24° Simpósio Brasileiro de Redes de Computadores
Curitiba, PR – Brasil
De 29 de Maio a 02 de Junho de 2006

ANAIS / PROCEEDINGS WCGA
IV Workshop de Computação em Grid e Aplicações
-
IV Workshop on Grid Computing and Applications

Editores / Editors
Bruno Schulze, Laboratório Nacional de Computação Científica (LNCC)
Michael Stanton, Rede Nacional de Educação e Pesquisa (RNP)
Radha Nandkumar, National Center for Supercomputing Applications (NCSA)

Realização / Organization
Pontifícia Universidade Católica do Paraná (PUCPR)
Universidade Federal do Paraná (UFPR)
Universidade Tecnológica Federal do Paraná (UTFPR)

Promoção / Sponsor
Sociedade Brasileira de Computação – SBC
Laboratório Nacional de Redes de Computadores – LARC

Capa: Naicele O. de Noronha
Produção Editorial: Gráfica Universitária Champagnat

ISBN 85-7669-072-1

Prefácio

As edições anteriores do Workshop de Computação em Grid e Aplicações - WCGA (2003 - 2004 - 2005), foram realizadas em conjunto com o Programa de Verão do LNCC, em Petrópolis-RJ, Brasil. Estas edições anteriores do WCGA receberam uma combinação interessante de diferentes áreas de computação em grids como infra-estrutura, *middleware*, redes e aplicações, com apresentações técnicas e posters fomentando discussões em vários tópicos interessantes, incluindo grids clássicos, grids baseados em objetos e serviços, escalonamento, e aplicações, tais como bioinformática; meteorologia e governo eletrônico, entre outros. Estas edições geraram um interesse crescente na comunidade e nos baseamos nesta tradição para mais esta edição.

A quarta edição do WCGA está pela primeira vez sendo realizada em conjunto com o Simpósio Brasileiro de Redes de Computadores - SBRC, e tem o objetivo da é promover o encontro de pesquisadores no campo de *grids* computacionais, abordando problemas de larga escala e do mundo real em ambientes de *grids*, incluindo os tópicos mais interessantes e estimulantes que surgiram em anos anteriores, e outros inéditos. Houve vinte (20) trabalhos submetidos sendo onze (11) aceitos como artigos completos e cinco (6) convidadas para apresentação como poster. Os apresentadores apontaram problemas e soluções em um ou mais temas dentre aqueles identificados para o workshop.

Como palestrante convidado tivemos Jim Basney do NCSA, apresentando *MyProxy: A Multi-Purpose Grid Authentication Service,* abordando aspectos relacionados á segurança de ciberinfraestruturas e ciberambientes.

Finalmente, a coordenação do workshop gostaria de agradecer a todos os autores e participantes pela apresentação dos trabalhos e contribuições de pesquisa em *grids* computacionais e aplicações, agradecer a todos os membros dos comitês de programa e de organização pela ajuda na elaboração do programa do workshop e nas atividades, e adicionalmente agradecer os apoios de LNCC, RNP, NCSA, GIGA, FACC, MCT, SGI e SBC.

Bruno Schulze, **co-coordenador e co-editor**
ComCiDis – Grupo de Computação Científica Distribuída,
Laboratório Nacional de Computação Científica - LNCC, Brasil

Michael Stanton, **co-coordenador e co-editor**
Rede Nacional de Educação e Pesquisa – RNP, Brasil

Radha Nandkumar, **co-coordenador e co-editor**
National Center for Supercomputing Applications - NCSA,
University of Illinois Urbana-Champaign UIUC, EUA

Preface

WCGA2006 - IV Workshop on Grid Computing and Applications

The previous editions of the Workshop on Grid Computing and Applications - WCGA were held in conjunction with the Summer Programme of LNCC, in Petrópolis-RJ, Brazil. The previous editions of WCGA saw a good combination of the many different flavors of grid computing, in terms of infrastructure, middleware, networking and applications, with several technical presentations and posters generating discussions on several stimulating topics including, Classic Grids, Object and Service-based Grids, Scheduling, and Applications such as Bioinformatics, Meteorology and E-Government, among others. The editions generated increasing interest in the community and we built on this tradition for this year.

The fourth edition of WCGA is being held for the first time in conjunction with the Brazilian Symposium on Computer Networks – SBRC, and has the objective to bring together researches in the field of computational grids, addressing large-scale and real world problems in grid environments, including the most interesting and stimulating topics emerged last year, and some novel. There were twenty (20) submissions with eleven (11) papers being selected as full papers and five (6) invited for poster presentation. The presenters highlighted issues and solutions in one or more of the themes that had been identified for the workshop.

As keynote speaker we had by Jim Basney, from NCSA, presenting *MyProxy: A Multi-Purpose Grid Authentication Service*, addressing issues related to security in cyberinfrastructures and cyberenvironments.

Finally, the workshop coordination would like to thank all the authors, and participants for presenting their work and contributions on research in computational grids and applications, to thank all the members of the program and organizing committee for helping to shape the workshop program and activities, and additionally to thank the sponsors from LNCC, RNP, NCSA, GIGA, FACC, MCT, SGI and SBC.

Bruno Schulze, **co-chair and co-editor**
ComCiDis - Distributed Scientific Computing Group,
National Laboratory for Scientific Computing - LNCC, Brazil

Michael Stanton, **co-chair and co-editor**
National Research and Education Backbone – RNP, Brazil

Radha Nandkumar, **co-chair and co-editor**
National Center for Supercomputing Applications - NCSA,
University of Illinois Urbana-Champaign UIUC, USA

Promoção

Sociedade Brasileira de Computação – SBC

Diretoria

Presidente
Cláudia Maria Bauzer Medeiros (UNICAMP)

Vice-Presidente
José Carlos Maldonado (ICMC - USP)

Diretoria Administrativa e Finanças
Carla Maria Dal Sasso Freitas (UFRGS)

Diretoria de Eventos e Comissões Especiais
Karin Breitmann (PUC-Rio)

Diretoria de Educação
Edson Norberto Cáceres (UFMS)

Diretoria de Publicações
Marta Lima de Queirós Mattoso (UFRJ)

Diretoria de Planejamento e Programas Especiais
Virgílio Augusto Fernandes Almeida (UFMG)

Diretoria de Secretarias Regionais
Aline dos Santos Andrade (UFBA)

Diretoria de Divulgação e Marketing
Altigran Soares da Silva (UFAM)

Diretoria de Regulamentação da Profissão
Roberto da Silva Bigonha (UFMG)

Diretoria de Eventos Especiais
Carlos Eduardo Ferreira (USP)

Conselho

Mandato 2003-2007
Flávio Rech Wagner (UFRGS)
Siang Wu Song (USP)
Luiz Fernando Gomes Soares (PUC/RJ)
Ariadne Maria B. Carvalho (Unicamp)
Taisy Silva Weber (UFRGS)

Mandato 2005 - 2009
Ana Carolina Salgado (UFPE)
Ricardo de Oliveira Anido (UNICAMP)
Jaime Simão Sichman (USP)
Daniel Schwabe (PUC/RJ)
Marcelo Walter

Suplentes - Mandato 2005-2009
Robert Carlisle Burnett (PUCPR)
Ricardo Reis (UFRGS)
José Valdeni de Lima (UFRGS)
Raul Sidnei Wazlawick (UFSC)

Laboratório Nacional de Redes de Computadores – LARC

Diretor do Conselho Técnico Científico:
Guido Lemos de Souza Filho (UFPB)

Vice-Diretor do Conselho Técnico-Científico:
José Neuman de Souza (UFC)

Diretor executivo:
Luci Pirmez (UFRJ)

Vice-Diretor executivo:
Lisandro Zambenedetti Granville (UFRGS)

Membros Institucionais
MEC-SESU, USP, PUC-RJ, UNICAMP, UFRGS, UFRJ, UFMG, UFPE, INPE, UFPB, LNCC, IME, UFSC, CEFET-PR, UFC, UFF, UFSCAR, CEFET-CE, UFRN, UFES, UFBA, UNIFACS, UECE.

Realização

Comitê de Organização

Coordenação Geral
Carlos Alberto Maziero, PUCPR
Elias Procópio Duarte Jr., UFPR
Keiko Verônica Ono Fonseca, UTFPR

Coordenação do Comitê de Programa
Joni da Silva Fraga, UFSC

Coordenação de Tutoriais
Raimundo José de Araújo Macêdo, UFBA

Coordenação de Minicursos
Edmundo Roberto Mauro Madeira, UNICAMP

Coordenação de Palestras e Painéis
Edmundo de Souza e Silva, UFRJ

Coordenação do Salão de Ferramentas
Thaís Vasconcelos Batista, UFRN

Coordenação de Workshops
Emílio Carlos Gomes Wille, UTFPR

Coordenação de Publicação
Mauro Fonseca, PUCPR

Comitê Consultivo
Rossana Maria de Castro Andrade, UFC
Joaquim Celestino Júnior, UFC
José Augusto Suruagy Monteiro, UNIFACS
José Neuman de Souza, UFC
Maria Janilce Bosquiroli Almeida, UFRGS
Lisandro Zambenedetti Granville, UFRGS
José Marcos Silva Nogueira, UFMG
Luci Pirmez, UFRJ NCE

Organização Local
Anelise Munaretto, UTFPR
Pedro Torres (POP-PR)
Silvia Avila

Comitê de Programa do WCGA 2006

Alba Melo (UnB)
Alexandre Sztajnberg (UERJ)
Antônio Tadeu A Gomes (LNCC)
Artur Ziviani (LNCC)
Bruno Schulze (LNCC)
Carlos Becker Westphall (UFSC)
Celso Mendes (UIUC)
Claudio Geyer (UFRGS)
Edison Ishikawa (IME-RJ)
Edmundo Madeira (UNICAMP)
Elias Duarte Jr. (UFPR)
Francisco Brasileiro (UFCG)
Inês Dutra (UFRJ)
Jairo Panetta (INPE/CPTEC)
Luciano Gaspary (UNISINOS)
Luis Carlos Trevelin (UFSCar)
Marcelo Albuquerque (CBPF)
Michael Stanton (RNP e UFF)
Noemi Rodriguez (PUC-Rio)
Osvaldo Carvalho (UFMG)
Philippe Navaux (UFRGS)
Rossana M de Castro Andrade (UFC)
Simone Martins (UFF)
Tereza Cristina Carvalho (USP)
Thais Vasconcelos Batista (UFRN)
Vinod Rebello (UFF)
Walfredo Cirne (UFCG)

Comitê de Programa do WCGA 2006

Alba Melo (UnB)
Alexandre Sztajnberg (UERJ)
André Detsch (UNISINOS)
Angelo Perkusich (UFCG)
Antônio Tadeu A Gomes (LNCC)
Artur Ziviani (LNCC)
Ayla Débora Dantas de Souza (UFCG)
Bruno Schulze (LNCC)
Carlos Westphall (UFSC)
Celso Mendes (UIUC)
Claudio Geyer (UFRGS)
Edison Ishikawa (IME-RJ)
Edmundo Madeira (UNICAMP)
Eliane Araújo (UFCG)
Francisco Carvalho (UFC)
Francisco H de Carvalho Júnior (UFC)
Inês Dutra (UFRJ)
Luciana Lima (LNCC)
Luciano Gaspary (UNISINOS)
Luis Carlos Trevelin (UFSCar)
Luiz Bittencourt (UNICAMP)
Marcelo Albuquerque (CBPF)
Michael Stanton (RNP e UFF)
Milena Pessoa M Oliveira (UFCG)
Nicolas Maillard (UFRGS)
Noemi Rodriguez (PUC-Rio)
Osvaldo Carvalho (UFMG)
Rossana M de Castro Andrade (UFC)
Simone Martins (UFF)

Sumário / Summary

Sessão Técnnica 1 /
Technical Session 1

Uma proposta para execução distribuída de consultas em um ambiente de Grid para o CoDIMS

Gustavo Gaburro Trevisol, Alvaro Cesar Pereira Barbosa

[1]Programa de Pós-Graduação em Informática
Laboratório de Pesquisa em Redes e Multimídia – LPRM
Departamento de Informática
Universidade Federal do Espírito Santo – UFES
Campus de Goiabeiras
Av. Fernando Ferrari, S/N 29060-970 Vitória, ES

{gtrevisol, alvaro}@inf.ufes.br

Abstract. *Grid is a computational environment in which applications can use multiple distributed computational resources in a safe, coordinated, efficient and transparent way. Data integration systems are distributed and they can make use of Grid environments to obtain a better performance and a rational use of available resources. This work describes a proposal for the Distributed Query Execution Engine (MECD) to CoDIMS. The MECD is inserted in a Grid Environment for distributed and parallel execution of sub-queries sent to data sources. This proposal will permit a more efficient execution of queries that requires more processing power.*

Resumo. *Grid é um ambiente de computação no qual aplicações podem utilizar múltiplos recursos computacionais distribuídos de forma segura, coordenada, eficiente e transparente. Sistemas de integração de dados são, por natureza, distribuídos e podem se beneficiar de ambientes Grid para melhor desempenho e uso racional dos recursos disponíveis. Neste artigo é apresentada uma proposta para a Máquina de Execução de Consultas Distribuídas (MECD) do CoDIMS. A MECD é inserida em um ambiente de Grid para execução distribuída e paralela das sub-consultas enviadas às fontes de dados. Esta nova proposta possibilitará a execução mais eficiente de consultas que necessitam de um grande poder de processamento.*

1. Introdução

O enorme crescimento da quantidade de Sistemas de Informação espalhados pelas organizações de todo o mundo gerou um grande problema no que diz respeito à integração de dados. É cada vez mais notável a necessidade de sistemas capazes de prover acesso integrado a dados distribuídos e heterogêneos, de forma transparente e homogênea. Esse é o objetivo do CoDIMS (*Configurable Data Integration Middleware System*) [Barbosa et al. 2002], um ambiente flexível e configurável que permite gerar sistemas *lightweight* [Batory and Thomas 1995] de integração de dados heterogêneos e distribuídos, configurados para aplicações específicas. A arquitetura do CoDIMS é baseada em componentes *frameworks* [Barbosa 2001], que podem ser instanciados para implementar serviços específicos de integração de dados, denominados DIMS (*Data Integration Middleware Services*). Esses componentes são baseados nos serviços de sistemas

gerenciadores de banco de dados e possuem a característica de serem *plug and play*, fornecendo assim um alto nível de interoperabilidade, composição e abstração da sua implementação e da forma de comunicação que utilizam. Os componentes do CoDIMS são implementados como *web services*, possibilitando distribuição do ambiente, execução remota de tarefas e reuso de tais componentes [Trevisol 2004].

Para que o CoDIMS consiga realizar a integração de dados de fontes heterogêneas é definido um modelo global para o qual os diferentes modelos de dados locais das fontes a serem integradas são mapeados, utilizando ontologias [Silvestre 2005]. Quando uma consulta global é submetida ao CoDIMS, através do mapeamento global-local, são definidas as sub-consultas a serem enviadas a cada fonte da dados, através de um Plano de Execução de Consulta (PEC) otimizado. A MEC [Pinheiro 2004] gerencia a execução do PEC, repassando as sub-consultas ao componente Acesso aos Dados, que as envia às fontes de dados por meio dos *wrappers*. Os sub-resultados retornados de cada fonte de dados são convertidos para o modelo global pelos *wrappers* [Côco 2005] e disponibilizados para a MEC, que executará as operações necessárias sobre eles para compor o resultado final (Figura 1).

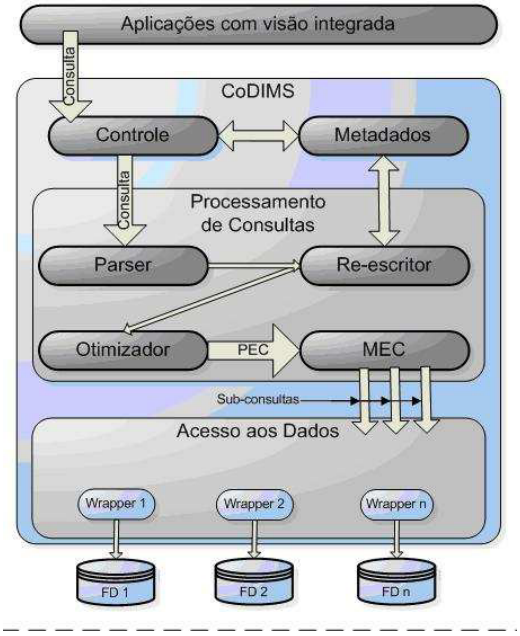

Figura 1. CoDIMS e seus componentes principais

Na implementação atual do CoDIMS os *wrappers* estão distribuídos em um ambiente de Grid [Biancardi 2005], permitindo a execução das sub-consultas em paralelo, melhorando o desempenho do sistema. Os sub-resultados gerados por cada sub-consulta são disponibilizados de forma distribuída nos próprios nós do Grid alocados aos *wrappers*. Porém a MEC continua localizada em um único servidor, de forma que as operações sobre os sub-resultados para gerar o resultado final sejam realizadas de forma centralizada e sequencial.

Em casos onde diversas fontes de dados com grande quantidade de informações devem ser integradas, é natural imaginar a utilização de um ambiente de Grid não somente

para a execução das sub-consultas, mas também para a execução das operações sobre os sub-resultados. Neste contexto, este trabalho tem por objetivo apresentar a proposta de uma Máquina de Execução de Consultas Distribuída (MECD) para o CoDIMS, inserida em um abiente de Grid. Com a MECD espera-se os seguintes resultados: primeiro, possibilitar a execução distribuída do PEC através de operadores algébricos que serão enviados dinamicamente aos nós do Grid; segundo, permitir que o resultado gerado pela execução de um operador fique disponível para ser utilizado por outros operadores na hierarquia de acordo com a árvore que descreve o PEC; terceiro, alocar/desalocar instâncias dos operadores em nós do Grid de modo a otimizar o uso dos recursos computacionais disponíveis e diminuir o tempo de processamento da consulta.

Segundo [Kossmann 2000], mesmo com a disponibilidade de infra-estrutura necessária para processamento de consultas distribuídas (ex: redes de alta velocidade), não existe uma solução completa já criada, sendo uma tarefa complexa com um grande número de problemas: *i)* sistemas distribuídos têm se tornado muito grandes envolvendo milhares de fontes de dados heterogêneas, incluindo PCs e Mainframes; *ii)* o estado dos sistemas distribuídos mudam muito rapidamente porque a utilização das fontes de dados variam durante o tempo e novas fontes são adicionadas ao sistema; *iii)* sistemas legados quase sempre não são projetados para processamento distribuído e necessitam interagir com outros "modernos" sistemas em um ambiente distribuído.

2. Trabalhos Relacionados

Na literatura são encontradas diversas propostas e sistemas voltados para o problema de processamento de consultas distribuídas. Dentre eles, e mais relacionados a este trabalho, destacam-se: CoDIMS-G [Fontes et al. 2004], ParGRES [Mattoso et al. 2005], MOCHA [Rodriguez-Martinez and Roussopoulos 2000].

O CoDIMS-G usa o ambiente de Grid para a execução em paralelo de programas sobre uma única base de dados. Ele não tem o objetivo de ser uma extensão do CoDIMS, de forma a adaptá-lo de maneira mais geral ao ambiente de Grid para efetuar integração de dados heterogêneos e distribuídos, mas sim usar Grid como solução para acessar uma grande base de dados.

O ParGRES propõe uma solução que combina técnicas de processamento paralelo de consultas em um cluster de Bancos de Dados relacionais PostgreSQL. Nele não há nenhuma integração de esquemas uma vez que utilizam um único Banco de Dados nem há otimização da consulta. Uma consulta SQL submetida é decomposta em sub-consultas e enviadas aos nós do cluster. Os sub-resultados são retornados para um componente expecífico que executa operações de junção para montar o resultado final.

No MOCHA as sub-consultas são executadas atraves de DAPs (*Data Access Providers*) que são instalados em máquinas próximas às fontes de dados ou nas próprias fontes, não suportando efetuar a realocação de uma DAP em tempo de execução caso ocorra sobrecarga de algum nó. Eles podem aplicar filtros sobre os sub-resultados porém a junção de todos os sub-resultados gerados é feita em um único nó.

De forma complementar aos anteriores, a proposta deste trabalho visa adicionar uma nova sistemática de execução de consultas ao CoDIMS de maneira a utilizar o ambiente de Grid no qual está inserido. A MEC distribuída torna-se mais flexível, possibilitando: num primeiro momento, distribuir todo o processamento do PEC, da execução

das sub-consultas à integração dos sub-resultados, ambas nos nós do Grid; futuramente, com a disponibilidade de escalonadores dinâmicos, torna-se possível determinar qual nó do Grid possui melhor capacidade para executar os *wrappers* e os operadores algébricos.

3. A MECD para o CoDIMS

A atual MEC inserida no CoDIMS [Pinheiro 2004, Biancardi et al. 2005] foi baseada no *framework* QEEF [Ayres 2003] e implementada como um componente *framework*: possui partes fixas que devem existir em todas as instâncias da MEC e partes que serão instanciadas dependendo do cenário em que o CoDIMS seja utilizado. As partes fixas criam uma infra-estrutura básica para a execução das consultas. Essas partes são: a interface da MEC, o processamento do PEC, a estrutura básica dos operadores e do conjunto resultado. As partes a serem instanciadas, responsáveis pela adaptação do CoDIMS a cada cenário, são compostas pelos operadores algébricos e pelo conjunto resultado de cada modelo de dados.

