras_msgtype_declare("set_n_processors", gras_datadesc_by_symbol(s_array));// sockets
	gras_msgtype_declare("pivots", gras_datadesc_by_symbol(s_array));
	gras_msgtype_declare("partition", gras_datadesc_by_symbol(s_array));
	gras_msgtype_declare("finish", gras_datadesc_by_symbol(s_array));

/*
	gras_msgtype_declare("wait", NULL);
	gras_msgtype_declare("notify", NULL);
*/

	gras_msgtype_declare("request", NULL);
	gras_msgtype_declare("grant", NULL);
//	gras_msgtype_declare("release", NULL);


	VERB0("Message type declared.");
}

/** realiza o merge de [src1 x src2] e armazena em [dst].*/
void merge(int dst[], int *src1, int length_src1, int *src2, int length_src2){
	int length = length_src1 + length_src2;
	int i = 0;
	int j = 0;
	int k = 0;
	while(i < length){
		if(j >= length_src1){
			while(k < length_src2){
				dst[i] = src2[k]; i++; k++;
			}
		} else if( k >= length_src2){
			while(j < length_src1){
				dst[i] = src1[j]; i++; j++;
			}
		} else{
			if(src1[j] <= src2[k]){
				dst[i] = src1[j]; j++;
			} else {
				dst[i] = src2[k]; k++;
			}
		}
		i++;
	}// while
}

array_t join(array_t src1, array_t src2){
	array_t result;
	result.length = result.size = src1.size + src2.size;
	result.element = xbt_malloc0(sizeof(int) * result.length);
	merge(result.element, src1.element, src1.size, src2.element, src2.size);
	return result;
}
void split(int dst[], int *src, int begin, int end){
	int length = end - begin;
	int i;
  for(i = 0; i <= length; i++, begin++){
		dst[i] = src[begin];
	}
}

void build(int array[], int length){
```

```
	VERB0("->build()");
	VERB1("array[%d]", length);
	int i = 0;
	while(i < length){
		array[i] = rand() % length;
		i++;
	}
}
//=============================================================//
//============================ESCRAVO==========================//
//=============================================================//
typedef struct{
	gras_socket_t mysocket;
	gras_socket_t tomaster;

	char *myname;

	array_t *local_list;
	array_t local_segment;

	int samples_length;
//	phase_one_data_t *po_data;
	array_t/*<gras_socket_t>*/ *slaves_addr;

	array_t/*<gras_socket_t>*/ worker_addr;
	array_t *local_partitions;

	array_t *received_partitions;
	int received_partitions_counter;

	int send_partition;

	array_t sorted;
	int killed;
} slave_data_t;

/*
void lock() {
	slave_data_t *slave_data = (slave_data_t *)gras_userdata_get();
  INFO0("send request");
  gras_msg_send(slave_data->tomaster,"request",NULL);
  gras_msg_wait(-1, "grant",NULL,NULL);
  INFO0("Granted by master");
}

void unlock() {
  INFO0("Release the token");
	slave_data_t *slave_data = (slave_data_t *)gras_userdata_get();
  gras_msg_send(slave_data->tomaster,"release",NULL);
}
*/

int quick_sort_rpc_cb(gras_msg_cb_ctx_t ctx, void *payload);
int slave_kill_cb(gras_msg_cb_ctx_t ctx, void *payload);

int quick_sort_rpc_cb(gras_msg_cb_ctx_t ctx, void *payload){
	VERB0("quick_sort_callback acionado.");

	array_t *list = (array_t*)payload;
```

```
	sort(list->element, list->length);

	gras_msg_rpcreturn(60,ctx,list);
	return 0;
}
//============================FASE 1============================//
int set_samples_cb(gras_msg_cb_ctx_t ctx, void *payload){
	slave_data_t *slave_data = (slave_data_t *)gras_userdata_get();
	slave_data->samples_length = *(int*)payload;
	DEBUG1("Informado o número de amostras que irá devolver. %d.",    slave_data->samples_length);

	return 0;
}

int initial_sort_cb(gras_msg_cb_ctx_t ctx, void *payload){
	VERB0("initial_sort_callback acionado.");
	slave_data_t *slave_data = (slave_data_t *)gras_userdata_get();

//	array_t *list = (array_t*)payload;

	slave_data->local_list = (array_t*)payload;