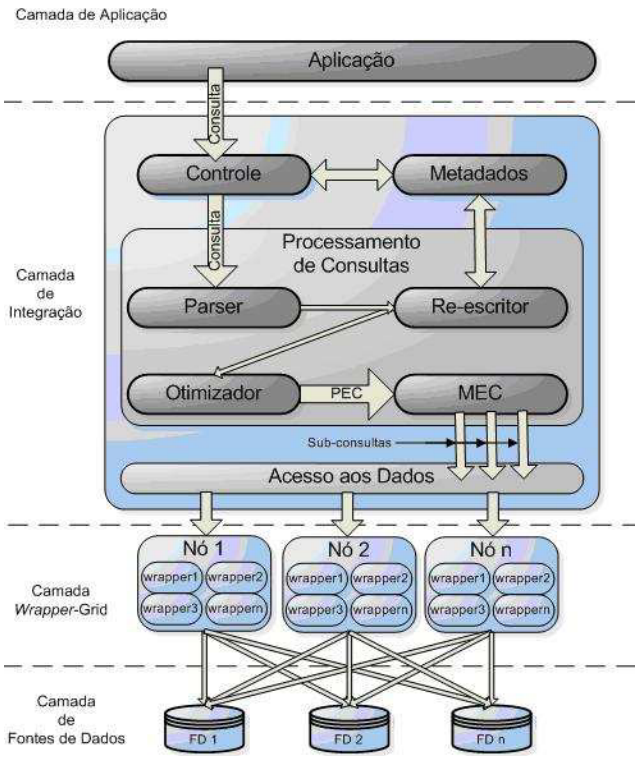

Figura 2. Camada *Wrapper*-Grid [Biancardi 2005]

Em [Biancardi 2005] foi proposta e implementada a camada *Wrapper*-Grid (Figura 2) com o objetivo de desacoplar os *wrappers* do componente Acesso aos Dados do CoDIMS. Os *wrappers* são alocados em um ambiente distribuído, mais precisamente em um Grid, podendo utilizar de uma maneira consistente, transparente e segura múltiplos recursos computacionais distribuídos [Foster et al. 2002]. Com essa abordagem reduziu-se a sobrecarga do servidor Acesso aos Dados, através da execução, em paralelo, dos *wrappers* e das sub-consultas. Além disso, os conjuntos-resultado provenientes da

execução das sub-consultas nas fontes de dados são armazenados em um ambiente distribuído, permitindo executar o PEC de forma distribuída [Özsu and Valduriez 2001] em um Grid.

A camada *Wrapper*-Grid foi implementada no CoDIMS tendo em vista a expansão do ambiente, principalmente do componente Processamento de Consultas: disponibilizar um ambiente com acesso distribuído e paralelo às diversas fontes de dados pertencentes a uma configuração. A camada *Wrapper*-Grid também dá suporte ao processamento distribuído de consultas que requerem maior poder de processamento, permitindo que operações mais complexas sejam processadas em paralelo nos nós do Grid.

3.1. Paralelizando Sub-Consultas

No CoDIMS, o PEC gerado pelo Otimizador [Gomes 2005] é implementado através de uma árvore de consulta. Cada folha desta árvore contém uma sub-consulta a ser direcionada para a fonte de dados a qual está relacionada. A capacidade do CoDIMS de integrar dados heterogêneos e distribuídos se dá, em parte, devido ao comportamento do componente Acesso aos Dados que, através dos *wrappers*, consegue acessar as fontes locais nos seus diversos formatos. Para tanto, cada sub-consulta deve ser enviada ao componente Acesso aos Dados pela MEC. Na implementação atual do CoDIMS este envio é feito de maneira síncrona, ou seja, para cada folha do PEC é estabelecida uma comunicação com o componente Acesso aos Dados e enviada a sub-consulta. Essa comunicação perdura até o término da recuperação do conjunto-resultado da sub-consulta. Isso segue para todas as folhas do PEC, de forma que nenhuma outra folha (sub-consulta) possa ser processada fora da ordem estabelecida (caminhamento nesta árvore).

A MECD viabiliza a eliminação dessa deficiência: É possível, para cada folha do PEC, ser feita uma chamada assíncrona ao componente Acesso aos Dados, sendo que nesse componente existe uma fila de sub-consultas a serem executadas. As mensagens a serem inseridas nessa fila contêm as seguintes informações: identificador da consulta, identificador da sub-consulta e a sub-consulta propriamente dita (embutida em XML e representada segundo o padrão estabelecido em [Pinheiro 2004]), além de outras informações. Assim, uma sub-consulta a ser executada é enviada pela MECD ao componente Acesso aos Dados sendo inserida no final da fila de execução de sub-consultas. Sempre que uma inserção é realizada nessa fila, uma notificação é feita ao observador da mesma, indicando que existe uma nova sub-consulta que deve ser executada. Para tanto, é preciso recuperar os dados armazenados na fila. O observador recupera os dados da sub-consulta a ser executada seguindo a ordem imposta pela fila. Para proceder essa execução, é preciso enviar esses dados para a camada de *Wrapper*-Grid, que é formada pelos Componentes *Wrapper* Grid (WGCs) [Biancardi 2005]. Como um dos objetivos é paralelizar a execução das sub-consultas e as tarefas desempenhadas pelos *wrappers*, é criada uma *thread* para cada sub-consulta na fila.

3.2. Paralelizando o Processamento dos Sub-Resultados

Um PEC é composto por uma seqüência de operadores algébricos que a máquina é capaz de executar, com o objetivo de obter o resultado de uma consulta às fontes de dados da forma mais rápida possível. O otimizador de consulta é responsável por escolher, dentre os operadores físicos da MEC, quais são os mais propícios para a criação do PEC de uma consulta específica. Geralmente, os PECs são estruturados na forma de uma árvore onde

as folhas e os nós internos representam os operadores e os arcos representam as relações produtor-consumidor existentes entre os operadores. Assim os PECs podem assumir duas topologias básicas: Linear (à direita ou à esquerda) e Ramificada (Figura 3).

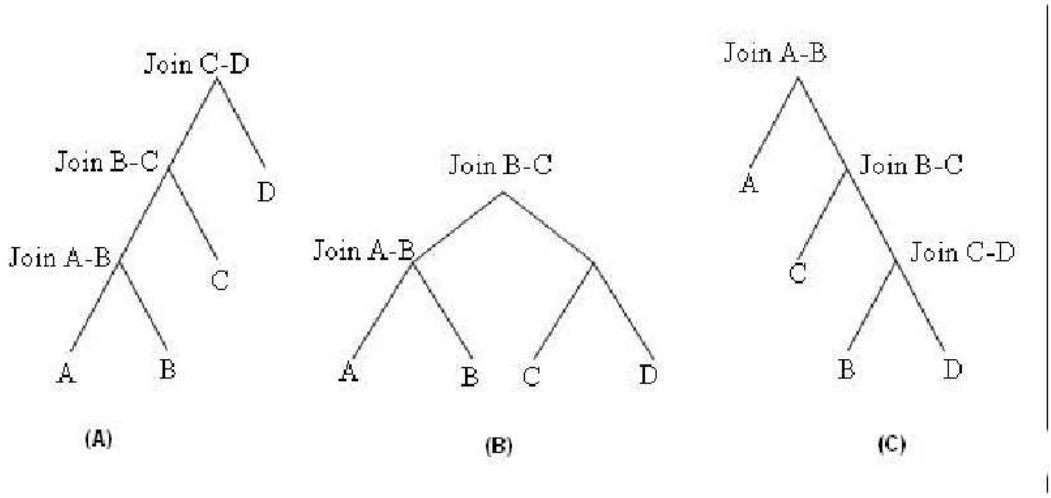

Figura 3. Topologia de PEC: linear a esquerda, ramificada, linear a direita [Ayres 2003]

A escolha entre as topologias influencia no processo de execução da consulta e no processo de otimização [Ayres 2003]. A utilização das topologias lineares tem como característica a redução da quantidade de tuplas nas relações intermediárias. Desta forma, as topologias lineares levam a execuções de consultas em tempos menores. Contudo, a utilização da topologia ramificada permite a criação de planos ótimos, porém, o tempo gasto em sua criação pode inviabilizar o seu uso. A topologia ramificada favorece o paralelismo horizontal e distribuído, enquanto que as lineares favorecem o paralelismo vertical. O paralelismo vertical ocorre quando cada operador utiliza o resultado de um outro operador, assim as tuplas fluem pelos operadores mantendo constante a carga de processamento entre os mesmos. O paralelismo horizontal é feito dividindo a árvore gerada pela consulta em sub-árvores, e executando cada sub-árvore em um site.

Imaginando que cada site seja um nó de um Grid, a MECD faz uso da camada *Wrapper*-Grid do CoDIMS para distribuir o processamento de integração dos sub-resultados, obtidos a partir das fontes de dados. A, MECD ao executar o PEC da Figura 3.B, enviará de forma assíncrona as sub-consultas que acessarão as fontes A e B. O componente Acesso aos Dados buscará no repositório (Figura 4) os *wrappers* que acessarão as fontes referenciadas e os enviará juntamente com as sub-consultas para os nós alocados para essas tarefas. Ao final da execução das sub-consultas o componente Acesso aos Dados notificará à MECD da finalização das sub-consultas e informará em quais nós do Grid os sub-resultados estão localizados. A MECD buscará no repositório (Figura 4) o operador responsável pela operação JOIN A-B e o enviará para um nó selecionado do Grid, juntamente das localizações dos sub-resultados. Em paralelo a esse procedimento também serão executadas as sub-consultas C e D e a operação JOIN C-D em nós que poderão não ser os mesmos alocados para as operações anteriores. Após a finalização de JOIN A-

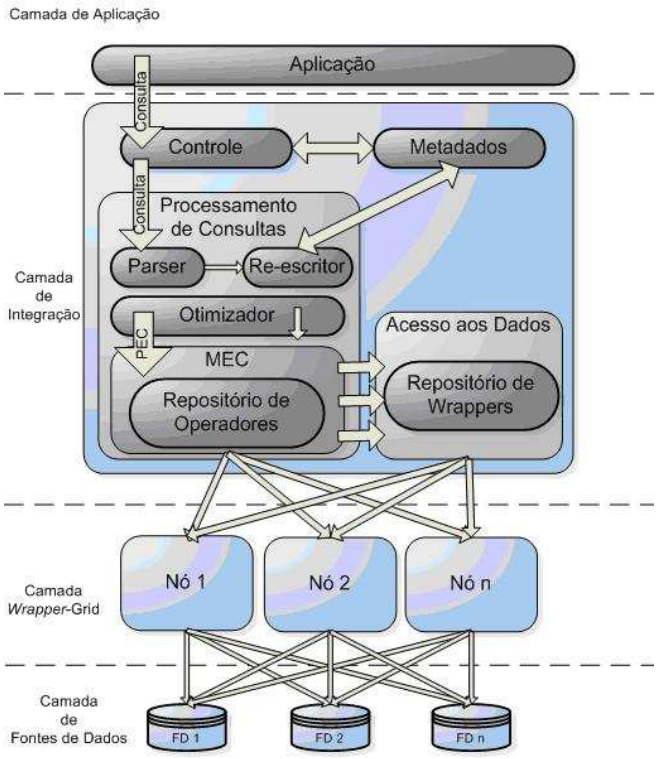

Figura 4. Repositório de Operadores e *Wrappers* e MECD acessando o Grid

B e JOIN C-D os respectivos operadores enviarão notificações à MECD informando da finalização das operações e também a localização dos sub-resultados. Ela por sua vez enviará novamente o operador JOIN para um nó do Grid, juntamente da localização dos sub-resultados gerados por JOIN A-B e JOIN C-D. Finalmente, esse operador executará a operação JOIN B-C e o endereço do resultado final será enviado para a MECD.

4. Implementação

O projeto CoDIMS está sendo desenvolvido no Laboratório de Pesquisa em Redes e Multimídia (LPRM) da Universidade Federal do Espírito Santo (UFES) utilizando o sistema operacional Linux e a linguagem de programação Java. O ambiente de Grid instalado é o *Globus Toolkit* 3.2 [Globus 2004].

A partir do trabalho de [Biancardi 2005] foi incorporado ao CoDIMS um otimizador de consultas [Gomes 2005] que tem por finalidade receber do Re-escritor (Figura 1) um PEC inicial e, a partir de estatísticas previamente coletadas das fontes de dados, gerar um PEC otimizado que é enviado para a sua execução na MEC.

Também foi desenvolvido, como objetivo do trabalho de [Laquine 2006], uma sistemática para envio dinâmico de código Java para ser executado nos nós do Grid. Isso possibilitou a evolução na camada *Wrapper*-Grid, já que na sua implementação em [Biancardi 2005], todos os *wrappers* necessários para acesso às fontes de dados devem necessariamente estar localizados em todos os nós do Grid (Figura 2), uma vez que não existe o envio dinâmico de um *wrapper* para ser executado em um único nó no momento

do envio da sub-consulta. Com essa abordagem, caso a implementação de algum *wrapper* sejá modificada, será necessário que todos os nós do Grid sejam atualizados com sua nova versão. Como solução, foi incorporado ao componente Acesso aos Dados um repositório de *wrappers* (Figura 4), cujo objetivo é armazenar todos os *wrappers* disponíveis para uma certa configuração do sistema. Ao receber uma sub-consulta, o componente Acesso aos Dados busca no repositório qual é o *wrapper* responsável por acessar a fonte de dados referenciada, e o envia juntamente com a sub-consulta ao nó determinado para realizar a execução. Com isso não é mais necessário que cada nó do Grid contenha todos os *wrappers* de uma configuração, aumentando assim a manutenabilidade do sistema já que uma possível atualização de um *wrapper* acarretará apenas a sua atualização no reposítorio de *wrappers*.

A sistemática de envio de código Java dinamicamente para os nós do Grid também foi aplicada à MECD. Ela, por sua vez, possui um repositório de operadores algébricos (Figura 4), que armazena todos os operadores à ela necessários. Quando a MECD for notificada da geração de sub-resultados que devem ser integrados (de acordo com a ávore que descreve o PEC) ela buscará qual o operador responsável pela operação e o enviará para o nó alocado para a execução.

Atualmente a MECD está em estágio de implementação final necessitando concluir a tradução do resultado, no formato interno do CoDIMS (XML), para o formato a ser retornado à aplicação que enviou a consulta (relacional).

4.1. Avaliação dos resultados

Após o término da implementação e incorporação da MECD ao CoDIMS serão realizados testes para avaliação, consolidação e homologação da solução aqui apresentada. Os testes serão realizados no próprio LPRM no ambiente de Grid já existente e utilizado em [Biancardi 2005].

A sistemática para a avaliação dos resultados seguirá os seguintes procedimentos: a) criar bases de dados de modelos distintos (XML, textual, relacional etc) que deverão ser integradas; b) dividir os testes em várias etapas, cada uma delas envolvendo diferentes quantidades de dados; c) executar cada etapa de testes várias vezes em dias/horários distintos; d) realizar os procedimentos anteriores em dois cenários: centralizado e distribuído, descritos a seguir.

O objetivo é avaliar e comparar o comportamento de cada cenário no que se refere a tempo médio de execução das consultas, para diferentes tamanhos das bases de dados envolvidas.

4.1.1. Cenário centralizado

Será utilizada uma instância do CoDIMS com a atual MEC centralizada [Pinheiro 2004]. Nesta instância os sub-resultados gerados pelos *wrappers* deverão ser transferidos para o servidor no qual a MEC está localizada. As operações sobre os sub-resultados serão realizadas de forma sequencial.

4.1.2. Cenário distribuído

Neste cenário será configurada uma instância do CoDIMS que utiliza a MECD no ambiente de Grid do LPRM. Os sub-resultados gerados pelos *wrappers* já estarão localizados nos nós do Grid [Biancardi 2005]. Neste caso a MECD enviará os operadores para serem executados nos nós do Grid de acordo com o PEC recebido, posibilitando assim a execução paralela das operações descritas pelo PEC.

5. Conclusões

Sistemas de integração de dados visam a liberar o usuário de ter que localizar as fontes de dados, interagir com cada uma delas isoladamente e combinar manualmente os dados de múltiplas fontes, heterogêneas e distribuídas [Hakimpour and Geppert 2001, Halevy 2003]. Cada vez mais, integração de dados se faz necessária em diversos domínios de aplicação, devido a grande quantidade de dados, essencialmente heterogêneos e distribuídos, que vem sendo gerados. Como exemplos podemos citar: uma indústria, pode ter o seu processo de produção monitorado por dados que precisam ser integrados para análise, provenientes de sensores instalados em diversas máquinas relacionadas a esse processo; pesquisas médicas e científicas podem fazer uso de dados gerados por diferentes instituições de pesquisas, estabelecendo uma colaboração global e incrementado a possibilidade de se alçancar um melhor resultado. Assim, integração de dados têm sido reconhecido como um problema ubíquo e criticamente importante, e vem demandando pesquisa na área de banco de dados por mais de uma década [Abiteboul et al. 2003, Sheth and Larson 1990], tendo várias questões em aberto [Ziegler and Dittrich 2004]. Atualmente, é importante explorar oportunidades para combinar banco de dados e tecnologias relacionadas que podem incrementar o uso da informação [Abiteboul et al. 2003].

Nesse contexto, vem sendo desenvolvido o CoDIMS (*Configurable Data Integration Middleware System*). O CoDIMS é um *middleware* para integração de dados heterogêneos e distribuídos, onde as consultas submetidas são decompostas em sub-consultas para cada fonte de dados integrante do sistema. As sub-consultas são enviadas pela MEC para o Componente de Acesso aos Dados que se utiliza de *wrappers* para acessar as diversas fontes integradas pelo sistema.

Após a incorporação da camada de *Wrapper*-Grid ao CoDIMS tornou-se viável a implementação de uma Máquina de Execução de Consultas Distribuída (MECD) que possa utilizar a mesma infra-estrutura construída para a camada *Wrapper*-Grid de modo que não somente a execução das sub-consultas seja distribuída, mas também que o processamento dos sub-resultados seja realizado em um ambiente de Grid.

Essa proposta contribui com o projeto CoDIMS por possibilitar que os seguintes resultados sejam alcançados com a incorporação da MECD: permitir a execução distribuída do PEC, através de operadores algébricos que serão enviados dinamicamente aos nós do Grid [Laquine 2006]; disponibilizar no Grid o resultado gerado pela execução de um operador para ser utilizado por outros operadores na hierarquia de acordo com a árvore que descreve o PEC; possibilitar a alocação/desalocação de instâncias dos operadores em nós do Grid de modo a otimizar o uso dos recursos computacionais disponíveis.

Espera-se, principalmente, que os resultados a serem alcançados com este trabalho sejam proporcionais aos obtidos em [Biancardi 2005], onde foi possível diminuir o tempo

médio de execução das consultas submetidas ao CoDIMS.

Terminada esta etapa, os trabalhos futuros a serem desenvolvidos são: migrar da versão 3.2 para a versão 4.0 do *Globus Toolkit*; Implementar um escalonador dinâmico para a re-alocação dos *wrappers* e dos operadores algébricos a serem enviados aos nós do Grid para sua execução.

Referências

Abiteboul, S., Agrawal, R., Bernstein, P., and et al., M. C. (2003). The lowell database research self assessment. Disponível em: http://research.microsoft.com/˜Gray/lowell.

Ayres, F. V. M. (2003). *QEEF: uma máquina de execução de consultas extensível*. PhD thesis, PUC-Rio, Brasil.

Barbosa, A. C. P. (2001). *Middleware para Integração de Dados Heterogêneos Baseado em Composição de Frameworks*. PhD thesis, PUC-Rio, Brasil. Disponível em: http://codims.lprm.inf.ufes.br/arquivos/publicacoes/tese_alvaro.pdf.

Barbosa, A. C. P., Porto, F. A. M., and Melo, R. N. (2002). Configurable data integration middleware‘system. *Journal of the Brazilian Computing Society*, pages 12–19.

Batory, D. S. and Thomas, J. (1995). P2: A lightweight dbms generator. Technical report, University of Texas.

Biancardi, C. (2005). Distribuição e execução de wrappers em ambiente de grid para o codims. Master's thesis, UFES. Disponivel em: http://codims.lprm.inf.ufes.br/publicacoes.html.

Biancardi, C., Silvestre, L. J., and Barbosa, A. C. P. (2005). Uma proposta para distribuição e execução de *wrappers* em um ambiente de grid para o codims. *III Workshop on Computational Grids and Applications*. Petrópolis - RJ.

Côco, T. (2005). Implementando wrappers xml e relacional para o codims. Monografia de Graduação - UFES. Disponível em: http://codims.lprm.inf.ufes.br/arquivos/publicacoes.html.

Fontes, V., Dutra, M., Porto, F. A. M., Schulze, B., and Barbosa, A. C. P. (2004). Codims-g: a data and program integration service for the grid. pages 29–34.

Foster, I., Kesselman, C., Nick, J., and Tuecke, S. (2002). The physiology of the grid: An open grid services architecture for distributed system integration. *Open Grid Service Infrastructure WG, Global Grid Forum.*

Globus (2004). The globus toolkit. Technical report. Disponível em: http://www.globus.org.

Gomes, F. R. (2005). Um otimizador de consulta para o codims. Monografia de Graduação - UFES. Disponivel em: http://codims.lprm.inf.ufes.br/publicacoes.html.

Hakimpour, F. and Geppert, A. (2001). Resolving semantic heterogeneity in schema integration: an ontology based approach. In *Proceedings of the international conference on Formal Ontology in Information Systems*, pages 297–308, Maine - USA.

Halevy, A. Y. (2003). Data integration: A status report. In *Proceedings of 10th Conference on Database Systems for Business Technology and the Web (BTW 2003)*, Heppenhiem - Germany.

Kossmann, D. (2000). The state of the art in distributed query processing. *ACM Computing Surveys*, pages 422–469.

Laquine, C. E. (2006). Envio de código dos *Wrappers* em ambiente de grid para o codims. Monografia de Graduação - UFES. Disponivel em: http://codims.lprm.inf.ufes.br/publicacoes.html.

Mattoso, M., Zimbrão, G., Lima, A. A. B., Baião, F., Braganholo, V. P., Aveleda, A. A., Miranda, B., Almentero, B. K., and Costa, M. N. (2005). Middleware para processamento paralelo de consultas olap em clusters de banco de dados. In *Simpósio Brasileiro de Banco de Dados*. ACM Press. Uberlândia, MG.

Pinheiro, F. S. (2004). Incorporando uma máquina de execução de consultas ao codims. Monografia de Graduação - UFES. Disponivel em: http://codims.lprm.inf.ufes.br/publicacoes.html.

Rodriguez-Martinez, M. and Roussopoulos, N. (2000). Mocha: A self-extensible database middleware system for distributed data sources. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 213–224. ACM Press. Texas, USA.

Sheth, A. P. and Larson, J. A. (1990). Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surv.*, pages 183–236. Disponível em: http://doi.acm.org/10.1145/96602.96604.

Silvestre, L. J. (2005). Uma abordagem baseada em ontologias para a gerência de metadados do codims. Master's thesis, UFES. Disponivel em: http://codims.lprm.inf.ufes.br/publicacoes.html.

Trevisol, G. G. (2004). Codims: Incorporando nova abordagem na comunicação entre seus componentes. Monografia de Graduação - UFES. Disponível em: http://codims.lprm.inf.ufes.br/arquivos/publicacoes/Monografia_Gustavo.pdf.

Ziegler, P. and Dittrich, K. R. (2004). Three decades of data integration - all problems solved? In *18th IFIP World Computer Congress (WCC 2004)*, IFIP International Federation for Information Processing, pages 3–12, Toulouse, France. Kluwer.

Özsu, M. T. and Valduriez, P. (2001). *Princípios de Sistemas de Banco de Dados Distribuídos*. Editora Campus, Rio de Janeiro - RJ, 2nd edition.

Desafios para Provisão de Integridade de Processamento em Grades Computacionais *

Felipe Martins[†,1,2], **Márcio Maia**[‡,1,3], **Rossana M. de Castro Andrade**[1,4], **Aldri L. dos Santos**[1,4] **e José Neuman de Souza**[1,2,4]

[1]Universidade Federal do Ceará (UFC), Campus do Pici, CEP 60.755-640, Fortaleza, Ceará, Brasil

[2]UFC, Programa de Pós-Graduação em Engenharia de Teleinformática

[3]UFC, Centro de Tecnologia, Departamento de Engenharia Elétrica

[4]UFC, Centro de Ciências, Departamento de Computação

{felipe,marcio}@cenapadne.br, {rossana,aldri,neuman}@lia.ufc.br

Abstract. *Highly distributed and dynamic environments such as computational grids are constantly under attacks, for example, malicious attacks that corrupt job results. This paper focuses on the main challenges for guaranteeing the integrity of processing in grids, presenting a classification of misbehaving nodes and well-known fault tolerance techniques that can be applied to detect and minimize the presence of malicious nodes. It also compares the most important grid simulators that could be extended to evaluate the behavior of such hostile environment. Furthermore, a case study shows the efficiency of the investigated techniques.*

Resumo. *Ambientes amplamente dispersos e dinâmicos como as grades computacionais estão sujeitos a diversas formas de ataques, entre eles, os ataques maliciosos que corrompem os resultados dos jobs. Este artigo aborda os principais desafios para a provisão de integridade de processamento em grades, apresentando uma classificação de nós com mau comportamento e técnicas de tolerância a falhas que podem ser aplicadas para detectar e minimizar estes ataques. Além disso, são discutidos e comparados os principais simuladores de grades que podem ser estendidos para avaliar o comportamento de um ambiente de grade hostil. Por fim, é apresentado um estudo de caso que analisa a eficiência das técnicas investigadas.*

1. Introdução

As grades computacionais são caracterizadas pela agregação, seleção e compartilhamento de recursos computacionais distribuídos entre os vários domínios de redes. Independente da localização geográfica, essa coleção de recursos pode ser a mais heterogênea

*Esta pesquisa foi realizada com o apoio do Centro Nacional de Processamento de Alto Desempenho no Nordeste (CENAPAD-NE).

†Mestrando financiado pela Fundação Cearense de Apoio ao Desenvolvimento Científico e Tecnológico (FUNCAP) durante dois anos e pela Flextronics/Sony Ericsson, convênio Nº. 1542 - UFC/FCPC, durante quatro meses.

‡Graduando financiado pela Samsung, convênio Nº. 1617 UFC/FCPC, desde Dezembro/2005.

possível, contando com máquinas com diferentes arquiteturas de *hardware* e sistemas operacionais. Esse "supercomputador virtual" permite que Organizações Virtuais (*Virtual Organizations* ou VOs) [Foster et al. 2001] trabalhem de forma colaborativa, proporcionando um significativo aumento de desempenho tanto em termos de processamento como de serviços oferecidos. No entanto, o paradigma das grades computacionais proporciona novos desafios para a comunidade científica. As características de dispersão e heterogeneidade, por exemplo, conferem às grades necessidades mais complexas se comparadas às redes tradicionais. No que diz respeito à segurança, uma grade deve fornecer mecanismos eficientes para a autenticação dos usuários. Do mesmo modo, também é necessário que certas informações sejam trafegadas com sigilo e somente entre os usuários autorizados.

Logo, os requisitos comuns de autenticação, controle de acesso, integridade, privacidade e não-repúdio precisam ser estendidos para atender a este ambiente de rede. Além disso, as grades podem ser formadas por diferentes redes, com diferentes domínios administrativos e, por conseguinte, diferentes políticas de segurança. Garantir a segurança da informação em ambientes deste tipo é uma tarefa complexa, envolvendo novos requisitos, tais como assinatura única, uniformidade e proteção das credenciais, interoperabilidade com soluções locais, estabelecimento dinâmico de domínios confiáveis, entre outros [Welch et al. 2003].

Além destes requisitos, a dinamicidade deve ser levada em consideração, pois o grande número de *jobs* e participantes que interagem o tempo todo com o ambiente impõem novos desafios. Um deles é a proteção das aplicações em execução, pois a possibilidade de manipulação indevida (maliciosa) dos resultados obtidos com o processamento compromete a utilidade e eficiência das grades. De nada adianta um ambiente de alto desempenho "contaminado" por máquinas maliciosas interessadas em invalidar ou corromper os resultados dos processos.

Nesse contexto, as grades devem ainda oferecer mecanismos que garantam a integridade dos dados não só durante a transmissão, mas também durante o processamento. Caso contrário, o resultado da aplicação como um todo fica comprometido, além de incidir num alto custo em termos de desempenho. Porém, a detecção de falhas desta natureza não é simples e, mesmo que fosse, haveria a necessidade de recomputar todos os dados de entrada, incorrendo mais uma vez em custo no desempenho. Para evitar que usuários de grades obtenham resultados indevidos, é necessário que os mecanismos de detecção de manipulação maliciosa também consigam isolar as máquinas que agem dessa forma.

O presente trabalho discute os principais desafios para a provisão de integridade dos resultados dos *jobs* distribuídos e processados pelas máquinas que compõem o ambiente de grade. Para isso, este artigo é organizado da seguinte forma: a seção 2 descreve as formas de ataques de segurança aos quais as grades estão expostas, com ênfase nos ataques à integridade; a seção 3 aborda as falhas de segurança causadas por mau comportamento dos usuários e as técnicas comumente usadas para verificação de integridade do processamento; a seção 4 discute o uso de diagnóstico em nível de sistema como solução para detecção de falhas maliciosas em grades; a seção 5 apresenta uma investigação sobre simuladores de grades, os quais podem ser utilizados para criar cenários e testar as técnicas investigadas, discutindo suas características, limitações e requisitos; a seção 6 apresenta um estudo de caso que aplica algumas das técnicas discutidas em um desses simuladores de grades; por fim, a seção 7 traz as considerações finais.

2. Formas de Ataques às Grades

Comparados às redes tradicionais, ambientes altamente dispersos e dinâmicos como os de grades são mais suscetíveis a ataques de segurança. Nesse caso, um sistema robusto de segurança deve evitar ou resolver problemas como mascaramento de usuários e servidores, mau uso ou uso excessivo dos recursos, utilização não autorizada de serviços, subversão dos recursos disponíveis, entre outros.

As grades estão, portanto, expostas às ameaças já bem conhecidas das redes tradicionais, embora com novas variações. Podemos citar, por exemplo, os ataques que comprometem a *disponibilidade* do serviço, como aconteceu recentemente com o serviço de computação sob demanda oferecido pela Sun Microsystems que foi alvo de um ataque do tipo DoS (*Denial-of-Service*), exatamente em seu primeiro dia de operação [Gohring 2006]. A solução utilizada foi aplicar um controle de acesso, permitindo que apenas usuários já cadastrados anteriormente pudessem acessar o serviço oferecido. Esta solução é fácil e relativamente eficaz contra ataques externos. No entanto, ela não impede que os usuários cadastrados possam, de um momento para outro, utilizar os recursos da grade para disparar um ataque DoS ou DDoS (*Distributed DoS*) contra a própria grade ou contra um outro *website* qualquer. Uma defesa para este problema seria limitar o uso de recursos por parte de usuários.

Além disso, as informações trafegadas na grade podem ser alvos de ataques à *confidencialidade*. Para algumas aplicações é primordial que estes dados sejam mantidos sob sigilo, de forma que nem todos possam conhecer o que está sendo processado em sua máquina, por questões de patente, segredo industrial, entre outros. Ações como "grampo" ou mascaramento de usuário comprometem a confidencialidade. Um atacante também pode simplesmente vasculhar todos arquivos temporários criados em sua própria máquina ou, se for o caso, explorar uma base de dados que esteja compartilhada no ambiente.

Ataques à *integridade* também são uma realidade na computação em grade. O ambiente deve oferecer uma infra-estrutura que garanta não só a integridade dos dados durante a transmissão, mas que também ofereça segurança aos fornecedores de recursos, a fim de encorajar uma maior participação e disponibilização dos mesmos. Os fornecedores de recursos, na maioria das vezes, desconhecem que códigos estão sendo executados em suas máquinas, e a grade precisa assegurar-se que estas aplicações em execução não irão, por exemplo, destruir o sistema de arquivos dos computadores ou mesmo congestionar a rede. Vírus, *worms* e cavalos-de-tróia constituem uma séria ameaça, pois quanto maior a grade, maior a extensão do estrago.

Mecanismos de proteção dos recursos têm sido desenvolvidos para garantir que uma dada aplicação não contenha código malicioso que venha a prejudicar o pleno funcionamento das máquinas. Uma solução, por exemplo, é criar uma área com sistema de arquivos, processos e recursos de rede isolados, sem poder acessar outros processos. Um determinado processo confinado nessa área só poderá acessar os outros processos, o sistema de arquivos e os recursos de rede que se encontram em sua própria área. A *virtualização*, como é conhecida esta solução, é adotada em técnicas como o Sandbox [Keahey et al. 2004], que além de limitar as possíveis ações do código de um processo, evita o consumo exagerado dos recursos disponíveis. Esta técnica é ilustrada na Figura 1, mostrando como o código oriundo de máquinas remotas não pode acessar diretamente os recursos do sistema local.

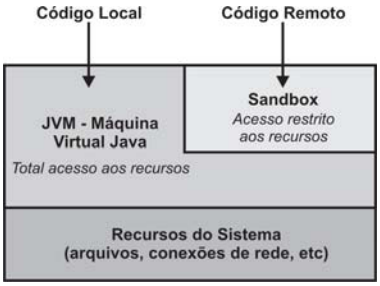

Figura 1. Exemplo da técnica de *virtualização*

Além de proteger os recursos, é preciso proteger também as aplicações. Os usuários precisam ter garantias de que seus códigos estão sendo executados em máquinas idôneas, que não se recusam a processar os *jobs*, nem manipulam os resultados obtidos. Logo, a infra-estrutura de grade precisa certificar-se de que, dentre as máquinas provedoras de recursos que compõem o ambiente (nós), não existem usuários maliciosos interessados em invalidar ou corromper os resultados dos *jobs*.

Em virtude do grande número de processos e usuários que podem interagir em uma grade, oferecer mecanismos de segurança que evite a manipulação dos resultados dos *jobs* por elementos maliciosos é uma tarefa fundamental. Assim, o esquema de segurança utilizado deve ser robusto de tal forma que garanta a integridade em dois níveis: durante a transmissão dos dados e durante o processamento dos *jobs*. E neste último caso, deve-se garantir a proteção tanto das máquinas fornecedoras de recursos, quanto das aplicações em execução.

3. Tratamento de Falhas Maliciosas

Esta seção apresenta uma classificação dos tipos de falhas causadas por mau comportamento, dentre as quais, destacam-se as falhas maliciosas. Em seguida, são abordadas as técnicas comumente utilizadas para proteger as aplicações em grade contra essas falhas de origem maliciosa bem como uma forma de diminuir o *overhead* de processamento introduzido por essas técnicas.

3.1. Classificação de Mau Comportamento

Por tratarem-se de ambientes distribuídos e heterogêneos, as grades computacionais estão sujeitas a erros que ocasionam falhas de diversas naturezas. Quanto maior a grade, maior a possibilidade de falhas, já que um número maior de componentes e elementos precisam estar continuamente interagindo. Entre os principais tipos de falhas encontrados em ambientes deste tipo, destacam-se as falhas bizantinas ou arbitrárias, as quais englobam todas as possíveis causas de falhas, inclusive as decorrentes de mau comportamento.

As falhas originadas pelo mau comportamento envolvem máquinas (nós) que passam a agir de maneira inativa, egoísta ou maliciosa [Hollick 2004]. Os nós inativos não cooperam com a rede, deixando de encaminhar pacotes, recusando-se a processar os *jobs* que lhe são entregues ou omitindo informações sobre seus recursos disponíveis. Os nós egoístas negligenciam ajuda aos demais nós, favorecendo apenas seus próprios interesses. Por exemplo, no contexto OurGrid [Andrade et al. 2004], um nó egoísta (chamado *free-rider*) é aquele que apenas consome recursos da comunidade, sem oferecer seus próprios

recursos quando solicitado. Mesmo não doando recursos, um nó egoísta pode manter-se respondendo às requisições de descoberta de recursos a fim de permanecer ativo no ambiente. Já a classe de nós maliciosos abriga aqueles nós interessados, por exemplo, em subverter os recursos da grade, oferecer um resultado inválido ou difundir *worms* e vírus entre as máquinas.

3.2. Técnicas para Verificação de Integridade de Processamento

Em geral, mecanismos de tolerância a falhas são empregados para garantir a confiabilidade (*reliability*) no funcionamento de um sistema. O objetivo é manter o sistema funcional, mesmo diante de problemas acidentais, como falta de energia elétrica, *bugs* de *software*, degradação do *hardware* e má configuração dos componentes. Em contrapartida, mecanismos de segurança (*security*) são utilizados para prevenir ou corrigir uma ação deliberadamente maliciosa, como por exemplo, distribuição de vírus, invasão de redes e quebra de chaves secretas. Contudo, mecanismos de confiabilidade podem ser aplicados no domínio da segurança e vice-versa. Mecanismos clássicos, como controle de acesso e autenticação, típicos de segurança, podem, por exemplo, ser utilizados para prevenção de falhas. Da mesma forma, ferramentas de tolerância a intrusões podem aplicar conceitos comuns à tolerância a falhas [Correia 2005].

Nesse sentido, técnicas comuns à área de confiabilidade podem ser utilizadas para tolerar falhas de segurança de natureza maliciosa. Duas dessas técnicas [Sarmenta 2001], *Voto Majoritário* e *Verificação Focalizada*, são discutidas a seguir.

Voto Majoritário

A técnica de Voto Majoritário é fundamentalmente baseada em replicação e conta com a presença de um nó gerente, responsável pela verificação de integridade dos nós fornecedores de recursos. O gerente replica e distribui o mesmo *job* para os nós fornecedores. Cada nó fornecedor processa o seu respectivo *job* e devolve o resultado para o gerente.

De posse dos resultados, o nó gerente verifica se a maioria dos nós concorda sobre o resultado, checando se a maioria dos resultados devolvidos coincide. Naturalmente, há um limite mínimo para que os resultados coincidentes sejam considerados válidos. Quanto maior for esse limite, maior a garantia que os resultados estão corretos. A Figura 2 ilustra o esquema de voto majoritário, onde dos seis nós fornecedores de recursos, apenas o nó D ofereceu um resultado diferente da maioria. Assim, o nó gerente assume "x" como resultado final válido e o nó D como malicioso.

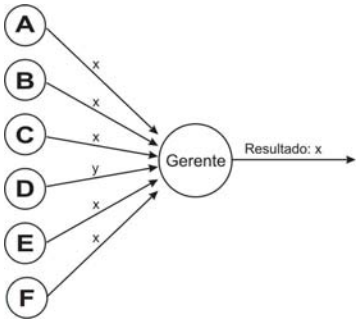

Figura 2. Seis nós fornecedores de recursos: nó D enviando valor corrompido.

Embora seja simples de implementar, o voto majoritário é indicado somente em casos onde a taxa de erros é necessariamente baixa. Se o ambiente possui uma taxa de erro muito alta, é preciso aumentar o limite mínimo para assegurar que as respostas sejam válidas, e isto aumenta o número de réplicas a serem distribuídas. A quantidade de réplicas reflete no desempenho do sistema, pois se vários nós estão ocupados com os mesmos *jobs*, consequentemente recursos são desperdiçados.

Verificação Focalizada

Uma alternativa ao esquema de voto majoritário é um esquema baseado em verificação focalizada, onde os nós fornecedores de recursos são testados aleatoriamente por um nó gerente. O teste consiste em um *job* enviado ao nó fornecedor, cujo resultado é previamente conhecido pelo nó gerente. Se o resultado divergir do esperado, o nó gerente assume que o nó fornecedor agiu de forma maliciosa. Todos os resultados anteriores oferecidos pelo nó malicioso são descartados e o nó gerente reenvia esses *jobs* para outros nós.

A utilização de um mecanismo do tipo lista negra (*blacklist*), associada à verificação focalizada, permite indicar quais nós não são mais desejados na grade. Os nós que fornecerem resultados incorretos entram para a lista negra e não recebem mais *jobs* para processar, aumentando ainda mais a confiabilidade do sistema.

3.3. Reputação

Uma forma de diminuir o *overhead* de processamento introduzido pelas técnicas discutidas é utilizando um esquema de reputação. Nós de boa reputação podem ser considerados melhores fornecedores de recursos e, portanto, não precisam ser testados com tanta freqüência. A reputação de um nó traduz o seu grau de confiabilidade.

Verificação de reputação tem sido utilizada tanto em ambientes centralizados, por exemplo, sistemas de leilão via *web*, como em ambientes distribuídos. Nesse último caso, os sistemas *Peer-to-Peer* (P2P) para compartilhamento de arquivos são exemplos mais comuns. Em ambientes deste tipo, o uso de reputação minimiza a presença de *peers* maliciosos interessados em difundir arquivos falsos ou incompletos, ou mesmo vírus e *worms*. Em grades computacionais essa realidade assemelha-se com os nós maliciosos que corrompem os resultados dos *jobs* processados. Desta forma, o uso de reputação permite avaliar o comportamento dos nós no ambiente.

4. Diagnóstico em Nível de Sistema como Estratégia contra Manipulação

Uma outra forma de detectar falhas em sistemas distribuídos é através de algoritmos de diagnóstico em nível de sistema. Tais algoritmos definem uma série de testes aplicados aos elementos do sistema, cujo conjunto de resultados obtidos, denominado *síndrome*, permite identificar quais nós estão falhos e quais estão em pleno funcionamento.

Uma estratégia de testes simplificada consiste no envio de estímulos periódicos dos nós testadores para os nós testados. Se as respostas dos nós testados chegam dentro de um intervalo de tempo, o nó testador assume esses nós como sem-falha. Caso contrário, o nó testador assume que os nós testados estão falhos. Dentre os principais modelos de diagnóstico em nível de sistema, destacam-se o PMC [Preparata et al. 1968], o Hi-ADSD [Duarte and Nanya 1998] e os baseados em comparações, como por exemplo, o modelo MM [Maeng and Malek 1981], o qual é ilustrado na Figura 3.

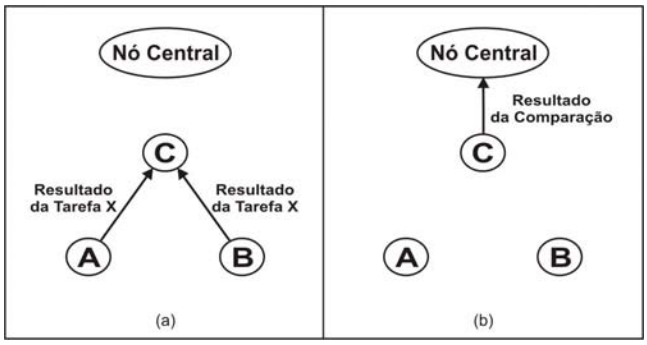

Figura 3. Funcionamento do Modelo MM.