	VERB0("Recebe lista");
/*
	int j;
  //=============
	j = 0;
	printf("[");
	while(j < slave_data->local_list->size -1){
		printf("%d, ", slave_data->local_list->element[j]);
		j++;
	}
	printf("%d", slave_data->local_list->element[j]);
	printf("]\n");
  //=============
*/
	// ordena
//	sort(list->element, list->length);
	sort(slave_data->local_list->element, slave_data->local_list->length);

	VERB0("Lista ordenada");
/*
  //=============
	j = 0;
	printf("[");
	while(j < slave_data->local_list->size -1){
		printf("%d, ", slave_data->local_list->element[j]);
		j++;
	}
	printf("%d", slave_data->local_list->element[j]);
	printf("]\n");
  //=============
*/
	// por que prepara agora?

	slave_data->local_segment.size = 0;
//	slave_data->local_segment.length = list->length;
	slave_data->local_segment.length = slave_data->local_list->length;
	slave_data->local_segment.element = xbt_malloc0(sizeof(int) * slave_data->local_segment.length);
```

```
	while(slave_data->local_segment.size < slave_data->local_list->length){
//	while(slave_data->local_segment.size < list->length){
//		slave_data->local_segment.element[slave_data->local_segment.size] = list->element[slave_data->local_segment.size];
		slave_data->local_segment.element[slave_data->local_segment.size] = slave_data->local_list->element[slave_data->local_segment.size];
		slave_data->local_segment.size++;
	}

	DEBUG0("Colhendo amostras.");
  // colhe amostras
	int position = floor(slave_data->local_list->length/(slave_data->samples_length + 1));

	array_t samples;
	samples.length = slave_data->samples_length;
	samples.size =	samples.length;
	samples.element = xbt_malloc0(sizeof(int) * samples.length);

	int i;
	for(i = 0; i < slave_data->samples_length; i++){
		samples.element[i] = slave_data->local_list->element[(i + 1) * position];
	}

	DEBUG0("Amostras selecionadas e enviadas:");
/*
	j = 0;
	printf("[");
	while(j < samples.size -1){
		printf("%d, ", samples.element[j]);
		j++;
	}
	printf("%d", samples.element[j]);
	printf("]\n");
  //=============
*/
	// retorna ao mestre
	gras_msg_send(gras_msg_cb_ctx_from(ctx),"local_samples", &samples);
	return 0;
}


int set_processors_cb(gras_msg_cb_ctx_t ctx, void *payload){
	VERB0("set_processors_callback acionado.");

	array_t *sockets = (array_t*)payload;

	slave_data_t *slave_data = (slave_data_t *)gras_userdata_get();

	slave_data->worker_addr.size = 0;
	slave_data->worker_addr.length = sockets->length;
	slave_data->worker_addr.element = xbt_malloc0(sizeof(int) * slave_data->worker_addr.length);

	while(slave_data->worker_addr.size < slave_data->worker_addr.length){
		slave_data->worker_addr.element[slave_data->worker_addr.size] = sockets->element[slave_data->worker_addr.size];
		slave_data->worker_addr.size++;
	}
	xbt_free(sockets->element);
```

```
	slave_data->received_partitions = xbt_malloc0(sizeof(array_t) * slave_data->worker_addr.size);

/*
	slave_data->can_receive_partition[slave_data->worker_addr.length];
	int i;
	for(i = 0; i < slave_data->worker_addr.length;i ++){
		slave_data->can_receive_partition[i] = false;
	}
*/
	VERB1("Recebe o endereço de todos os %d processos colaboradores(workers).", slave_data-
>worker_addr.size);
	return 0;
}

int pivots_cb(gras_msg_cb_ctx_t ctx, void *payload){
	VERB0("pivots_callback acionado.");
	VERB0("Recebe os pivôs como referêcia para separar as partições.");
	//---------- recebe pivots
	array_t *pivots = (array_t *)payload;
	//---------- identifica indices das particoes
	slave_data_t *slave_data = (slave_data_t *)gras_userdata_get();
	// indices dos elementos não maiores que o pivot
	int index_size = slave_data->worker_addr.size - 1;
	int partition_index[index_size];

	int i;
	int j;
	for(i = 0, j = 0; i < slave_data->local_segment.size; i++){
		if(slave_data->local_segment.element[i] > pivots->element[j]){
			partition_index[j] = i - 1;
			j++; if(j == index_size) break;
		}
	}
	VERB0("Define os índices dos pivôs.");