No exemplo da Figura 3 acima, um dado nó testador C envia para os nós A e B uma mesma tarefa e coleta os resultados produzidos (a). A seguir, o nó C repassa o resultado da comparação ao nó central que realiza o diagnóstico (b). Embora diagnóstico não seja uma novidade, ele pode ser empregado em ambientes amplamente distribuídos e dinâmicos. No entanto, ao considerar estas características, é preciso realizar novas asserções quanto ao comportamento dos nós. Além disso, para grades é necessário, obviamente, aplicar testes mais sofisticados do que, por exemplo, um simples *ping*, ainda mais se tratando de uma rede composta por nós que podem corromper os resultados dos *jobs* por eles processados. Uma possibilidade é combinar diagnóstico com teste do tipo verificação focalizada, e também utilizar uma lista negra e um esquema eficiente de reputação.

A Figura 4 ilustra essa situação, onde o nó central envia um *job* de teste para o nó A (a) e recebe o resultado do processamento, a fim de compará-lo com o resultado esperado (b). Quanto maior a incidência de resultados corretos oferecidos pelo nó A, maior o seu grau de reputação. Caso em algum momento a reputação do nó A ultrapasse um limite mínimo estabelecido, ele poderá ser inserido em uma *blacklist*.

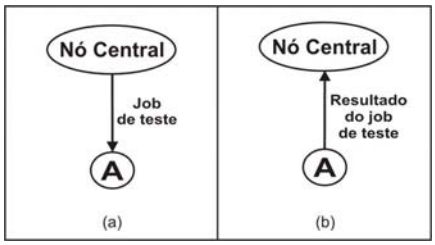

Figura 4. Verificação focalizada no contexto de diagnóstico.

A utilização de diagnóstico como estratégia contra ataques maliciosos mostra-se uma solução eficiente, uma vez que considera a natureza heterogênea das grades. Diagnóstico independe das plataformas de *hardware* e *software* utilizadas. Outro ponto positivo é que diagnóstico pode ser aplicado tanto em grades fechadas, que são VOs formadas entre corporações, como em grades abertas, constituídas por nós comuns conectados à Internet e dispostos a doarem seus recursos ociosos num contexto de computação voluntária.

Contudo, o uso de diagnóstico suscita algumas observações. Ao utilizar uma abordagem baseada em comparações, é necessário considerar o formato dos testes que são realizados, visto que dois nós distintos sem falhas (i.e., não maliciosos) podem oferecer resultados diferentes para uma mesma tarefa em virtude de sua arquitetura. Uma tarefa que envolva ponto flutuante, por exemplo, pode resultar em diferentes respostas, dependendo da arquitetura de *hardware* dos nós.

Uma outra questão refere-se ao intervalo de tempo que o nó possui para responder a um teste. Em diagnóstico, o tempo de uma rodada de testes é normalmente limitado. Uma rodada corresponde ao intervalo de tempo necessário para se obter informações de diagnóstico sobre todos os nós. Como as grades podem ser compostas por nós de poder computacional diversificado, o tempo de resposta a um teste também pode variar. Esse problema agrava-se ainda mais se os nós estão completamente espalhados e a grade atinge proporções intercontinentais, pois fatores como latência e largura de banda influem diretamente no tempo de transmissão dos resultados obtidos.

5. Simulação em Grades

Obter resultados analíticos para comparar as heurísticas de diferentes técnicas, especialmente em ambientes heterogêneos e dispersos, não é trivial. Uma forma de análise comumente empregada é através de simulação. Simuladores são ferramentas capazes de representar aspectos de uma situação ou de um processo de forma realística. Contudo, é preciso saber se a ferramenta se adequa a realidade estudada, antes de aplicar uma solução em um ambiente de simulação.

Há uma diversidade de simuladores de grades disponíveis na Internet. Dentre eles, destacam-se: OptorSim [Bell et al. 2003], GridNet [Lamehamedi et al. 2002], MicroGrid [Song et al. 2000], SimGrid [Legrand et al. 2003] e GridSim [Buyya and Murshed 2002]. Esta seção apresenta um breve estudo destes simuladores, avaliando suas limitações e características como forma de utilização, documentação, portabilidade, extensibilidade e mecanismos de simulação (Veja Tabela 1).

O OptorSim é um simulador de grades implementado em Java que trabalha com réplicas de *jobs*. A ferramenta simula o comportamento da grade, de acordo com os parâmetros definidos pelo usuário: topologia, recursos disponíveis, conjunto de *jobs* e estratégia de replicação. Desta forma, o OptorSim é voltado basicamente para a comparação e análise de estratégias de escalonamento baseadas em replicação.

O GridNet é um módulo do NS (*Network Simulator*) que permite especificar as características dos nós provedores de recursos da grade e as trocas de mensagens entre os mesmos. Escrito em C++, o GridNet utiliza bibliotecas NS para simular decisões de replicação em cada nó, adicionando novos componentes específicos para o ambiente de grade, como por exemplo, a capacidade de armazenamento e processamento de um determinado nó.

O MicroGrid tende a ser classificado na verdade como um emulador de grades, visto que emula uma grade Globus para se conhecer como as aplicações se comportam. Dessa forma, esse simulador permite que se execute aplicações em uma grade virtual para estudos de *testbeds* sob condições controladas. O MicroGrid é implementado em C e utiliza arquivos XML como entrada, onde se pode especificar o *job*, bem como o valor máximo de ciclos de CPU e memória utilizada pelo nó virtual.

O SimGrid também foi escrito em C e utiliza arquivos XML como entrada, definindo a topologia de rede e as características e responsabilidades dos nós. Capaz de realizar simulações orientadas a eventos, o SimGrid fornece um conjunto de bibliotecas que permite ao usuário criar, destruir e verificar as tarefas e os recursos envolvidos na simulação. As decisões de escalonamento podem ser pré-definidas ou tomadas durante a sua execução. Entretanto, devido às freqüentes verificações condicionais e desvios dentro de seu código, o SimGrid é considerado uma plataforma complexa.

Com vasta documentação, o GridSim permite a modelagem e simulação de entidades envolvidas em uma grade, através da criação e monitoria de seus diferentes recursos. Por utilizar o pacote SimJava para simulação baseada em eventos, o GridSim tem a facilidade da máquina virtual JVM (*Java Virtual Machine*), disponível para sistemas mono e multiprocessados, além de trabalhar com *multithread*, tornando o sistema bastante escalável e portável.

Em detrimento da segurança, os simuladores de grades preocupam-se especialmente com os aspectos de escalonamento. Em sua grande maioria, eles implementam métodos que permitem imitar a distribuição de processos segundo estratégias, como FIFO (*First In First Out*) e *Round-Robin*, além de permitir que os usuários construam seus próprios algoritmos de escalonamento.

Como os aspectos de segurança não são tratados nos simuladores de grades, cabe ao usuário incorporar novos métodos que traduzam as características e comportamentos desejados para a plataforma de simulação de sua preferência. No entanto, é preciso certificar-se que o simulador atenda aos requisitos e que suas limitações não impactem no trabalho a ser desenvolvido. A Tabela 1 traz um resumo comparativo das plataformas de simulação investigadas, de acordo com as características mencionadas.

Tabela 1. Comparação entre os simuladores de grades avaliados.

Característica	OptorSim	GridNet	MicroGrid	SimGrid	GridSim
Utilização	Simulador	Simulador	Emulador	Simulador	Simulador
Linguagem	Java	C++	C	C	Java
Documentação	Boa	Fraca	Boa	Alta	Alta
Portabilidade	Sim	Não	Não	Não	Sim
Extensibilidade	Boa	Alta	Fraca	Regular	Boa
Mecanismo de simulação	Multithread	Serial	Paralelo	Serial	Multithread

Dentre os simuladores estudados, o GridSim mostra-se como a plataforma mais adequada do ponto de vista da flexibilidade. Embora o GridNet apresente maiores facilidades para ser estendido, o GridSim possui uma ampla documentação da API disponível na Internet [Buyya and Murshed 2002]. O GridSim também permite que os nós que compõem o ambiente possam desempenhar diferentes papéis em um dado momento - diferente de simuladores como o SimGrid, por exemplo, que assume os papéis de todos os nós de forma estática. Por ser escrito em Java, o GridSim oferece ainda portabilidade e a vantagem de trabalhar com múltiplas *threads*, permitindo que diversas entidades usuários submetam *jobs* para execução simultânea.

6. Estudo de Caso

Embora o GridSim permita simulações de cenários realísticos, algumas modificações foram introduzidas no simulador para representar o uso de verificação focalizada no contexto de diagnóstico. Logo, para simular a situação descrita na Figura 4 foram inseridos novos métodos que permitem que o nó central execute os *jobs* de teste antes de enviá-los aos demais nós. Desta forma, o nó central pode conhecer previamente os resultados destes *jobs* para só então compará-los com os resultados recebidos dos nós testados.

As simulações foram realizadas em um *cluster* de quatro nós SunFire Opteron biprocessados com 2Gb de RAM cada. Os cenários de testes assumem um ambiente de grade composto de 10.000 *jobs* (incluindo os de teste) distribuídos entre 200 nós. Cada *job* envolve a fatoração de uma *string* gerada aleatoriamente, onde o código ASCII de cada caractere da *string* é multiplicado por um elemento de um conjunto finito de números primos e o resultado final é a soma das multiplicações de todos os fatores. Por exemplo: seja a *string* "abcde" e o conjunto de primos {3,5,7,11}. O resultado final do cálculo é 97 x 3 + 98 x 5 + 99 x 7 + 100 x 11 + 101 x 3 = 2877.

Variando o percentual de nós maliciosos na grade, a freqüência e a quantidade de *jobs* de teste, foram avaliadas a quantidade de rodadas de testes necessárias para a detecção dos nós de mau comportamento, bem como o impacto do uso da *blacklist*. A fim de obter resultados consistentes, para cada um dos experimentos foram realizadas 100 simulações, onde os valores médios são apresentados nos gráficos a seguir.

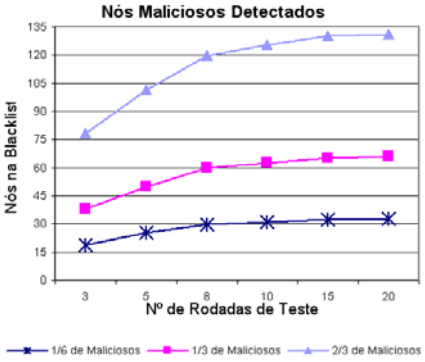

Figura 5. Quantidade de nós inseridos na blacklist.

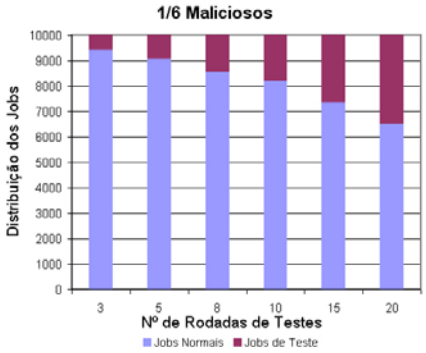

Figura 6. Custo com 1/6 dos nós maliciosos.

A Figura 5 mostra a variação da quantidade de nós maliciosos detectados em função do número de rodadas de testes. Inicialmente foi considerado 1/6 da grade comprometida, ou seja, dos 200 nós, 33 deles podem manipular os resultados. Em seguida, foi avaliado o comportamento da grade com 1/3 e 2/3 de seus nós comprometidos. É importante notar que nem todos os *jobs* são necessariamente corrompidos por estes nós maliciosos. Assim, tais nós recebem um *job* e decidem aleatoriamente se vão ou não processá-lo corretamente. Nas simulações, os nós que podem agir maliciosamente foram modelados com 25% de chances de retornarem um resultado inválido.

Para cada cenário com diferentes quantidades de nós maliciosos, foram executadas simulações com 3, 5, 8, 10, 15 e 20 rodadas de testes respectivamente. Foi observado que praticamente todos os nós maliciosos são detectados com 15 rodadas de testes. No

entanto, o *overhead* inserido no sistema é muito alto com essa quantidade de rodadas. A Figura 6 ilustra essa situação com 1/6 dos nós agindo maliciosamente. Por questão de espaço, os outros gráficos com 1/3 e 2/3 de nós maliciosos são omitidos.

Como observado na Figura 6, dos 10.000 *jobs* distribuídos, mais de 2.500 são destinados apenas para testes. Acredita-se que 8 rodadas ofereça um compromisso (*trade-off*) aceitável entre custo e segurança, pois, como mostrado na Figura 5, dos 33 nós maliciosos (1/6), 30 foram detectados. Vale ressaltar que as simulações não consideraram os aspectos de reputação, o que pode aumentar ainda mais o índice de nós maliciosos detectados sem aumentar o *overhead* de processamento.

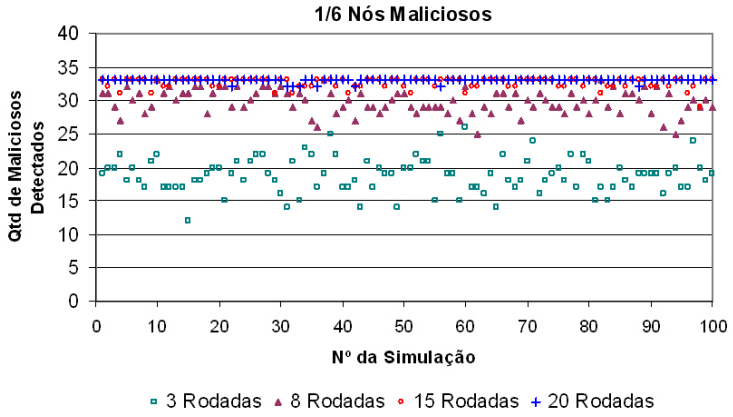

Figura 7. Nós maliciosos detectados em cada uma das 100 simulações.

A Figura 7 ilustra o número de nós maliciosos detectados em cada uma das 100 simulações realizadas em um cenário com 1/6 dos nós agindo maliciosamente (33 nós maliciosos). Para melhor visualização, foram considerados neste gráfico apenas os experimentos realizados com 3, 8, 15 e 20 rodadas de teste. Com 3 rodadas, o esquema mostra-se relativamente instável, visto que no melhor caso são detectados 26 nós e no pior somente 12. Porém, à medida que mais rodadas de teste são empregadas, a variância da quantidade de nós maliciosos detectados tende a diminuir.

7. Considerações Finais

A proteção das aplicações em execução em uma grade constitui um desafio para qualquer infra-estrutura de segurança. A presença de nós maliciosos corrompendo os resultados dos *jobs* pode ser detectada e minimizada com o uso de técnicas de tolerância a falhas. A combinação dessas técnicas com um esquema de reputação e *blacklist* pode aumentar ainda mais a segurança no ambiente e minimizar o desperdício de recursos. Uma solução promissora é o uso destes conceitos em um modelo de diagnóstico em nível de sistema. Um estudo de caso utilizando o simulador GridSim foi apresentado para investigar a eficiência da técnica de verificação focalizada, associada ao uso da *blacklist* na detecção de nós maliciosos. Os resultados preliminares obtidos demonstraram que, mesmo variando a quantidade de nós maliciosos, quase todos os nós com mau comportamento foram detectados e isolados do ambiente.

Como trabalho futuro, será realizada uma avaliação mais extensiva das técnicas discutidas neste artigo, e sua inclusão num modelo de diagnóstico que considere outras métricas e cenários de simulação.

Referências

Andrade, N., Mowbray, M., Cirne, W., and Brasileiro, F. (2004). When can an autonomous reputation scheme discourage free-riding in a peer-to-peer system? In *4th Workshop on Global and Peer-to-Peer Computing (GP2PC)*.

Bell, W., Cameron, D., Capozza, L., Millar, P., Stockinger, K., and Zini, F. (2003). Optorsim - a grid simulator for studying dynamic data replication strategies. In *International Journal of High Performance Computing Applications*, volume 17, pages 403–416.

Buyya, R. and Murshed, M. (2002). Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. In *Journal of Concurrency and Computation: Practice and Experience (CCPE)*.

Correia, M. (2005). Serviços distribuídos tolerantes a intrusões: resultados recentes e problemas abertos. In *V Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais - Livro Texto dos Minicursos*, pages 113–162. Sociedade Brasileira de Computação.

Duarte, E. P. and Nanya, T. (1998). A hierarchical adaptive distributed system-level diagnosis algorithm. *IEEE Transactions on Computers*, 47(1):34–45.

Foster, I., Kesselman, C., and Tuecke, S. (2001). The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3).

Gohring, N. (2006). Sun grid weathers dos attack. *Computerworld Home Page*, http://www.computerworld.com/securitytopics/security/story/0,10801,109809,00.html.

Hollick, M. (2004). On the effect of node misbehavior in ad hoc networks. In *Proceedings of IEEE International Conference on Communications, ICC'04*, volume 6, pages 3759–3763.

Keahey, K., Doering, K., and Foster, I. T. (2004). From sandbox to playground: Dynamic virtual environments in the grid. In *Proceedings of 5th International Workshop in Grid Computing*, pages 34–42.

Lamehamedi, H., Szymanski, B., shentu, Z., and Deelman, E. (2002). Data replication strategies in grid environments. In *Proceedings of the Fifth International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'02)*.

Legrand, A., Marchal, L., and Casanova, H. (2003). Scheduling distributed applications: the simgrid simulation framework. In *Proceedings of the 3rd International Symposium on Cluster Computing and the Grid - CCGRID2003*, pages 138–145.

Maeng, J. and Malek, M. (1981). A comparison connection assignment for self-diagnosis of multiprocessor systems. In *Digest 11th International Symposium Fault Tolerant Computing*, pages 173–175.

Preparata, F., Metze, G., and Chien, R. (1968). On the connection assignment problem of diagnosable systems. *IEEE Transactions on Electronic Computers*, 16:848–854.

Sarmenta, L. F. G. (2001). Sabotage-tolerance mechanisms for volunteer computing systems. In *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, page 337, Washington, DC, USA. IEEE Computer Society.

Song, H. J., Liu, X., Jakobsen, D., Bhagwan, R., Zhang, X., Taura, K., and Chien, A. A. (2000). The microgrid: a scientific tool for modeling computational grids. In *Proceedings of IEEE Supercomputing (SC 2000)*.

Welch, V., Siebenlist, F., Foster, I., Bresnahan, J., Czajkowski, K., Gawor, J., Kesselman, C., Meder, S., Pearlman, L., and Tuecke, S. (2003). Security for grid services. In *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*, page 48, Washington, DC, USA. IEEE Computer Society.

Escalonamento Tolerante a Falhas na Recuperação de Aplicações em MPI

Idalmis Milián Sardiña, Cristina Boeres, Lúcia Drummond

[1]Instituto de Computação – Universidade Federal Fluminense (IC-UFF)
Rua Passo da Pátria, 156 - Bloco E - 3° andar, São Domingos - 24.210-240 - Niterói - RJ - Brasil

{isardina, boeres, lucia}@ic.uff.br

Abstract. *A crucial problem in distributed systems is the probability to the occurrence of failures in resources. Recent studies search different forms to improve application execution time, also including fault-tolerant mechanisms, but in many cases testing its politicies on simulated environments. This work presents an MPI tool to execute parallel applications on a real architecture, recovering the application execution using fault-tolerant scheduling techniques. For such, it considers the information generated by a static scheduling heuristic, offering mechanisms for automatic failure detection and efficient application recovery.*

Resumo. *Um problema marcante em sistemas distribuídos é a propensão à ocorrência de falhas em recursos. Estudos realizados buscam diferentes formas de melhorar o tempo de execução das aplicações já incluindo mecanismos de tolerância a falhas, em muitos casos testando suas políticas sobre ambientes simulados. Este trabalho apresenta a implementação de uma ferramenta em MPI para executar aplicações paralelas sobre uma arquitetura real, recuperando a aplicação mediante técnicas de escalonamento tolerante a falhas. Para tal, considera as informações produzidas pela heurística de escalonamento estático, oferecendo mecanismos de detecção automática de falhas e recuperação eficiente da aplicação.*

1. Introdução

O crescimento cada vez maior do uso de redes de computadores em forma de grades computacionais para executar aplicações de alto desempenho tem levado a um estudo aprofundado das características destes sistemas distribuídos e de aplicações de alto desempenho executadas nestes ambientes com objetivo de aproveitar ao máximo os seus recursos disponíveis. Estudos recentes mostram que os ganhos obtidos em desempenho são elevados [Kee et al. 2004] e atualmente, pode-se notar como diversas aplicações das ciências e da engenharia, aquelas com alto consumo computacional e que manipulam enorme volume de dados [Pearlman et al. 2004], são executadas de forma razoavelmente eficiente em ambientes distribuídos, conhecidos como grades computacionais.

A arquitetura das grades pode ser vista como um conjunto de *clusters* de computadores conectados por redes metropolitanas, por exemplo. Esses ambientes tipicamente constituídos por recursos heterogêneos, também se caracterizam por apresentarem conexões mais rápidas intra-*clusters* e mais lentas entre máquinas de diferentes *clusters*. Isto faz com que tais ambientes possam ser modelados por meio de topologias hierárquicas,

onde a maneira de distribuir as tarefas da aplicação e a forma de realizar a comunicação nos diferentes níveis da hierarquia representam importantes questões que devem ser abordadas.

Devido a crescente necessidade de alcançar alto desempenho, diferentes tipos de heurísticas de escalonamento de aplicações em ambientes distribuídos com recursos heterogêneos [Boeres et al. 2003, Boeres et al. 2004b, Hagras and necek. 2004, Boeres et al. 2005] tem sido propostas com o objetivo de alcançar uma distribuição eficiente das tarefas da aplicação. Além da heterogeneidade, um outro aspecto marcante em ambiente de grades é a propensão à ocorrência de falhas em recursos. Este problema também tem sido extensivamente estudado considerando ambientes distribuídos [Anderson and Jonsson 2000, Liberato et al. 2000, Qin et al. 2002, da Silva Martins Jr. and Gonçalves 2005, Al-Omari et al. 2005] e deve ser atacado para a execução efetiva de aplicações.

A primeira parte deste trabalho objetiva realizar um estudo que relaciona as duas problemáticas acima citadas: utilização de uma heurística de escalonamento estático em conjunto com uma estratégia de tratamento de falhas para conseguir além de um bom desempenho, uma recuperação consistente da aplicação em caso de falha.

A estratégia de escalonamento estático tolerante a falhas apresentada neste trabalho foi proposta em [Qin et al. 2002] e é do tipo *list scheduling* [Kwok and Ahmad 1999]. Para atacar o problema de falhas permanentes é considerada a técnica de replicação passiva [Anderson and Jonsson 2000, Ghosh et al. 1997, Liberato et al. 2000, Naedele 1999, Qin et al. 2002, Al-Omari et al. 2005], baseada na especificação de cópias primárias e *backups* das tarefas da aplicação. Como os dois problemas são abordados em conjunto, a heurística tem como objetivo principal minimizar uma função multi-objetivo formada por: tempo de fim das tarefas e custo de confiabilidade.

Este trabalho tem como foco principal investir no estudo de políticas de escalonamento com mecanismos de tolerância a falhas voltadas para aplicações paralelas implementadas em MPI (*Message Passing Interface*) [Snir et al. 1995]. Para tal, foi implementado um sistema que considera as informações de escalonamento estático produzidas pela heurística, e oferece mecanismos de detecção automática de falhas, e recuperação eficiente da aplicação de acordo com dados produzidos pela mesma heurística.

Inicialmente, o sistema em MPI proposto tem um escopo menor voltado para um ambiente de *cluster* de computadores. A idéia é, na fase inicial deste estudo, viabilizar a estratégia tolerante a falhas para um ambiente real menor e depois, estender a mesma a uma grade computacional, acoplando-a a uma topologia hierárquica.

O artigo está organizado como segue. A seção 2 apresenta brevemente alguns trabalhos relacionados na literatura. A seção 3 apresenta o modelo heterogeneo da arquitetura e da aplicação. A seção 4 descreve o algoritmo de escalonamento tolerante a falhas utilizado pela ferramenta proposta.

2. Trabalhos Relacionados

Com o objetivo de atingir alto desempenho através de estratégias de escalonamento conscientes do problema de falhas em ambientes distribuídos, trabalhos como [Anderson and Jonsson 2000, Al-Omari et al. 2005, Dima et al. 2001,

Ghosh et al. 1997, Liberato et al. 2000, Naedele 1999, Qin et al. 2002] integram heurísticas de escalonamento com mecanismos que abordam tolerância a falhas. Muitas destas propostas usam mecanismos de replicação passiva.

Em [Naedele 1999] investiga-se o desempenho de diferentes heurísticas de escalonamento usando o esquema primária/*backup*. Três heurísticas de seleção foram comparadas: busca seqüencial, baseada em carga e candidata aleatória e foram incorporadas algumas modificações nas rotinas que verificam a alocação das cópias primárias e *backups*. Foi concluído que embora muitos trabalhos deste tipo não concedam importância a heurística de escalonamento, esta escolha terá um papel fundamental no desempenho obtido. O ideal seria ter um escalonador adaptativo que monitorasse a variação dos parâmetros ajustando em cada caso a estratégia de escalonamento. No entanto, as aplicações são constituídas por tarefas independentes que são escalonadas dinamicamente em processadores homogêneos.

Já em [Ghosh et al. 1997] qualquer algoritmo dinâmico para escalonar as tarefas pode ser usado tendo como objetivo principal estudar as técnicas sobre replicação passiva usadas para tolerar falhas. Neste artigo são utilizadas duas técnicas (*desallocation* e *overloading*), com o objetivo de alcançar um melhor desempenho e que um maior número de tarefas sejam escalonadas. Embora estas técnicas ofereçam um uso eficiente dos recursos em ambientes distribuídos, foi concluída que existe a necessidade de usar rápidos e simples algoritmos de escalonamento para as tarefas *backups*. O método proposto pode tolerar múltiplas falhas e não considera restrições de precedência para as tarefas da aplicação.

Em [Liberato et al. 2000] é apresentado um esquema de escalonamento tolerante a falhas onde as tarefas são consideradas com restrições de precedência e múltiplas falhas transientes podem ser toleradas, mas o sistema não é heterogêneo. O interessante deste trabalho foi conseguir a recuperação de múltiplas falhas de tarefas aperiódicas e o desenvolvimento de uma solução ótima usando a política de escalonamento tolerante a falhas *Earliest-Deadline-First*(*EDF*).

Em [Anderson and Jonsson 2000] descreve-se dois métodos que são variações do esquema de primária/*backup* para suportar tolerância a falhas de tarefas que chegam dinamicamente. No primeiro método uma função é usada para deteção de falhas e todas as falhas são detectadas no final da execução, enquanto no segundo método múltiplas cópias são usadas para a detecção de falhas e falhas podem ser detectadas em qualquer tempo durante a execução. A idéia é decidir dinamicamente se a replicação deve ser temporal ou espacial dado o tempo de execução, o *deadline* e a carga no sistema. Os métodos oferecem uma flexibilidade que melhora o escalonamento das tarefas que chegam e são consideradas falhas transientes e intermitentes.

Em [Dima et al. 2001] é proposto um algoritmo de escalonamento estático tolerante a falhas utilizando também replicação passiva, o mesmo considera dois tipos de falha: *fail silent* e *omission*. O objetivo do artigo é obter um escalonamento que seja possível e não necessariamente o melhor. Este trabalho contribui ao mostrar princípios que governam a execução de escalonamentos tolerantes a falhas sobre arquiteturas distribuídas e ao fazer uma formalização do problema para falhas tanto de processadores como de canais.

O artigo [Qin et al. 2002] mostra um algoritmo onde tarefas de uma aplicação

com restrições de precedência são executadas em um ambiente heterogêneo. A abordagem considera também heterogeneidade de recursos de processamento e comunicação como também características de confiabilidade do sistema. Para tolerar uma falha permanente de processador, este algoritmo utiliza um esquema de cópia primária e *backup* de cada tarefa da aplicação. Para melhorar a qualidade do escalonamento, visto que é tolerante somente a "uma falha", a heurística considera a possibilidade de sobreposição de execução de cópias primárias e *backups* de diferentes tarefas no mesmo processador. A abordagem faz uso de um tempo de detecção estimado e o desempenho do algoritmo é avaliado somente a partir de simulações.

3. Modelo heterogêneo do sistema

Com o objetivo de utilizar ambientes heterogêneos, o trabalho proposto se aplica a um modelo de sistema que descreve características da aplicação e da arquitetura nas Seções 3.1 e 3.2, respectivamente. A modelagem parte da necessidade de representar estes ambientes de trabalho que devem suportar a execução de grandes e variadas aplicações paralelas.

3.1. Modelagem das aplicações paralelas

Este trabalho considera aplicações paralelas que podem ser modeladas por *grafos acíclicos direcionados* (GAD), onde os nós do grafo representam as tarefas da aplicação e os arcos, a precedência entre estas, que devem ser satisfeitas para a execução correta da aplicação. Um GAD $G = (V, E, e, c)$ modela uma aplicação paralela sendo que V é o conjunto de vértices que representam as tarefas, E a relação de precedência, $e(v_i)$ o peso de execução associado com cada tarefa v_i de V e $c(v_i, v_j)$ o peso de comunicação associado com o arco (v_i, v_j) de E.

As relações de precedência das tarefas definem o GAD, considerando um sistema de precedência $S = (v_1, v_2, ..., v_n, <<)$ definido como um conjunto de tarefas com relação de precedência $<<$, onde $v_i << v_j$ com v_i,$v_j \subset V$ significa que a execução de v_j não pode ser iniciada enquanto não seja completada a execução de v_i e os dados de v_i para v_j sejam recebidos por este. Como resultado destas relações de precedência, obtemos o conjunto de predecessores e sucessores imediatos da tarefa v_i que podem ser escritos por $pred(v_i)$ e $succ(v_i)$, respectivamente. Podemos então definir $v_i \rightarrow v_j$ como a relação de precedência imediata entre duas tarefas v_i e v_j, sendo que v_i é o predecessor imediato de v_j e v_j o sucessor imediato de v_i.

3.2. Modelo da arquitetura com recursos heterogêneos

Ao descrever nossa arquitetura, temos um conjunto de processadores geograficamente dispersos com diferentes tipos de comunicações, características da arquitetura, enlaces de comunicação, distância entre os recursos dispersos e outros aspectos importantes que devem ser considerados nos estudos de desempenho, para assim obter um resultado mais próximo a realidade.

No modelo deste trabalho, $P = p_0, p_1, ..., p_{m-1}$ é considerado um conjunto de m processadores heterogêneos e $h(p_i)$ o fator de heterogeneidade do processador p_i em P, tal que o tempo de execução da tarefa v sobre o processador p_i é dado por $eh(v, p_i) = e(v) \times h(p_i)$. Também se considera que para duas tarefas adjacentes (u e v), alocadas a processadores distintos p_u e p_v, respectivamente, o custo associado com a comunicação

de $c(u, v)$ é definido como $ch(u, v) = L \times c(u, v)$, onde L é a latência L sobre cada enlace. Um processador se comunica com outros processadores através de troca de mensagens.

No modelo é considerado uma falha permanente de processador onde cada tarefa tem duas cópias idênticas denominadas cópia primária (v^P) e cópia *backup* (v^B) seguindo o esquema de cópia primária/*backup*. Ambas são executadas seqüencialmente sobre processadores diferentes. O custo de confiabilidade de uma tarefa v_i sobre um processador p_j é definido como o produto da taxa de probabilidade de falha de p_j e o tempo de execução de v_i sobre p_j.

Dada uma tarefa v, temos então os seguintes parâmetros associados $d(v)$, $s(v)$ e $f(v)$ que denotam *deadline*, tempo de inicio e tempo de fim respectivamente e $p(v)$ é o processador onde v está alocada. Estes parâmetros devem cumprir as seguintes restrições: $f(v) = s(v) + eh(v)$ e $f(v) <= d(v)$ onde $p(v) = p_i$. Uma tarefa tem um escalonamento possível se para todas a tarefas v, $f(v^p) <= d(v)$ e $f(v^b) <= d(v)$.

4. Algoritmo de Escalonamento Estático Tolerante a Falhas

O algoritmo de escalonamento que serviu como base deste trabalho eFRCD (efficient Fault-tolerant Reliability Cost Driven algorithm) [Qin et al. 2002] utiliza uma política do tipo *list scheduling* para escalonar as tarefas, incluindo mecanismos que permitem tolerar uma única falha permanente de processador. Para isto, emprega uma técnica de replicação passiva chamada primária-*backup* [Al-Omari et al. 2005, Anderson and Jonsson 2000, Ghosh et al. 1997, Naedele 1999, Liberato et al. 2000, Dima et al. 2001].

Um dos objetivos deste algoritmo é minimizar o tempo total de execução (*makespan*) de tal maneira que mais tarefas possam ser executadas antes de atingir seu *deadline*. Outro objetivo importante é obter um sistema mais confiável reduzindo o custo de confiabilidade durante o escalonamento e permitir que falhas permanentes de processador possam ser toleradas.

De um modo geral, o algoritmo consiste das seguintes 3 etapas:

1. Ordenar tarefas usando critérios de prioridade: $OrdenaTarefas()$;
   O *deadline* de cada tarefa define-se como o tempo máximo possível em que a tarefa pode terminar a sua execução e é cálculado no inicio do algoritmo. As tarefas são ordenadas por seus *deadlines*, de maneira crescente, sendo que tarefas com menores *deadlines* apresentam altas prioridades, critério que será denotado por $prior$. O *deadline* é cálculado no ínicio do algoritmo como um *deadline* relativo da forma $d(v) = tlevel(v) + MaxTarefa/2$. O algoritmo implementado considera outros critérios de prioridade como por exemplo ordenação pelo caminho crítico, por blevel, por tlevel e por tlevel mais blevel [Kwok and Ahmad 1999], dando possibilidade ao usuário de escolher.
2. Escalonar cópias primárias: $EscalonaPrimarias()$;
   Um *list scheduling* foi utilizado para escalonamento das cópias primárias. Uma lista ordenada por $prior$ com as tarefas primárias é percorrida e, para cada tarefa da lista é efetuada uma procura pelo melhor processador de acordo com os seguintes critérios. Seja $v \in V$, a próxima tarefa a ser escalonada:
   (a) De todos os processadores disponíveis, só aqueles cuja a execução obedece o *deadline* $d(v)$ são candidatos a executar v. O conjunto de processadores candidatos é denotado por $P' \subset P$ e definido como:

$$P^{'} = \{p_i \in P \mid EFT(v, p_i) + e(v) \times h(p_i) \leq d(v)\} \quad (1)$$

(b) A partir de $P^{'}$ são escolhidos aqueles processadores de melhor confiabilidade (ou seja, tem menos probabilidade de falhar). A redução de $P^{'}$ de acordo com esse critério leva a especificação do conjunto de processadores candidatos $P^{''}$, definido como:

$$P^{''} = \left\{p_i \in P^{'} \mid RC(p_i) = MIN_{p_j \in P^{'}} \{RC(p_j)\}\right\} \quad (2)$$

(c) Finalmente, o melhor processador em $P^{''}$ a ser escolhido para execução de v de acordo com os critérios a serem oferecidos em (a) e (b) é aquele que minimiza o tempo de fim de v, ou seja,

$$EFT(v, p) = MIN_{p_i \in P^{'}} \{EFT(v, p_i)\} \quad (3)$$

EFT representa o tempo de fim de execução da tarefa v no processador p_i e $RC(p_i)$ denota o custo de confiabilidade do processador p_i.

3. Escalonar cópias *backups*: $EscalonaBackups()$.
Para obter o escalonamento das *backups* é utilizado também um *list scheduling* similar ao passo anterior junto a uma estratégia primária-*backup*. O escalonamento das cópias *backups* (ver Figura 1) é realizado logo depois das primárias e verifica-se que seja satisfeito um conjunto de critérios ($C_1 - C_7$) que podem ser agrupados em função dos objetivos desejados seja durante o escalonamento das *backups* ou para a re-distribuição das mensagens.
Uma tarefa é dita "cópia primária forte"se a falha ocorre no processador onde a tarefa foi escalonada $p(v)$ antes do tempo final da sua execução $f(v^P)$ e sem que v^P deixe de receber as mensagens de seus predecessores. Este conceito (C_1) é utilizado também como critério dentro de outros para prever como vai ser o redirecionamento das mensagens em caso de falha ajudando a determinar o conjunto de predecessores e sucessores imediatos de uma cópia *backup*.
A cópia *backup* de cada tarefa deve ser alocada em um processador diferente ao processador que contém a cópia primária (C_2). Desta maneira, a cópia *backup* pode ser executada se o processador alocado a respectiva cópia primária falhar. Só serão executadas aquelas *backups* das tarefas primárias que fazem parte do processador que apresentou a falha. C_3 e C_4 são critérios empregados para achar o conjunto de processadores disponíveis para escalonar.
A possível sobreposição de *backups* (C_3) na hora de escalonar é considerado devido ao fato que o algoritmo foi proposto para o caso de uma única falha de processador. Assim, para melhorar a qualidade do escalonamento, cópias *backups* podem ser escalonadas sobrepostas em um mesmo processador. Para terminar, o procedimento atualiza a informação referente a o envio das mensagens fazendo uma re-distribuição adequada entre *backups* e/ou primárias a partir de C_6 e C_7.

A ferramenta em MPI proposta recebe como entrada a informação gerada depois de executar o algoritmo de escalonamento estático tolerante a falhas. A Figura 2 mostra um exemplo de um escalonamento de primárias e *backups* obtido utilizando o algoritmo de escalonamento tolerante a falha para um GAD de 6 tarefas. A Figura 3 mostra como seria a re-execução em caso de ocorrer falha em P_1.

procedimento $EscalonaBackups()$
1: $f(v^B) := \infty$;
2: $rc := \infty$;
3: **para** cada tarefa $v \in L_{ord}$ **faa**
4: $VerificaPrimariaForte(v)$; /*C_1*/
5: $AchaCjtoProcs(v^B, F)$; /*C_2, C_4 e C_5*/
6: **para** cada $p_i \in F$ **faa**
7: $CriaListaSemSobreposic(v, Lnsup)$; /*$C_3$*/
8: EFT:=$CalcEFT(v^B, p_i, Lnsup)$;
9: $AtualizaMinTempoProcRC(f, p, rc, EFT, p_i, RC(p_i))$;
10: **fim para**
11: $AlocaBackup(v^B, p)$
12: $AtualizaMsgs(v^B)$ /*C_1, C_6 e C_7*/
13: **fim para**
end.

Figura 1. Procedimento para escalonar *backups*

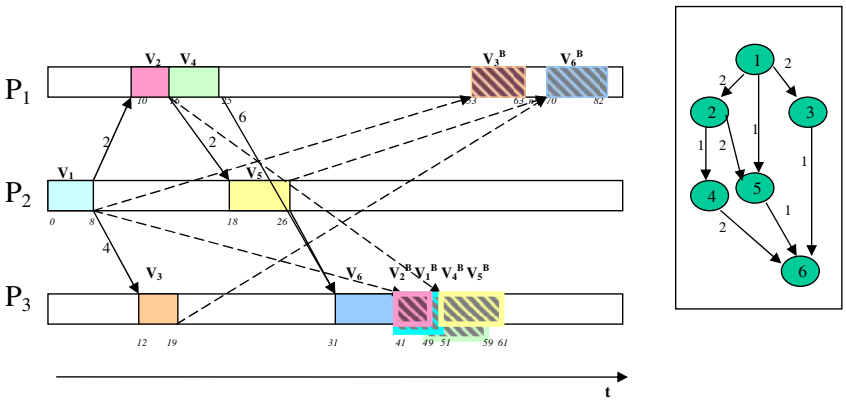

Figura 2. Exemplo de Escalonamento para um GAD.

5. Ferramenta em MPI

A implementação da ferramenta proposta está dividida em duas etapas. A primeira etapa descrita neste artigo tem como objetivo principal testar a estratégia de escalonamento tolerante a falhas em um ambiente real. Com isso, este esquema proposto é avaliado em um escopo menor, uma rede local ou cluster, para depois, ser aplicado a um ambiente hierárquico como uma grade computacional. A estrutura geral da ferramenta na primeira etapa é apresentada na Figura 4.

A ferramenta recebe como entrada os arquivos contendo a informação gerada pelo algoritmo de **escalonamento estático tolerante a falhas**. Para cada processador, um arquivo mostra a maneira em que foram escalonadas as cópias primárias e backups. O escalonamento das cópias *backups*, como visto na Figura 2, descreve como será o comportamento da aplicação em caso de falha. Esta saída apresenta diferentes combinações ou possíveis execuções da aplicação. Como uma única falha de processador é tolerada, somente uma destas combinações será disparada se a falha ocorrer. Informações sobre o redirecionamento das mensagens em caso de falhas também são geradas pelo escalonador em outro arquivo e descrevem a relação de precedência imediata (envio de mensagens) entre *backups* e primárias.

Como aparece na Figura 4, uma **biblioteca** para detecção e recuperação de falhas faz parte dos quatro elementos principais da ferramenta. Esta biblioteca redefine

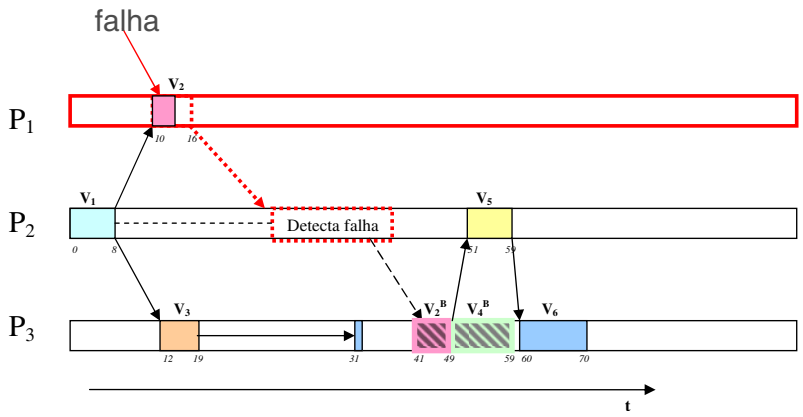

Figura 3. Exemplo da re-execução da aplicação em caso de P_1 falhar.

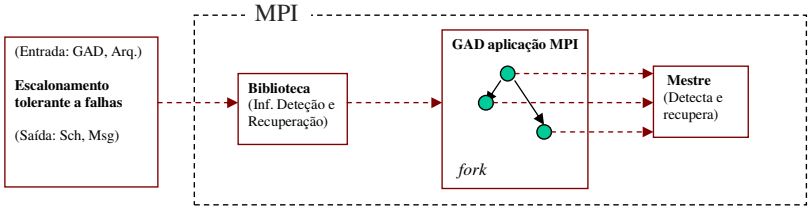

Figura 4. Estrutura da ferramenta em MPI da primeira etapa.

o conjunto de funções MPI que a **aplicação**, representada por um GAD, pode usar. A redefinição destas funções permite tratar a tolerância a falhas em conjunto com um processo **mestre** que monitora e processa a ocorrência de falhas no sistema. Este processo mestre controla a detecção e recuperação de falhas, no caso de falha, se estabelece a comunicação dos processos da aplicação com o processo mestre enviando informação necessária.