	DEBUG2("[%d, %d]", partition_index[0], partition_index[1]);

	//---------- constroi as particoes
	VERB0("Separa as partições.");
	array_t partition_data[slave_data->worker_addr.size];
	/**
		há problemas na geração das partições:
		com p processos não há p - 1 partições.
		em uma partição que não conteria um
		elemento está entrando um!
		-- o problema pode está no meio ou na definição dos indices!
	*/

	//inicio
	partition_data[0].length  = partition_index[0] + 1;
	partition_data[0].size    = partition_data[0].length;
	partition_data[0].element = xbt_malloc0(sizeof(int) * partition_data[0].length);

	//meio
	for(i = 1; i < index_size; i++){
		partition_data[i].length  = (partition_index[i] - partition_index[i - 1]);
		partition_data[i].size    = partition_data[i].length;
		partition_data[i].element = xbt_malloc0(sizeof(int) * partition_data[i].length);
	}
	//fim
```

```
	partition_data[i].length  = (slave_data->local_segment.length - 1) - partition_index[i - 1];
	partition_data[i].size    = partition_data[i].length;
	partition_data[i].element = xbt_malloc0(sizeof(int) * partition_data[i].length);

	int k;
	i = 0;
	for( k = 0; k < slave_data->worker_addr.size - 1; k++){
		j = 0;
		while(i <= partition_index[k]){
			partition_data[k].element[j] = slave_data->local_segment.element[i];
			i++; j++;
		}
	}

	j = 0;
	while(i < slave_data->local_segment.length){
		partition_data[k].element[j] = slave_data->local_segment.element[i];
		i++; j++;
	}

	slave_data->local_partitions = xbt_malloc0(sizeof(array_t) * slave_data->worker_addr.size);

	for(i = 0; i < slave_data->worker_addr.size; i++){
		slave_data->local_partitions[i].length = partition_data[i].length;
		slave_data->local_partitions[i].size = partition_data[i].size;
		slave_data->local_partitions[i].element = xbt_malloc0(sizeof(int) * slave_data-
>local_partitions[i].length);
		for(j = 0; j < slave_data->local_partitions[i].size; j++){
			slave_data->local_partitions[i].element[j] = partition_data[i].element[j];
		}
		xbt_free(partition_data[i].element);
	}

	VERB0("Partições separadas:");
	VERB1("%s", gras_os_myname());
/*
  //=============
	for( k = 0; k < slave_data->worker_addr.size; k++){
		j = 0;
		printf("[");
		while(j < slave_data->local_partitions[k].size -1){
			printf("%d, ", slave_data->local_partitions[k].element[j]);
			j++;
		}
		printf("%d", slave_data->local_partitions[k].element[j]);
		printf("]\n");
	}
  //=============
*/
	slave_data->send_partition = true;

//	gras_os_sleep(0);

	return 0;
	// fim do pivots.
/*


	array_t partitions[slave_data->worker_addr.size];
	slave_data->received_partitions = &partitions;
```

```
	VERB0("Inicializa e envia partições.");

	INFO0("Envia as partições");


	INFO1("counter: %d.", slave_data->received_partitions_counter);
	INFO1("My name: %s", gras_os_myname());

	// pergunta ao mestre se pode continuar.

	if(!streql(gras_os_myname(),
					gras_socket_peer_name(slave_data->worker_addr.element[slave_data-
>worker_addr.size - 1]))){
		INFO0("######lock()#####");
		lock();
	}else{
		unlock(); unlock(); unlock();
	}

	//---------- envia as particoes
	for(i = 0; i < slave_data->worker_addr.size; i++){
//		INFO1("loop[%d]", i);
//		INFO2("slave_data->worker_addr.element[%d]: %s", i, gras_socket_peer_name(slave_data-
>worker_addr.element[i]));
		if(streql(gras_os_myname(), gras_socket_peer_name(slave_data->worker_addr.element[i]))){
			slave_data->received_partitions[slave_data->received_partitions_counter].length  =
partition_data[i].length;
			slave_data->received_partitions[slave_data->received_partitions_counter].size    = 0;
			slave_data->received_partitions[slave_data->received_partitions_counter].element =
xbt_malloc0(sizeof(int) *
						slave_data->received_partitions[slave_data-
>received_partitions_counter].length);