Para a aplicação tornar-se tolerante a falhas precisa primeiro ser compilada com a biblioteca e incluir os arquivos necessários com a informação do mestre e do mecanismo de detecção e recuperação. Para executar a aplicação, nenhum código é alterado e como resultado da compilação, a ferramenta de tolerância a falhas é automaticamente inserida dentro da aplicação. Consequentemente, a aplicação compilada está pronta para executar e apresenta a capacidade de recuperar-se em caso de falha.

Para detectar a falha, o processo mestre pode usar diferentes mecanismos os quais podem ser classificados em mecanismos de detecção internos, externos e híbridos. Entre os mecanismos internos que oferece MPI temos a utilização de *error handlers* para comunicações do tipo ponto-a-ponto [Lumsdaine et al. 2002]. De mecanismos externos, chamamos qualquer implementação fora do MPI, neste caso, temos por exemplo, a implementação do *Heartbeat* [Aguilera et al. 1997] ou o acesso ao monitoramento do NWS [Wolski et al. 1999]. No terceiro grupo, encontram-se os mecanismos híbridos que misturam vantagens de mecanismos internos com externos, permindo uma detecção mais forte.

Inicialmente, foi implementada a detecção utilizando um mecanismo interno do MPI: o uso de *error handlers*. Cada função da aplicação, redefinida na biblioteca MPI, interceptam os erros que ocorrem durante a execução da aplicação. Funções de

comunicação estão sendo usadas para identificar a ocorrência de erros em operações de envio e recebimento seja entre os processos da aplicação ou entre os processos da aplicação e o processo mestre. A identificação de falhas pelo MPI é habilitada através de *error handlers* que são associados aos comunicadores. Para que seja possível identificar e tratar erros, é associado a cada comunicador o *error handler MPI_ERRORS_RETURN*, o qual permite que as funções MPI retornem um código de erro na ocorrência de falhas.

Durante a detecção, as funções de envio (MPI_Send) e de recebimento (MPI_Recv) capturam os erros produzidos durante a execução da aplicação. Existem 3 casos diferentes que são tratados neste mecanismo. Os dois primeiros acontecem durante a comunicação ponto-a-ponto entre dois processos da aplicação. No primeiro caso, um processo envia uma mensagem a um outro processo mediante a função MPI_Send, a qual detecta se o outro processo falhou. O segundo caso, acontece também entre dois processos da aplicação, no entanto a detecção é realizada no ponto extremo da comunicação com a função MPI_Recv. No último caso, a comunicação é feita entre um processo da aplicação e o processo mestre. A função MPI_Recv no processo mestre detecta se o processo da aplicação falhou.

Para a recuperação da aplicação, o mestre em conjunto com o código das funções redefinidas que formam parte da biblioteca, permitem a continuidade da execução da aplicação em caso de falha. O mestre tem uma tabela com as mensagens recentes que a utiliza em caso de falha. As funções redefinidas fazem o re-encaminhamento das mensagens segundo a informação obtida nos arquivos gerados pelo escalonamento estático. Durante a recuperação, a aplicação continua a sua execução ativando as cópias backups escalonadas nos processadores indicados, apartir dos arquivos de entrada.

6. Ambiente de Testes

Para avaliar o desempenho da ferramenta estão sendo realizados vários testes sobre um *cluster* de computadores do Grid Sinergia da UFF e para uma aplicação alvo como descrito no modelo. O *cluster* utilizado é composto por 8 processadores Pentium IV 2.6 GHz com 512 Mb de RAM, executando Linux Fedora Core2, Globus toolkit 2.4 e MPI LAM 7.0.6, interconectados por uma rede Gigabit Ethernet.

Os estudos comparativos no ambiente tem como objetivo analisar quanto o desempenho da aplicação é afetado em caso de falha de um processador ao disparar-se os mecanismos de detecção automática de falhas e recuperação na ferramenta. Para testar a tolerância a falhas é simulada a ocorrência de uma falha permanente de processador mediante a execução de um arquivo *script* que contém a chamada ao comando *killall*, para matar os processos que formam parte do processador com falha. Este arquivo deve ser disparado enquanto a aplicação ainda está sendo executada.

Os testes estão sendo realizados com o ambiente de execução em modo exclusivo e principalmente para 3 cenários de execução da aplicação: sem a ferramenta, com a ferramenta sem falha e com a ferramenta com falha. Medições de tempo de execução e do número de mensagens são registradas, variando o número de tarefas e o número de processadores dedicados. Desta maneira, estuda-se o comportamento de parâmetros como *delay* e os *overheads* introduzidos pelo mecanismo de detecção e recuperação durante a execução da aplicação em caso de falha.

No primeiro cenário a aplicação não contém a ferramenta e para execução utiliza

o escalonamento baseado na heurística *list scheduling*, como visto na sessão anterior. Este escalonamento é feito selecionando-se uma tarefa livre de mais alta prioridade e um processador ocioso para alocar essa tarefa. Uma tarefa é considerada livre quando todos os seus predecessores imediatos já foram escalonados. Já no segundo caso, como resultado da compilação, a ferramenta de tolerância a falhas é automaticamente inserida dentro da aplicação e é utilizado o mesmo escalonamneto do cenário anterior. No terceiro cenário a aplicação é similar ao caso anterior, no entanto, ocorre uma falha de processador.

7. Discussões e Trabalhos Futuros

Em resultados preliminares obtidos observa-se que o tempo de execução e o número de mensagens aumentam com o incremento do número de tarefas. A execução da aplicação testada (Eliminação Gausseana) sem a ferramenta obteve melhor desempenho quando comparada com as execuções dos outros cenários com a ferramenta. A aplicação com a ferramenta, embora sem falha envolve troca de mensagens com o processo mestre ao longo da execução, o que acarreta um custo maior quando comparado com o primeiro cenário. No terceiro caso, ao ocorrer uma falha de processador, além da troca de mensagens entre os processos da aplicação e o mestre, é necessário o reencaminhamento de novas mensagens para ativar as cópias backups, assim como é necessário um tempo extra acrescentado ao tempo final esperado de execução para realizar o tratamento da falha.

Nesta análise inicial, métricas como tempo de execução e número de mensagens não apresentam um aumento tão considerável quanto os *overheads* que foram introduzidos variando a quantidade de tarefas e o número de processadores. Nesta etapa estuda-se quanto a estratégia é viável para esta arquitetura menor e como vários fatores podem influenciar no desempenho da aplicação, tais como: o número de processos criados ao mesmo tempo, o número de tarefas escalonadas por processador e o número máximo de mensagens que a estrutura de memória usada pelo processador pode armazenar.

A ferramenta implementada em MPI nesta primeira etapa tem um escopo menor voltada para funcionar dentro de um *cluster* de computadores. Uma segunda etapa será estendê-la a um grade computacional com a adaptação ao SGA do Projeto EasyGrid [Boeres et al. 2004a] da UFF e incluir também a implementação de algoritmos de escalonamento dinâmico com critérios de tolerância a falhas para replicação passiva, analisando-se a vantagem de utilizar outros tipos de heurísticas de escalonamento. Em relação ao modelo de falhas, levar-se-á consideração a ocorrência de múltiplas falhas assim como de outros tipos de falhas.

Referências

Aguilera, M. K., Chen, W., and Toueg, S. (1997). Heartbeat: A timeout-free failure detector for quiescent reliable communication. In *WDAG '97: Proceedings of the 11th International Workshop on Distributed Algorithms*, pages 126–140, London, UK. Springer-Verlag.

Al-Omari, R., Somani, A. K., and Manimaran, G. (2005). An adaptive scheme for fault-tolerant scheduling of soft real-time tasks in multiprocessor systems. *J. Parallel Distrib. Comput.*, 65(5):595–608.

Anderson, M. and Jonsson, J. (2000). Dynamic replication decision in fault-tolerant multiprocessor-based real-time systems. Technical report, University of Technology Goteborg Sweden.

Boeres, C., Fonseca, A. A., Vianna, B. A., Menezes, L. T., Moura, N. T., and Rebello, V. (2004a). Easygrid: towards a framework for automatic grid enabling of legacy mpi applications. *Concurrency And Computation Practice And Experience*, 16(5):425–432.

Boeres, C., Fonseca, A. A., Vianna, B. A., Menezes, L. T., Moura, N. T., and Rebello, V. (2004b). Um ambiente para o desenvolvimento e avaliação de algoritmos de escalonamento para grades computacionais. In *V WSCAD 2004*, Foz do Iguacu.

Boeres, C., Lima, A., and Rebello, V. (2003). Hybrid task scheduling: Integrating static and dynamic heuristics. In *Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing*, pages 199–206, Washington, DC, United States. IEEE Computer Society.

Boeres, C., Nascimento, A. P., Sena, A. C., and Rebello, V. (2005). Efficient hierachical self-scheduling for mpi applications executing in computacional grids. In *3rd International Workshop on Middleware for Grid Computing*, New York.

da Silva Martins Jr., A. and Gonçalves, R. A. L. (2005). Extensões na lam/mpi para automatizar o checkpoint e tolerar falhas em cluster de computadores. In *VI WSCAD 2005*, Rio de Janeiro.

Dima, C., Girault, A., Lavarenne, C., and Sorel, Y. (2001). Off-line real-time fault-tolerant scheduling. In *Euromicro Workshop on Parallel and Distributed Processing*, pages 410–417, Mantova, Italy.

Ghosh, S., Melhem, R., and Mosse;, D. (1997). Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems. *IEEE Trans. Parallel Distrib. Syst.*, 8(3):272–284.

Hagras, T. and necek., J. (2004). A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems. In *18th International Parallel and Distributed Processing Symposium (IPDPS'04)- Workshop 1*.

Kee, Y.-S., Casanova, H., and Chien, A. A. (2004). Realistic modeling and svnthesis of resources for computational grids. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 54, Washington, DC, USA. IEEE Computer Society.

Kwok, Y.-K. and Ahmad, I. (1999). Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys*, 31(4).

Liberato, F., Melhem, R., and Mosse;, D. (2000). Tolerance to multiple transient faults for aperiodic tasks in hard real-time systems. *IEEE Trans. Comput.*, 49(9):906–914.

Lumsdaine, A., Squyres, J. M., and Barrett, B. (2002). Reliability in lam/mpi requirements specification. Technical report, Indiana University Department of Computer Science.

Naedele, M. (1999). Fault-tolerant real-time scheduling under execution time constraints. In *RTCSA '99: Proceedings of the Sixth International Conference on Real-Time Com-*

puting Systems and Applications, page 392, Washington, DC, USA. IEEE Computer Society.

Pearlman, L., Kesselman, C., Gullapalli, S., B. F. Spencer, J., Futrelle, J., Ricker, K., Foster, I., Hubbard, P., and Severance, C. (2004). Distributed hybrid earthquake engineering experiments: Experiences with a ground-shaking grid application. In *HPDC '04: Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC'04)*, pages 14–23, Washington, DC, USA. IEEE Computer Society.

Qin, X., Jiang, H., and Swanson, D. R. (2002). An efficient fault-tolerant scheduling algorithm for real-time tasks with precedence constraints in heterogeneous systems. In *ICPP '02: Proceedings of the 2002 International Conference on Parallel Processing (ICPP'02)*, page 360, Washington, DC, USA. IEEE Computer Society.

Snir, M., Otto, S. W., Walker, D. W., Dongarra, J., and Huss-Lederman, S. (1995). *MPI: The Complete Reference*. MIT Press, Cambridge, MA, USA.

Wolski, R., Spring, N., and Hayes, J. (1999). The network weather service: A distributed resource performance forecasting service for metacomputing . *Journal of Future Generation Computing Systems*, 15(5-6):757–768.

Sessão Técnnica 2 /
Technical Session 2

ZeliGrid: uma arquitetura para a implantação de aplicações com requisitos não-funcionais dinâmicos em grades computacionais

Rodrigo Souza Granja[1], Alexandre Sztajnberg[1,2]

[1]DICC/IME e [2]PEL/FEN – Universidade do Estado do Rio de Janeiro
Rua São Francisco Xavier, 524, 6018-D – Maracanã, Rio de Janeiro

{rodrigos, alexszt}@ime.uerj.br

Abstract. *The management of resources is an important aspect to be considered regarding applications that might have different non-functional or operational requirements, when running in distributed and heterogeneous environments, such as the Computational Grids. In this context it is necessary to provide means to specify the required resource constraints and an infrastructure that can adapt the applications in face of changes on the recource avalilability. In this paper we adopt the CR-RIO approach to deploy parallel applications that have non-functional requirements. To provide the supporting infrastructure we integrate some of the Globus services and NWS monitoring services to the concept of QoS Contracts in the form of an architecture. Deployment and implementation details are discussed.*

Resumo*. A gerência de recursos é um dos aspectos importantes a serem considerados na execução de aplicações com diferentes requisitos não-funcionais ou operacionais em ambientes computacionais heterogêneos e distribuídos, como os das Grades Computacionais. Neste contexto são necessários meios para especificar as necessidades de recursos e uma infra-estrutura que permita que as mesmas possam se adaptar frente a mudanças na disponibilidade destes recursos. Neste artigo adotamos a abordagem CR-RIO para a implantação de aplicações paralelas com requisitos não-funcionais. Como infra-estrutura de suporte integramos serviços do Globus e serviços de monitoramento do NWS ao conceito de Contratos de QoS, na forma de uma arquitetura. Detalhes de implantação e implementação são discutidos.*

1. Introdução

Os recursos compartilhados por aplicações que executam em grades computacionais podem variar em suas características e disponibilidade, e nem sempre estão dedicados a estas aplicações [Foster 2001]. A disponibilidade destes recursos pode variar dinamicamente ao longo da execução das mesmas. Neste contexto, o desenvolvimento de alguns tipos de aplicação para Grades torna-se um desafio, visto que em tempo de projeto não é possível determinar qual será seu ambiente real de execução. Em especial, estamos preocupados com as aplicações com requisitos não-funcionais, que precisam de certa quantidade de recursos (por exemplo, largura de banda ou capacidade de processamento) para que funcionem como esperado. Surge, assim, a necessidade de uma infra-estrutura que ofereça serviços para a aplicação descobrir e caracterizar os

recursos disponíveis, selecionar e acessar os que atendem a seus requisitos não-funcionais. Também são necessários mecanismos para adaptar dinamicamente aplicação caso os recursos disponíveis não estejam mais de acordo com os requisitos especificados.

Em [Granja 2005] apresentamos os conceitos gerais de uma arquitetura de suporte que integra um *middleware* de suporte para grades computacionais, o Globus Toolkit, e um sistema de coleta de informações de recursos distribuídos, o Network Weather Service (NWS), para oferecer os serviços básicos de descoberta e indexação de recursos, e implantação remota de processos. Com base nestes serviços, propomos uma arquitetura que utiliza o conceito de Contratos de QoS (*quality of service*) discutido em [Loques 2004], para criar os mecanismos necessários para o funcionamento das aplicações com requisitos não-funcionais. No presente artigo apresentamos uma implementação e refinamentos introduzidos a arquitetura original, agora chamada de ZeliGrid, e discutimos alguns resultados.

O restante do texto está estruturado da seguinte forma. Na Seção 2, apresentamos os conceitos gerais de CR-RIO. Na Seção 3 relacionamos as ferramentas integradas em nossa arquitetura: o Globus e o NWS. Na Seção 4, arquitetura proposta é apresentada e na Seção 5 abordamos sua implantação. Na Seção 6 discutimos sua implementação e alguns resultados. Na Seção 7 relacionamos trabalhos relacionados e na Seção 8 observações finais.

2. CR-RIO

O *framework* CR-RIO (*Contractual Reflective - Reconfigurable Interconnectable Objects*)integra o paradigma de arquitetura de software, centrado numa linguagem de descrição de arquiteturas (ADL), a conceitos tais como a capacidade de reflexão e adaptação dinâmica [Loques 2004]. Ele inclui um Configurador, elemento reflexivo que oferece serviços para instanciar, executar e re-configurar aplicações distribuídas, cujas arquiteturas foram descritas da ADL CBabel. Para especificar aspectos não-funcionais adotamos o conceito de Contrato de QoS, em que duas partes expressam seus requisitos não-funcionais, através de serviços e parâmetros operacionais requeridos, regras de negociação e de adaptação.

A arquitetura de suporte do CR-RIO é composta por um conjunto de elementos que são: o Gerenciador de Contratos (GC), o *Contractor* e Agentes de Recursos (AR). O GC interpreta a descrição de um contrato e extrai desta a máquina de estados de negociação de serviços. Inicialmente, o GC identifica qual serviço será negociado e envia as descrições de configuração, e os perfis associados, ao *Contractor* em cada nó participante. O *Contractor* tem duas responsabilidades principais: (a) traduzir as propriedades dos perfis associados aos serviços do nível de arquitetura em serviços de suporte do nível de sistema, e requisitar estes serviços, com os parâmetros adequados, aos ARs e (b) receber notificações dos ARs. Um AR encapsula o acesso aos mecanismos de nível de sistema, provendo interfaces para efetivar a alocação de recursos, iniciar serviços locais de sistema e monitorar os valores de propriedades requeridas. Se um serviço não pode ser implantado ou não pode ser mais provido, um outro serviço é negociado, segundo uma política de negociação descrita no Contrato de QoS. Detalhes sobre a arquitetura de suporte CR-RIO em [Corradi 2005].

Nossa abordagem para empregar os conceitos de CR-RIO em um ambiente de grades procurou integrar elementos e serviços já estabelecidos nestes ambientes, e adicionar os elementos de CR-RIO necessários para gerenciar os contratos. A Tabela 1 apresenta o mapeamento entre os conceitos de CR-RIO e ZeliGRID. Os detalhes deste mapeamento serão discutidos ao longo das Seções 4 e 5.

Tabela 1. Mapeamento dos elementos de CR-RIO para Zeligrid

CR-RIO	**ZeliGRID**
ADL e Contratos Não-funcionais	Arquitetura mestre-escravos e Perfis de Recursos
Configurador	Modulo de Gerência Remota de Processos (MGRP)/ GRAM
Contractor	Módulo de Gerência de Recursos (MGR) / LDAP / GIIS
Agente de Recurso (AR)	NWS / GRIS
Gerente de Contratos	Módulo de interpretação de regras de contrato (MIRC) / Módulo de Controle e reconfiguração de Experimento (MCRE)

3. Infra-estrutura de suporte

Destacamos nesta seção os serviços do Globus e NWS utilizados em nossa arquitetura.

Globus Toolkit. O Globus Toolkit é um *middleware* que oferece os serviços básicos propostos na arquitetura de Grades descrita em [Globus 2006]. Das ferramentas que o compõe, usamos:

- o MDS (*Monitoring and Discovery Service*), um serviço de diretórios para Grades que é responsável pela indexação e consulta das informações dos recursos computacionais disponíveis. Ele é formado pelo *Grid Resource Information Service* (GRIS) e o *Grid Index Information Service* (GIIS);
- o GRIS, um *framework* implementado como um servidor OpenLDAP que pode ser configurado "plugando-se" em diversas fontes de informações. Cada recurso pode rodar uma instância do GRIS localmente, e esta é responsável pela coleta das informações neste recurso;
- o GIIS, um *framework* usado para construção de diretórios, aceitando mensagens de registros tanto de instâncias de provedores GRIS quanto de outras instâncias do GIIS. Essas informações são colocadas em espaço comum de nomes e organizadas de forma hierárquica, o que permite que as consultas realizadas por clientes do GIIS possam retornar dados de uma ou até de todas as instâncias GRIS e GIIS registradas;
- o GRAM, responsável pela submissão e execução de processos na Grade, oferecendo uma API para a realização destas operações e suporte para o uso da linguagem RSL (*Resource Specification Language*), estruturada para especificar os parâmetros necessários para que esta aplicação execute remotamente.

Java Commodity Grid Kit. O *Commodity Grid Kit* ou CoG Kit é um *framework* de desenvolvimento para aplicações em Grades. Ele oferece aos desenvolvedores uma API em Java para acesso aos serviços oferecidos pelo Globus em um nível mais alto de programação. Em nosso projeto usamos a versão em Java do CoG kit.

Network Weather Service. O NWS é um sistema distribuído que realiza medições e faz previsões da disponibilidade de vários tipos de recursos computacionais, tais como

memória, processador e recursos de comunicação [Wolski 1999]. Ele atua através de sensores distribuídos que coletam informações diretamente dos recursos e as armazenam em "servidores de memória". Em nossa arquitetura o NWS interage diretamente com o servidor *slapd* [Open 2006], através do Nwsldap, como descrito em [Wolski 2006]. Para isso, é instanciado um servidor *slapd* que se comporta como um servidor de informações que coleta as informações do NWS e é acessado diretamente por consultas feitas no formato LDAP [Yegon 1995].

4. A arquitetura proposta

Os elementos descritos anteriormente formam a base da infra-estrutura empregada. Sobre esta base operam os módulos adicionais propostos neste trabalho, segundo a abordagem de CR-RIO, constituindo a arquitetura de suporte que oferece os serviços necessários para a implantação de aplicações paralelas com requisitos não-funcionais.

Seguindo a Figura 1, o NWS e os provedores padrão GRIS atuam acima dos recursos, monitorando e coletando informações destes. O NWS coleta dados dinâmicos, como disponibilidade de CPU e latência de canais de comunicação TCP, e os provedores GRIS coletam dados estáticos, como o sistema operacional e a quantidade e características dos processadores de uma máquina.

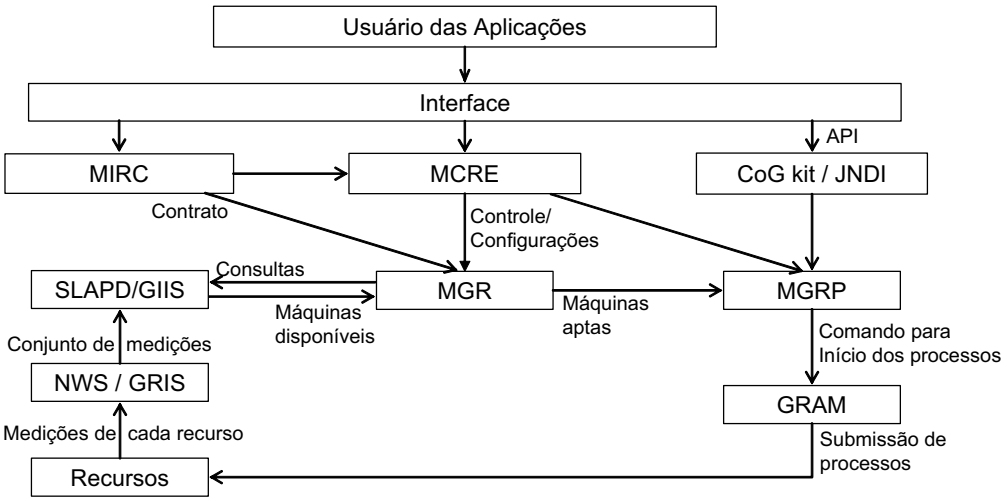

Figura 1. Arquitetura de suporte

Os dados coletados juntos aos recursos pelas múltiplas instâncias do GRIS são disponibilizados através do GIIS, enquanto aqueles coletados pelo NWS são disponibilizados através de um servidor *slapd*. O MDS atua como um serviço de diretório, indexando os recursos e especificando quais informações eles disponibilizam e, quando requisitado, busca esses dados sob demanda [Wolski 2006]. O GRAM é responsável pela carga e execução remota de processos.

Interagindo com esses módulos, estão módulos desenvolvidos neste projeto: o MIRC (módulo de interpretação de regras de contratos); o MGR (módulo de gerência de recursos); o MGRP (módulo de gerência remota de processos); e o MCRE (módulo de controle e reconfiguração de experimentos), detalhados nas próximas subseções.

4.1 Módulo de interpretação de regras de contrato (MIRC)

O MIRC é responsável pela interpretação e execução das regras estabelecidas pelo contrato de requisitos não-funcionais da aplicação. Este contrato deve ser fornecido em um arquivo de texto e a princípio deve ser elaborado em tempo de projeto, descrevendo as características esperadas dos recursos que compõe o ambiente de execução.

Um contrato é composto por um conjunto de perfis, onde cada perfil representa um conjunto de características, através de uma lista de atributos. Cada atributo contém um nome e um valor, de forma similar aos atributos de registro do modelo LDAP [Yeong 1995], usado no NWS. Cada contrato possui mais de um perfil para que a aplicação possa mudar suas configurações caso o perfil atual não possa (mais) ter todos seus requisitos atendidos.

A Figura 2 apresenta um exemplo de contrato. Neste são especificados dois perfis de recursos (linhas 02 e 13), cada um deles com dez atributos necessários: largura de banda de conexões TCP (linhas 03 e 14); tipo de arquitetura da CPU (linhas 04 e 15); disponibilidade da CPU (linhas 05 e 16); número de processadores (linhas 06 e 17); freqüência do *clock* da CPU em MHz (linhas 07 e 18); tamanho da memória cachê L2 (linhas 08 e 19); memória livre (linhas 09 e 20); plataforma da máquina (linhas 10 e 22); Nome do sistema operacional (linhas 11 e 21) e total de memória RAM disponível (linhas 12 e 23).

```
01   # Contract Example
02   [profile1]                          13   [profile2]
03      bandWidth>78                     14      bandWidth>40
04      cpuArchitecture=IA32             15      cpuArchitecture=IA32
05      cpuAvail>0.6                     16      cpuAvail>0.4
06      cpuCount>1                       17      cpuCount=1
07      cpuSpeed>2000                    18      cpuSpeed>1000
08      cpuL2cache>500                   19      cpuL2cache>500
09      freeMemory>200                   20      freeMemory>100
10      hostPlataform=i686               21      osName=Linux
11      osName=Linux                     22      hostPlataform=i686
12      totalMemory>400                  23      totalMemory>200
```

Figura 2. Exemplo de contrato de QoS

Nossa implementação utiliza uma versão simplificada de contratos [Loques04], em que cada contrato tem apenas dois perfis. O primeiro perfil representa os requisitos ideais para o funcionamento da aplicação e o segundo os requisitos mínimos. Dessa forma, a aplicação sempre irá priorizar as máquinas que atendem os requisitos do primeiro perfil e, caso estes não possam ser atendidos, o segundo perfil é usado para a avaliação. Se uma máquina que executa um processo remoto não atender mais às especificações do segundo perfil, a aplicação é reconfigurada, realocando este processo para outra máquina que tenha recursos disponíveis de forma a atender ao primeiro perfil do contrato. Se não houver uma máquina disponível que atenda este, então é procurada uma que atende o segundo perfil e, caso não exista nenhuma disponível, a aplicação Zeligrid aguarda um tempo entre 10 e 300 segundos para uma nova tentativa. Este tempo é configurado pelo usuário, como descrito na Subseção 4.4.

4.2 Módulo de Gerência de Recursos (MGR)

O MGR é o responsável pela descoberta, gerência e avaliação dos recursos que são usados no experimento. Para isso, ele oferece aos outros módulos funções para a busca

dos recursos registrados nos servidores NWS e MDS, além de funções para a verificação do estado desses recursos de acordo com as especificações dos perfis do contrato de QoS. Como mostrado na Figura 1, este módulo interage diretamente com os módulos SLAPD e GIIS realizando consultas e recebendo destes as máquinas disponíveis. O MGR também recebe os contratos analisados pelo MIRC.

Logo após obter estas informações, o MGR já está em condições de avaliar, por comparação, as máquinas que estão aptas para a execução de acordo com o contrato recebido do MIRC. Para isso, ele oferece uma função que analisa todas as máquinas disponíveis de acordo com um dos perfis do contrato e retornam apenas as máquinas que atendem as especificações deste, além de classificar todos os atributos das máquinas como aptos, não-aptos e não avaliados, que é a situação dos atributos que não constam no perfil especificado.

4.3 Módulo de Gerência Remota de Processos (MGRP)

O módulo de iniciação remota de processos é o responsável por submeter os processos para as máquinas aptas selecionadas pelo MGR, recebendo deste uma lista com os dados das máquinas que estão aptas, de acordo com o contrato interpretado pelo MIRC, e submetendo os processos de acordo com as configurações do MCRE.

O MGRP oferece duas funções para submissão de processos remotos, que permitem tanto a submissão de apenas um único processo para uma máquina quanto a submissão de vários processos para diferentes máquinas. O MGRP também disponibiliza funções para o cancelamento da execução remota de processos, tanto individual quanto coletiva.

4.4 Módulo de Controle e Reconfiguração de Experimentos (MCRE)

O MCRE é responsável pelo controle das aplicações (denominados *experimentos* em ZeliGrid) executando em ZeliGrid. Sua primeira função é permitir a configuração do experimento, possibilitando que o usuário especifique (i) o número de máquinas necessárias para a execução do experimento; (ii) qual será a máquina que irá executar o processo mestre; (iii) dados das aplicações como: localização do arquivo executável, argumentos, se suas saídas de dados (STDOUT, STDERR) devem ser redirecionadas, permitindo diferentes configurações para os processos mestre e escravos [Magee 1999]; (iv) especificar a localização dos servidores NWS e MDS utilizados; (v) o tempo entre as verificações do estado de cada máquina; (vi) o tempo entre tentativas de reconfiguração caso não existam máquinas disponíveis; (vii) se um processo escravo que for concluído deve ser reiniciado e (viii) tempo de validade de uma medição feita pelo NWS.

Além disso, a outra função do MCRE é coordenar o experimento, utilizando as funções do módulo MGR e MGRP. Através do MGR, ele coleta uma lista com as referências para as máquinas necessárias para a execução do experimento. Todas elas devem atender as especificações do contrato de QoS fornecido e devem ser suficientes para atender o número mínimo de máquinas especificadas na configuração do experimento.

Utilizando as funções do MGRP, ele inicia um processo na máquina escolhida como a mestra e logo em seguida os processos escravos, que são executados nas

máquinas fornecidas pelo MGR, que também é o responsável pelo armazenamento da lista com as máquinas que estão sendo usadas no momento. Desta mesma forma, esse módulo realiza a reconfiguração da aplicação, verificando o estado das máquinas que estão sendo usadas e, caso estas não estejam mais de acordo com as especificações do contrato, estas são transferidas para outra máquina que esteja apta.

5. Implantação

São previstos 5 tipos de nós distribuídos ao se implantar uma aplicação em ZeliGrid (Figura 3). O primeiro tipo, chamado de *Recursos*, representa as máquinas onde serão executados os diversos processos da aplicação. Além dos processos da aplicação, em cada nó Recursos devem executar os sensores do NWS e uma instância GRIS.

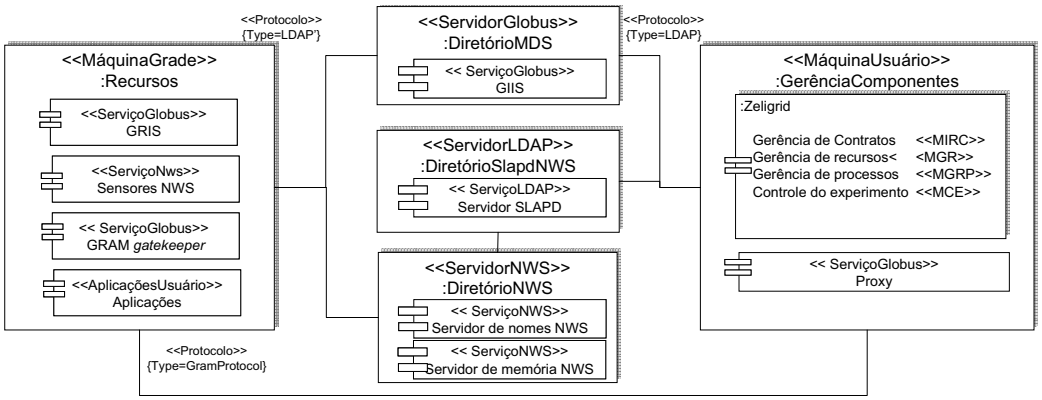

Figura 3. Diagrama de implantação do experimento

Os sensores NWS devem ser configurados para se registrar com o servidor de nomes NWS, presente no nó *DiretórioNWS*, e devem executar três atividades para o monitoramento de disponibilidade de CPU, latência de conexão TCP e largura de banda entre os nós e a máquina que será usada como mestre, conforme os procedimentos descritos em [Obertelli 2006]. Já a instância GRIS, deve ser configurada para se registrar no nó *DiretórioMDS*, onde está configurado o GIIS principal da grade, conforme os procedimentos descritos em [Globus 2006]. Além disso, deve existir uma instância do GRAM *gatekeeper*, que é responsável pelo controle da execução das aplicações remotas em cada máquina. Ela deve ser configurada pelo superusuário (*root*) do nó para executar como um serviço do sistema.

Os nós chamados *DiretórioMDS* e *DiretórioNws* são repositórios de informações relativas aos estados dos nós *Recursos*, sendo que o primeiro armazena informações coletadas dos servidores GRIS e o outros dos sensores NWS. Em cada implantação existe apenas um nó de ambos os tipos, podendo haver mais de uma instância apenas para propósitos relacionados a tolerâncias à falhas. O nó *Diretório SlapdNWS* é usado para que as informações armazenadas no nó *DiretórioNWS* possam ser acessada através de consultas LDAP [Wolski 2006].

O nó *Máquina do Usuário* controla a arquitetura ZeliGrid, que executa os módulos de gerência (Seção 4). Neste nó, o controlador do experimento irá configurar e controlar a aplicação, por isso sendo necessário que nela estejam presentes as bibliotecas do CoG kit, o JRE 5.0 [Sun 2006] e o Contrato de QoS que será usado. Nesta máquina também deve residir um certificado, autenticado pela Autoridade

Certificadora da Grade, para que se possa criar o *proxy*, utilizado para iniciar os serviços autenticados na Grade. O *proxy*, por ter um prazo curto de validade, precisa ser recriado com freqüência, através do comando *grid-proxy-init* do Globus [Globus 2006].

Observa-se, ainda, que para garantir a consistência dos registros GRIS e GIIS é necessário que todas as máquinas da grade tenham seus relógios sincronizados. Em nossa implantação isso foi feito através de um cliente NTP [Mills 1992] para Linux.

6. Implementação

A implementação da arquitetura ZeliGrid foi feita em Java, utilizando as APIs do CoG Kit para acesso aos serviços do MDS e a API JNDI da Sun [Sun 2006] para acesso aos diretórios LDAP. A organização dos módulos apresentados anteriormente se reflete na implementação, onde existe um pacote para cada um dos quatro módulos. Outros dois pacotes contêm as classes responsáveis pela interface gráficas e as classes utilitárias. No total a implementação conta com 21 classes que, em conjunto, possuem cerca de 5.500 linhas código, ignorando os comentários e códigos de documentação.

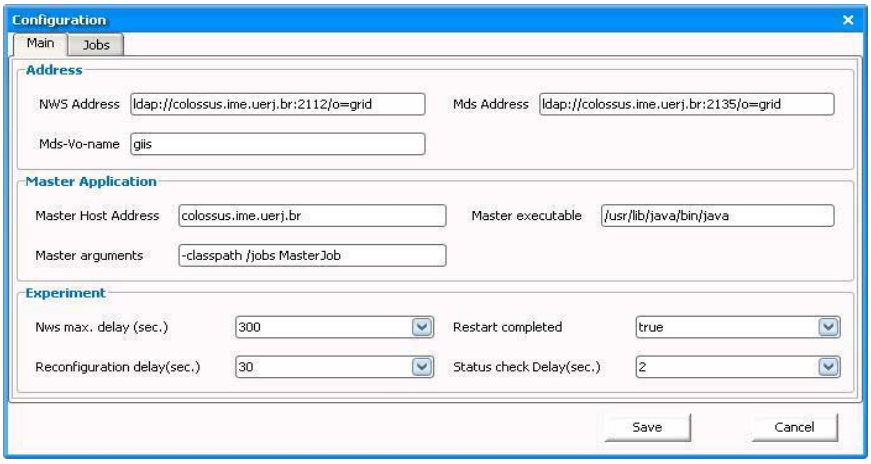

Figura 4. Painel de configuração de uma aplicação

6.1 Interface Gráfica

ZeliGrid oferece uma interface gráfica usada para configurar, implantar e gerenciar as aplicações. Ela permite ao usuário que a configuração do experimento seja feita de forma amigável e persistente e também permite o acompanhamento visual do estado de todas as máquinas e da execução de processos remotos.

A Figura 4 apresenta o painel de configuração da aplicação. Neste painel o usuário pode ajustar as configurações do experimento discutidas na Subseção 4.4, tais como a localização dos servidores Slapd e GIIS, o número de nós a ser utilizado para distribuir os processos escravos (na aba *Jobs*, não mostrada na figura) e a freqüência de verificação dos requisitos não-funcionais. Este painel é ativado a partir do principal.

O painel principal de gerência do experimento, Figura 5, permite ao usuário ativar o painel de configuração do experimento, indicar e inspecionar os contratos, monitorar a operação dos diversos processos distribuídos e controlar a operação do experimento. Também pode obter detalhes da versão do programa e ajuda (menus *Experiment* e *About*).

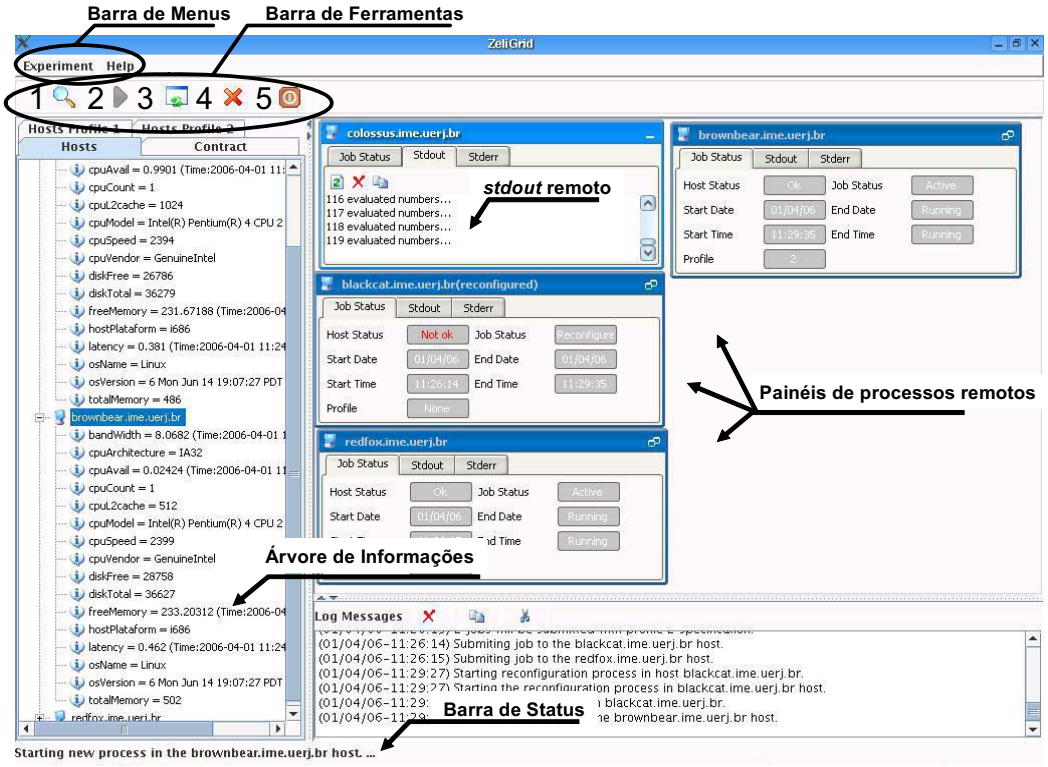

Figura 5. Interface principal do ZeliGrid

A **barra de ferramentas** possui cinco botões (numerados de 1 a 5) usados para controlar o experimento e que possuem as seguintes funções: (a) *Buscar Máquinas*, inicia uma *Thread* que busca por máquinas registradas nos servidores NWS e MDS, obtendo seus atributos e organizando em árvores; (b) *Iniciar Experimento*, inicia uma *Thread* que executa o experimento; (c) *Atualiza lista de máquinas*, que atualiza todos os atributos das máquinas registradas e verifica a presença de novas; (d) *Cancelar o experimento*, cancela todos os processos executados remotamente e retira todos painéis de nós remotos da aplicação; e (e) *Fecha a aplicação*.

A **árvore de informações** apresenta informações sobre as máquinas disponíveis e o contrato de QoS em uso. Ao exibir as máquinas 3 diferentes perspectivas podem ser usadas: (i) mostrar apenas os atributos das máquinas; (ii) mostrar os atributos avaliados de acordo com a especificação do primeiro perfil do contrato ou de acordo com o segundo perfil. Nos dois últimos tipos de perspectiva, cada atributo é marcado com uma *etiqueta* indicando se o atributo de cada recurso esta *OK*, ou está fora do limite requisitado no contrato (*NOTOK*) ou ainda se o atributo não foi especificado no contrato (*Not Evaluated*).

Se houver algum atributo que não esteja de acordo com o contrato, a máquina onde se encontra o recurso sendo exibido é marcada como *NOTOK* e não será usada para a execução de processos, se não houver, ela é marcada como *OK* e pode ser uma das usadas, de acordo com a demanda. Além disso, a máquina que foi configurada como sendo a responsável pela execução do processo mestre é marcada com a etiqueta (*Master*), que indica que independente de seu estado, ela será usada para a execução deste processo.

A janela de **mensagens de *log***, que registra todas as ações realizadas pelo programa, destacando com cor vermelha as mensagens de erro. Já a barra de status, mostra mensagens sobre ações que estão sendo realizadas ou a última ação realizada.

O restante da área do painel principal é usado para exibir painéis informando o **estado corrente dos processos executados** remotamente e, casos tenham sido redirecionados, as mensagens de *stdout* e o *stderr* dos mesmos. Na Figura 5 são exibidas as informações de 4 processos, cada qual executando em uma máquina diferente.

6.2 Estado atual e desempenho da infra-estrutura

Com o objetivo de avaliar o funcionamento da arquitetura e obter algumas medidas de desempenho, utilizamos uma aplicação simples, do tipo *bag-of-tasks* com topologia mestre-escravos [Magee 1999]. Os testes foram realizados em máquinas Pentium IV e Atlhon XP (mestre), 512 Mb RAM e Pentium III com 256 RAM (escravas), executando o sistema Linux Slackware 10 (kernel 2.4.26), interligadas em rede Ethernet 10/100. As estações escravas ligadas através de um roteador/*firewall* também baseado em Linux. O contrato utilizado é descrito na Figura 2 para o mestre e para os escravos.