			INFO0("Partition data.");
			//==============
			j = 0;
			printf("[");
			while(j < partition_data[i].size - 1){
				printf("%d, ", partition_data[i].element[j]);
				j++;
			}
			printf("%d", partition_data[i].element[j]);
			printf("]\n");
			//==============

			int counter;
			for(counter = 0; counter < partition_data[i].size; counter++){
				slave_data->received_partitions[slave_data-
>received_partitions_counter].element[counter] =
					partition_data[i].element[counter];
				slave_data->received_partitions[slave_data->received_partitions_counter].size++;
			}

			INFO0("Partition assigned to me.");

			//==============
			j = 0;
			printf("[");
```

```
                        while(j < slave_data->received_partitions[slave_data->received_partitions_counter].size -1 )
{
                                printf("%d, ", slave_data->received_partitions[slave_data-
>received_partitions_counter].element[j]);
                                j++;
                        }
                        printf("%d", slave_data->received_partitions[slave_data-
>received_partitions_counter].element[j]);
                        printf("]\n");
                        //==============

                        slave_data->received_partitions_counter++;
                }else{
                        INFO1("Partition sended to %s.", gras_socket_peer_name(slave_data-
>worker_addr.element[i]));

                        //==============
                        j = 0;
                        printf("[");
                        while(j < partition_data[i].size - 1){
                                printf("%d, ", partition_data[i].element[j]);
                                j++;
                        }
                        printf("%d", partition_data[i].element[j]);
                        printf("]\n");
                        //==============

                        // tem que esperar que os outros rodem a fase 3 antes.
                        gras_msg_send((gras_socket_t)slave_data->worker_addr.element[i],"partition",
&partition_data[i]);
                }
        }

        INFO0("============END FASE 3============");
        return 0;
*/
}
//      INFO0("ZAMBAS");
int get_partition_cb(gras_msg_cb_ctx_t ctx, void *payload){
        VERB1("received from: %s", gras_socket_peer_name(gras_msg_cb_ctx_from(ctx)));

        VERB0("GET PARTITION");

        slave_data_t *slave_data = (slave_data_t *)gras_userdata_get();

        array_t *received = (array_t*)payload;

        DEBUG1("counter: %d", slave_data->received_partitions_counter);

        VERB1("received_partitions[%d].", slave_data->received_partitions_counter);

        VERB1("received->length: %d", received->length);

        slave_data->received_partitions[slave_data->received_partitions_counter].length  = received->length;
        slave_data->received_partitions[slave_data->received_partitions_counter].size    = 0;

        slave_data->received_partitions[slave_data->received_partitions_counter].element =
                                xbt_malloc0(sizeof(int) * slave_data->received_partitions[slave_data-
>received_partitions_counter].length);
```

```
	int i;
	for(i = 0; i < received->size; i++){
		slave_data->received_partitions[slave_data->received_partitions_counter].element[i] = received->element[i];
		slave_data->received_partitions[slave_data->received_partitions_counter].size++;
	}

	slave_data->received_partitions_counter++;
/*
	if(slave_data->received_partitions_counter > 0){
		INFO0("merge!");
		array_t merge = slave_data->received_partitions[slave_data->received_partitions_counter - 1];
		slave_data->sorted = join(slave_data->sorted, merge);
		print(slave_data->sorted);
	}
*/

	DEBUG1("received_partitions_counter: %d", slave_data->received_partitions_counter);

	if(slave_data->received_partitions_counter >= slave_data->worker_addr.size){
		for(i = 0; i < slave_data->received_partitions_counter; i++){
			array_t merged = slave_data->received_partitions[i];
			slave_data->sorted = join(slave_data->sorted, merged);
		}

//		print(slave_data->sorted);
		gras_msg_send(slave_data->tomaster, "finish", &slave_data->sorted);
	}

/*
  //=============
	int j = 0;
	INFO1("printing %d elements", received->size);
	printf("[");
	while(j < received->size -1){
		printf("%d, ", received->element[j]);
		j++;
	}
	printf("%d", received->element[j]);
	printf("]\n");
  //=============
*/
/*
		slave_data->received_partitions[i].length  = slave_data->local_segment.size;
		slave_data->received_partitions[i].size    = 0;
		slave_data->received_partitions[i].element = xbt_malloc0(sizeof(int) * slave_data->received_partitions[i].length);
*/
	return 0;
}

int slave_kill_cb(gras_msg_cb_ctx_t ctx, void *payload){
	slave_data_t *slave_data = (slave_data_t *)gras_userdata_get();
	slave_data->killed = true;
	VERB0("kill msg received.");
	return 0;
}

int set_sorted_partitions(){
	return 0;
```

```
}

// divide quantas vezes for necessário e manda para os escravos
int slave(int argc, char *argv[]) {
//          gras_socket_t mysock;       /* socket on which I listen */
            gras_socket_t tomaster;  /* socket used to write to the slaves */
            slave_data_t *slave_data;

            double time_begin, time_end;

            gras_init(&argc,argv);
            VERB0("Init GRAS");
            time_begin = gras_os_time();
            INFO1("begin time: %fs", time_end);

            slave_data = gras_userdata_new(slave_data_t);
            slave_data->killed = false;
            slave_data->send_partition = false;
            slave_data->received_partitions_counter = 0;

            gras_socket_t socket = gras_socket_server_range(1024, 10000, 0, 0); // (minport, maxport, buf_size,
measurement)
            VERB1("My name: %s.", gras_os_myname());
            VERB1("time %d", gras_os_time());

            slave_data->mysocket = xbt_malloc0(sizeof(gras_socket_t));
            slave_data->mysocket = socket;

            VERB1("Slave listening at port %d.", gras_socket_my_port(slave_data->mysocket));

//          gras_os_sleep(1.5); /* sleep 1 second and half */

            tomaster = gras_socket_client(argv[1], atoi(argv[2]));// (host, port)
            slave_data->tomaster = tomaster;

            message_declaration();

            gras_cb_register("quick_sort",&quick_sort_rpc_cb);
            gras_cb_register("kill",&slave_kill_cb);

            gras_cb_register("set_n_samples",&set_samples_cb);
            gras_cb_register("initial_sort",&initial_sort_cb);

            gras_cb_register("set_n_processors",&set_processors_cb);
            gras_cb_register("pivots",&pivots_cb);

            gras_cb_register("partition",&get_partition_cb);

            gras_msg_send(tomaster,"register", NULL);// (dst, msg, payload)

            VERB0("Daemon state.");
            while(!slave_data->send_partition){
                        gras_msg_handle(-1);
            }
            VERB0("============END FASE 3============");
            VERB0("============BEGIN FASE 4============");
            // prepara para receber as partições.
            int i;
            for(i = 0; i < slave_data->worker_addr.size; i++){
                        // indice da partição local.
```

```
                if(streql(gras_os_myname(), gras_socket_peer_name(slave_data->worker_addr.element[i]))){
                        break;
                }
        }

        slave_data->received_partitions[slave_data->received_partitions_counter].length =

                        slave_data->local_partitions[i].length;
        slave_data->received_partitions[slave_data->received_partitions_counter].size  = 0;

        slave_data->received_partitions[slave_data->received_partitions_counter].element =
                xbt_malloc0(sizeof(int) * slave_data->received_partitions[slave_data-
>received_partitions_counter].length);

        int j;
        for(j = 0; j < slave_data->local_partitions[i].size; j++){
                slave_data->received_partitions[slave_data->received_partitions_counter].element[j] =
                        slave_data->local_partitions[i].element[j];
                slave_data->received_partitions[slave_data->received_partitions_counter].size++;
        }
/*
        //=============
        VERB1("received: %d from self.", slave_data->received_partitions[slave_data-
>received_partitions_counter].size);
        j = 0;
        printf("[");
        while(j < slave_data->received_partitions[slave_data->received_partitions_counter].size - 1){
                printf("%d, ", slave_data->received_partitions[slave_data->received_partitions_counter].element[j]);
                j++;
        }
        printf("%d", slave_data->received_partitions[slave_data->received_partitions_counter].element[j]);
        printf("]\n");
        //=============
*/
        slave_data->received_partitions_counter++;

        DEBUG0("WAIT");
        gras_msg_send(tomaster,"request",NULL);
  gras_msg_wait(-1, "grant",NULL,NULL);
        DEBUG0("NOTIFIED");

        for(i = slave_data->worker_addr.size - 1; i >= 0; i--){
                if(!streql(gras_os_myname(), gras_socket_peer_name(slave_data->worker_addr.element[i]))){
                        gras_socket_t connection = gras_socket_client(
                                        gras_socket_peer_name(slave_data->worker_addr.element[i]),
                                        gras_socket_peer_port(slave_data->worker_addr.element[i]));
                        gras_msg_send(connection,"partition", &slave_data->local_partitions[i]);
                }
        }



/*
        for(i = 0; i < slave_data->worker_addr.size; i++){
                if(!streql(gras_os_myname(), gras_socket_peer_name(slave_data->worker_addr.element[i]))){
                        gras_socket_t connection = gras_socket_client(
                                        gras_socket_peer_name(slave_data->worker_addr.element[i]),
                                        gras_socket_peer_port(slave_data->worker_addr.element[i]));
                        INFO0("SEND PARTITION");
//                      gras_os_sleep(0.5);
```

```
                        gras_msg_send(connection,"partition", &slave_data->local_partitions[i]);
                }
        }
*/
        while(!slave_data->killed){
                gras_msg_handle(-1);
        }
        VERB2("Register msg sended to %s at port %d.", gras_socket_peer_name(tomaster),
gras_socket_peer_port(tomaster));
        time_end = gras_os_time();
        INFO1("end time: %fs", time_end);
        INFO1("elapse time: %fs", time_end - time_begin);
        gras_exit();
        return 0;
}

void sort(int array[], int length){
        DEBUG0("->sort()");
        quicksort(array,0,length - 1);
}

/**
        Distribui o valores no array onde o pivot terminarÃ¡ na sua posiÃ§Ã£o
         correta e os valores menores que ele estarÃ£o Ã  esquerda do pivot e
        os maiores Ã  direita do pivot.
*/
int divide(int array[], int inicio, int fim){
        //INFO0("  ->devide()");
        //int idx_pivot = rand() % fim;         // pivot randÃ´mico.
        //swap(array, inicio, idx_pivot); // pivot Ã© posicionado no inicio.
        int esq = inicio;
        int dir = fim;

        //INFO2("[%d,%d])", inicio, fim);

        int idx_pivot = inicio;
        //INFO1("            idx_pivot: %d", idx_pivot);
        inicio++;
        //INFO1("inicio++(%d)", inicio);
        while(fim >= inicio){
                while(array[idx_pivot] >= array[inicio]){
                        inicio++;
                //      INFO1("inicio++(%d)", inicio);
                }
                while( array[idx_pivot] < array[fim]){
                        fim--;
                        //INFO1("fim--(%d)", fim);
         }
                if(fim < inicio){
                  //    INFO1("posicionou o pivot - %d", fim);
                        swap(array, idx_pivot, fim);
                        idx_pivot = fim;
                        break;
                }else{
                        if(array[inicio] > array[fim]){
                //                INFO0("trocou");
                                swap(array, inicio, fim);
                        }
                }
        }
```

```
        return idx_pivot;
}

void quicksort(int array[], int inicio, int fim) {
  //INFO0("->quicksort()");
        if(inicio < fim){
                int idx_pivot = divide(array, inicio, fim);
                quicksort(array, inicio, idx_pivot - 1);
                quicksort(array, idx_pivot + 1, fim);

        }
}

void swap(int array[], int i, int j){
        int aux  = array[i];
        array[i] = array[j];
        array[j] = aux;
}
```

7 REFERÊNCIAS BIBLIOGRÁFICAS

BERNSTEIN, P. Middleware: A Model for Distributed System Services. Communications of the ACM, 1996.

DANTAS, M. Computação Distribuída de Alto Desempenho: Redes, Clusters e Grids Computacionais. Axcel Books, 2005.

FERREIRA, D. J.; SILVA, A. P. C.; DANTAS, M. A. R.; QIN, J.; BAUER, M. A. Toward Resource Management in Multi-Cluster Grid Configurations through an Ontology-Fuzzy Approach. 2009

FOSTER, I.; KESSELMAN, C. Computational Grids. The Grid: Blueprint for a New Computing Infrastructure, 1999.

MARZOLLA, M. et al. Open Standards-Based Interoperability of Job Submission and Management Interfaces across the Grid Middleware Platforms gLite and UNICORE. E-Science and Grid Computing, IEEE International Conference on, p. 592–601, 2007.

OSCAR, http://oscar.openclustergroup.org/, disponível em Junho de 2009.

ROMBERG, M. The UNICORE architecture: seamless access to distributed resources. High Performance Distributed Computing, 1999. Proceedings. The Eighth International Symposium on, p. 287–293, 1999.

SimGrid, http://simgrid.gforge.inria.fr/, disponível em Junho de 2009.