A implantação do experimento constando de 1 processo mestre e 3 três escravos, é realizada em aproximadamente 3072 ms. Uma operação de reconfiguração completa consta de 3 etapas: (i) o tempo de cancelamento do processo que deve ser migrado (~7945 ms), (ii) a submissão da nova instância do processo na nova maquina selecionada (~528 ms) e, (iii) o *overhead* dos mecanismos de suporte de ZeliGrid (~840 ms). Observa-se que o maior gasto de tempo está associado à suspensão do processo, em que atuam o MGRP e o GRAM. Ressalta-se que a duração de uma reconfiguração reflete o tempo decorrido entre a decisão de se realizar a reconfiguração e a conclusão da mesma. Adicionalmente, para evitar a instabilidade causada por flutuações momentâneas nos estados dos recursos, que ocasionariam reconfigurações equivocadas, adotamos um algoritmo em que cada representação de máquina na estação de controle é associada a um contador inicializado com 10. Quando a máquina é classificada como *NOTOK*, este número é decrescido de 2 e quando ela é classificada como *OK*, ele é acrescido de 1, não podendo ultrapassar 10. Se este valor for igual a zero, ZeliGrid inicia o processo de reconfiguração. Observa-se que o algoritmo é *ad hoc*, mas pode ser substituído por filtros, como média móvel, ou detectores de falha. Esta verificação é feita a cada ciclo de avaliação do estado das máquinas, cujo intervalo é configurado pelo usuário, e que pode variar entre 1 e 30seg.

Na versão atual da arquitetura, o processo de reconfiguração é realizado apenas reinstanciando-se um processo para outra máquina, visto que as aplicações de teste podem ser realocadas sem a necessidade de persistência de estado. Estratégias que mantenham o estado do processo podem ser introduzidas com certa facilidade devido à modularidade da implementação. O uso de herança ou de interceptadores podem ser considerados.

7. Trabalhos Relacionados

Em [Freitas 2005] foi desenvolvida uma arquitetura baseada em contratos para servidores de tempo real adaptativos. Embora o ambiente de execução seja diferente, este trabalho também utilizou a abordagem CR-RIO e apresenta uma série de desafios

comuns, principalmente em relação aos contratos e a adaptação das aplicações a mudanças no ambiente de execução. Nesse caso, os resultados obtidos foram satisfatórios, com o tempo de resposta percebido pelo usuário sendo reduzido com o uso das técnicas de adaptação e com a alocação dinâmica de recursos de acordo com a demanda do sistema.

[Krauter 2002] apresenta uma taxonomia para sistemas de gerenciamento de recursos para Grades. Neste trabalho são comparados sistemas como o Condor-G, Legion, além do Globus, permitindo situar ZeliGrid como uma extensão de serviços mais gerais para o provimento de controle dinâmico de recursos segundo uma política associada à aplicação. Devido ao espaço reduzido apenas mencionamos outros trabalhos relacionados, como [Keahey 2003], [Shin 2003] e [Menascé 2004], além dos que foram elencados em [Loques 2004] e em [Krauter 2002].

8. Conclusões

A arquitetura ZeliGrid foi desenvolvida com o objetivo de facilitar a concepção, implantação e gerência de aplicações em Grade que possuem requisitos ou restrições em relação aos recursos disponíveis. Acreditamos que a abordagem de Contratos de QoS é adequada para esta finalidade: o projetista da aplicação consegue descrever requisitos não-funcionais e as variações aceitáveis em termos de perfis de qualidade; e a arquitetura utiliza estas informações para alocar e monitorar os recursos. Os perfis também definem uma política para o caso dos recursos não estarem disponíveis, dirigindo reconfigurações na aplicação. Neste sentido, aplicamos a abordagem CR-RIO para o contexto de Grades Computacionais, investigando como integrar a o padrão arquitetural de gerencia de Contratos de QoS aos padrões utilizados neste ambiente (e.g., Globus e NWS).

As dificuldades encontradas inicialmente, relacionadas à própria implantação do ambiente do Globus e sua integração com o NWS, foram superadas e a implementação da arquitetura do ZeliGrid permitiu que, nesta fase da investigação, nos concentrássemos na avaliação dos mecanismos de monitoramento, ajustes na periodicidade das medidas - de forma a se evitar instabilidades e na avaliação dos procedimentos de reconfiguração. Também foi possível realizar medidas nos tempos envolvidos em reconfigurações (acréscimos e realocações de processos) verificando se os mesmos são compatíveis com o desempenho esperado para aplicações em Grade (e.g., aplicações científicas) e se, no conjunto, a abordagem empregada traz benefícios para o desempenho dos experimentos.

Atualmente estamos preparando novos experimentos e adicionando estratégias de reconfiguração com a suspensão e migração dos processos de forma persistente. Isto pode ser feito inserindo código adicional nos próprios módulos, por exemplo, através de *checkpoints* ou permitindo a reconfiguração apenas após a sincronização em uma barreira. Alternativas também estão sendo investigadas, de forma a não se interferir no código da aplicação, através do uso de mecanismos reflexivos de interceptação e serialização do próprio Java [Sara 2001]. O emprego de outros sistemas de middleware, como o Condor-G, que apresenta mecanismos específicos para este tipo de operação [Michael 1997] também estão sendo considerados.

Agradecimentos. Agradecemos o apoio do PIBIC/UERJ e CNPq; o apoio da Faperj (E-26/171.130/2005). Agradecemos os revisores anônimos pelas avaliações e comentários..

9. Referências

Corradi, A. S., “Um Framework de Suporte a Requisitos Não-Funcionais para Serviços de Nível Alto”, Dissertação de Mestrado, Inst. de Computação, UFF, 2005.

Foster, I., Kesselman, C., Tuecke, S., "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", International Journ. Supercomp. Applicaions., 15(3), 2001.

Freitas G., Cardoso L., Santos A., Loques O., “Otimizando a Utilização de Servidores através de Contratos Arquiteturais”, WTR’04, Fortaleza, 2005.

Globus Alliance, “Globus Toolkit 2.4 Release Manuals”, http://www.globus.org/toolkit/docs/2.4/ (2006).

Granja, R, e Sztajnberg, A., “Um experimento integrando contratos de QoS e middleware para grades para a implantação de aplicações com requisitos não-funcionais dinâmicos”, WSCAD 2005, Outubro, 2005.

Krauter, K., Buyya, R., Maheswaran, M., “A taxonomy and survey of grid resource management systems for distributed computing”, Software-Practice and Experience, John Wiley & Sons, Vol 32, No. 2, pp: 135–164, 2002.

Keahey, K., Ripeanu, M. e Doering, K., “Dynamic Creation and Management of Runtime Environments in the Grid”, Global Grid Forum 9, Chicago. EUA, 2003.

Loques, O., Sztajnberg, A. e outros, “A contract-based approach to describe and deploy non-functional adaptations in software architectures”. Jour. of the Braz. Computer Society, Vol. 10, No. 1, pp. 5-18, Julho, 2004.

Magee, J., Kramer, J., “Concurrency: State Models & Java Programs”, Capítulo 9, John Wiley & Sons; Bk&CD-Rom edition, Abril, 1999.

Menascé, D. e Casalicchio, E., "A Framework for Resource Allocation in Grid Computing", 12th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, Volendam, Holanda, Outubro, 2004.

Michael, L. e outros, "Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System", University of Wisconsin-Madison, CS, Tec Report #1346, Abril, 1997.

Mills, David, “Network Time Protocol (Version 3) Specification, Implementation and Analysis”, RFC 1305, Março 1992.

Obertelli, G. e Wolski, R., “Network Weather Service User's Guide”, http://nws.cs.ucsb.edu/users_guide.html (2006).

Open LDAP foundation, “Software Man Pages: slapd”, http://www.openldap.org/software/man.cgi?query=slapd (2006).

Sara, B, “Making Java Applications Mobile or Persistent”. 6th USENIX COOTS'2001, San Antonio, Texas, USA, Janeiro, 2001.

Shin, K.G., Abdelzaher, T.F., e Bhatti, N., “User-level qos-adaptive resource management in server end-systems”, IEEE Transactions on Computers, Vol. 52, No. 5, Maio, 2003.

Sun Microsystems, “J2SE 5.0”, http://java.sun.com/j2se/1.5.0/ (2006).

Wolski, R., “Lecture notes”, http://www.cs.ucsb.edu /~rich/class/cs290I-grid/notes/ (2006).

Wolski, R., Spring, N., Hayes, J., “The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing”, Joural of Future Generation Computing Systems, Vol. 15, N. 5-6, pp. 757-768, Outubro, 1999.

Yeong, W., Howes, T., Kille, S., “Lightweight directory access protocol”, RFC 1777, Março, 1995.

MAG, um *middleware* de grade baseado em agentes: estado atual e perspectivas futuras

**Francisco José da Silva e Silva, Rafael Fernandes Lopes,
Bysmarck Barros de Sousa, Antônio Eduardo Viana, Stanley Araújo de Sousa**

[1]Universidade Federal do Maranhão (UFMA)
Av. dos Portugueses s/n, Campus Universitário do Bacanga
São Luís – MA – Brasil

{fssilva,rafaelf}@deinf.ufma.br, {bysmarck,eduardo,stanley}@lsd.ufma.br

Abstract. *MAG (*Mobile Agents for Grid Computing Environments*) explores the mobile agent paradigm as a way to overcome the construction challenges of computational grids. MAG executes Grid applications by dynamically loading the application code into mobile agents. The MAG agent can be dynamically reallocated to Grid nodes through a transparent migration mechanism, as a way to provide load balancing and support for non-dedicated nodes. MAG includes mechanisms for fault tolerance, pervasive grid, and data grid. We make extensive use of the agent paradigm to design and implement MAG components, forming a multi agent infrastructure for computational Grids.*

Resumo. *O MAG (*Mobile Agents for Grid Computing Environments*) explora a tecnologia de agentes móveis como uma forma de superar os desafios de construção de grades de computadores. O MAG executa aplicações carregando dinamicamente seus códigos nos agentes móveis. O agente do MAG pode ser realocado dinamicamente entre nós da grade através de um mecanismo de migração transparente chamado MAG/Brakes, como uma forma de prover suporte a nós não dedicados. O MAG inclui mecanismos de tolerância a falhas de aplicações, de grade pervasiva e grade de dados. O paradigma de agentes foi extensivamente utilizado para projetar e implementar os componentes do MAG, formando uma infraestrutura multiagente para grades computacionais.*

1. Introdução

A computação em grade permite a integração e o compartilhamento de computadores e recursos computacionais, como software, dados e periféricos, em redes corporativas e entre estas redes, estimulando a cooperação entre usuários e organizações, criando ambientes dinâmicos e multi-institucionais, fornecendo e utilizando recursos de forma a alcançar objetivos comuns e individuais [Foster et al. 2001]. Através da infraestrutura da grade é possível executar um grande número de aplicações como: aplicações de supercomputação distribuída, alto rendimento, aplicações sob-demanda, aplicações que fazem uso intensivo de dados e aplicações colaborativas

Entretanto, a construção de um *middleware* de grade é uma tarefa complexa. Os desenvolvedores deste tipo de *middleware* devem superar muitos desafios de projeto e implementação, tais como: gerenciamento e alocação eficiente de recursos distribuídos;

escalonamento dinâmico de tarefas de acordo com a disponibilidade de recursos; mobilidade transparente de código, como uma forma de promover balanceamento de carga e suporte a nós não dedicados; mecanismos de tolerância a falhas, uma vez que os nós da grade não constituem um ambiente controlado; suporte a alta escalabilidade e grande heterogeneidade de componentes de hardware e software; e mecanismos eficientes para comunicação colaborativa entre os nós da grade.

A tecnologia de agentes móveis exibe grande adequação para o desenvolvimento de infraestruturas de grade, por causa de algumas de suas características, como:

- *Cooperação:* agentes têm a habilidade de interagir e cooperar com outros agentes. Isto pode ser explorado para o desenvolvimento de mecanismos de comunicação complexos entre os nós da grade;
- *Autonomia:* agentes são entidades autônomas, significando que suas execuções ocorrem sem nenhuma, ou com muito pouca intervenção dos clientes que os iniciaram. Este é um modelo adequado para a submissão e a execução de aplicações na grade;
- *Heterogeneidade:* muitas plataformas de agentes móveis podem ser executadas em ambientes heterogêneos, uma característica importante para um melhor uso dos recursos computacionais entre ambientes multi-institucionais;
- *Reatividade:* agentes podem reagir a eventos externos, como variações na disponibilidade de recursos;
- *Mobilidade:* agentes móveis podem migrar de um nó para outro, movendo parte da computação sendo executada de forma a prover balanceamento de carga entre os nós da grade e suporte a nós não dedicados.

O MAG[1] (*Mobile Agents Technology for Grid Computing Environments*) [Lopes et al. 2005, Lopes 2006] é um projeto de *middleware* de grade desenvolvido por alunos e professores do Departamento de Informática da Universidade Federal do Maranhão (DEINF/UFMA), cujo principal objetivo é disponibilizar uma infraestrutura de software livre baseada na tecnologia de agentes de software que permita a resolução de problemas computacionalmente intensivos em grades de computadores.

O MAG utiliza o *middleware* Integrade [Goldchleger et al. 2004] como fundação para sua implementação, evitando assim, a duplicação de esforços no desenvolvimento de uma série de componentes. O MAG adiciona ao Integrade um novo mecanismo de execução de aplicações, através do uso da tecnologia de agentes móveis, para permitir a execução de aplicações Java, não suportadas nativamente pelo Integrade. Além desta funcionalidade, o MAG possui, em seu estado atual de implementação, mecanismos para prover tolerância a falhas de suas aplicações, migração transparente (de forma a prover suporte a nós não dedicados), além de permitir o acesso de clientes móveis e nômades à grade (através de dispositivos de computação portátil e da web).

Este artigo está organizado como a seguir. A seção 2 descreve a arquitetura do MAG. A seção 3 descreve a infra-estrutura para o acesso de clientes móveis e nômades à grade. A seção 4 apresenta a arquitetura dos serviços de dados e metadados desenvolvidos no contexto do MAG. Por fim, na seção 5 são apresentadas as conclusões obtidas a partir do trabalho realizado e descritos seus próximos passos.

[1]Disponí vel em: http://www.lsd.ufma.br/mag

2. Arquitetura Geral do MAG

A arquitetura do MAG está organizada na forma de uma pilha de camadas, conforme pode ser visto na Figura 1. Atualmente no MAG é possível executar duas classes distintas de aplicações: regulares e paramétricas. As aplicações regulares são aplicações seqüenciais compostas por um único binário que executa em uma só máquina. Já as aplicações paramétricas são aquelas que executam múltiplas cópias do mesmo binário em máquinas distintas e com entradas diferentes. Nesta classe de aplicações (também chamada de BoT ou *Bag-of-Tasks*) as tarefas são divididas em sub-tarefas menores que executam de forma independente, sem haver comunicação entre elas. Várias aplicações enquadram-se nesta categoria, como as de processamento de imagens, simulação e mineração de dados. Aplicações escritas na linguagem Java são executadas pelo MAG, enquanto que aplicações nativas do sistema operacional são executadas pelo Integrade.

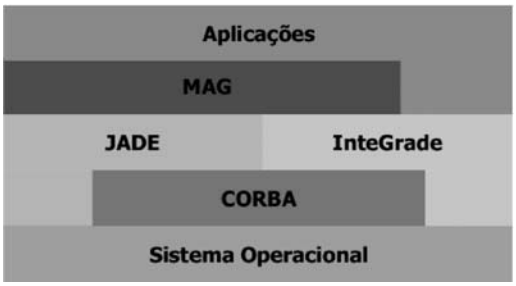

Figura 1. Arquitetura em camadas do *middleware* MAG

A camada JADE (*Java Agent Development Framework*) representa o arcabouço utilizado para a construção de sistemas multiagentes adotado no MAG. Este arcabouço provê ao MAG várias funcionalidades como facilidades de comunicação, controle do ciclo de vida e monitoração da execução dos agentes móveis. O JADE é uma plataforma portável (pois foi desenvolvida com tecnologia Java) e aderente à especificação FIPA[2].

As camadas superiores utilizam a tecnologia de objetos distribuídos CORBA para promover a comunicação entre seus componentes. Além disso, esta tecnologia fornece diversos serviços que podem ser utilizados pela infraestrutura do MAG.

2.1. Integrade

Integrade é um *middleware* de grade cujo desenvolvimento envolve um conjunto de universidades brasileiras (USP, Puc-Rio, UFMS, UFG e UFMA), coordenadas pelo Instituto de Matemática e Estatística da Universidade de São Paulo (IME/USP). O Integrade é estruturado em aglomerados organizados de uma forma hierárquica. Um aglomerado é definido como uma unidade autônoma dentro da grade, uma vez que o mesmo contém todos os componentes necessários para funcionar independentemente. Os componentes centrais da arquitetura do Integrade são:

- *Application Submission and Control Tool* (ASCT): interface gráfica que permite aos usuários da grade, submeter aplicações para serem executadas na grade;

[2] *Foundation for Intelligent Physical Agents* – organização sem fi ns lucrativos que objetiva a produção de padrões de interoperabilidade entre agentes de software heterogêneos. Disponí vel em http://www.fi pa.org/

- *Application Repository* (AR): armazena as aplicações a serem executadas na grade;
- *Local Resource Manager* (LRM): componente que executa em cada nó do aglomerado, coletando informações sobre o estado dos recursos dos nós da grade como memória CPU, disco e utilização da rede. Ele é também responsável por instanciar aplicações escalonadas ao nó;
- *Global Resource Manager* (GRM): gerencia os recursos do aglomerado, recebendo notificações de utilização de recursos a partir dos LRMs e executando o escalonador que aloca tarefas aos nós, de acordo com a disponibilidade de seus recursos.

Para permitir a execução de aplicações Java através da tecnologia de agentes de software, o MAG adiciona dois novos componentes à arquitetura original do Integrade: o *MagAgent* e o *AgentHandler*.

O *MagAgent* é um agente móvel responsável por executar aplicações na grade. Em resposta a cada solicitação de execução de aplicação, um novo *MagAgent* é criado. Ele atua como um *wrapper* da aplicação, sendo responsável por: (a) baixar o *bytecode* das aplicações a partir do repositório de aplicações; (b) dinamicamente instanciar e executar a aplicação utilizando mecanismos de reflexão computacional fornecidos pela plataforma Java; (c) capturar qualquer exceção não capturada pela aplicação; e (d) capturar e salvar periodicamente o estado de execução da aplicação (*checkpoint*), de forma a permitir sua posterior recuperação caso uma falha ocorra.

Um *AgentHandler* executa em cada nó da grade que fornece recursos para a execução de aplicações. Ele mantém o *container* de agentes executando no nó e também é responsável por criar *MagAgents* ao receber requisições de execução. Este componente torna o paradigma de agentes de software transparente ao Integrade.

Além de permitir a execução de aplicações Java através da tecnologia de agentes de software, o MAG disponibiliza um mecanismo para migração transparente de aplicações e um mecanismo de tolerância a falhas de aplicações executadas pelo MAG.

O impacto dos componentes de gerenciamento de recursos local e global (LRM e GRM) e do *AgentHandler*, utilizando como métrica o uso relativo de CPU, foi medido através de experimentos realizados sobre estes componentes. Os resultados obtidos comprovam a baixa sobrecarga de CPU causada pela presença dos componentes do MAG em nós da grade. Como resultado deste experimento, foram obtidas as médias de 0.086%, 0.040% e 0.012% de consumo de CPU causado pelos componentes *AgentHandler*, *LRM* e *GRM*, respectivamente. Maiores detalhes a respeito deste e de outros experimentos realizados na plataforma podem ser encontrados em [Lopes et al. 2005] e [Lopes 2006].

2.2. Mecanismo de migração transparente de aplicações

O MAG utiliza o poder computacional ocioso de máquinas existentes em redes computacionais para executar aplicações. Isto implica dizer que as máquinas que compõem a grade não são, necessariamente, dedicadas à grade, podendo pertencer a outros usuários. Dessa forma, é fundamental que o MAG disponibilize um mecanismo através do qual os usuários possam solicitar o uso exclusivo de seus recursos. Assim, quando um usuário solicita o uso exclusivo de sua máquina, todas as computações que

nela executam devem ser migradas para um outro nó da grade de forma transparente, sem perda de nenhuma computação já realizada.

Uma vez que o MAG executa aplicações Java utilizando agentes móveis, passa a ser necessário que exista um mecanismo que permita a captura e a recuperação do estado de execução de *threads* Java. Entretanto, a plataforma Java não provê mecanismos suficientes para capturar o estado de execução de computações. Seu mecanismo de serialização somente permite a salva do código e do valor dos atributos de objetos. Além disso, as classes Java não podem acessar informações nativas e internas da Máquina Virtual Java (por exemplo, o contador de instrução e a pilha de chamadas), necessárias para a captura do estado completo de execução de *threads* Java.

Durante os últimos anos, algumas técnicas foram desenvolvidas no intuito de permitir a captura do estado de execução de *threads* Java. Estas técnicas podem ser classificadas em quatro abordagens básicas [Illmann et al. 2000]: modificação da máquina virtual, instrumentação do código-fonte das aplicações, instrumentação do *bytecode* das aplicações e a modificação da *Java Platform Debugger Architecture*[3]. Cada uma destas técnicas traz vantagens e desvantagens, sendo que a abordagem de instrumentação de *bytecode* mostrou-se a mais adequada para uso em ambientes de grades computacionais.

O mecanismo de captura e recuperação do estado de execução de *threads* Java do MAG é baseado em uma versão modificada do arcabouço Brakes [Truyen et al. 2000], desenvolvido no Katholieke Universiteit Leuven, Bélgica, pelo grupo de pesquisa em sistemas distribuídos e redes de computadores (DistriNet). Esta versão modificada do Brakes foi chamada MAG/Brakes [Lopes and Silva 2005a, Lopes and Silva 2005b], e fornece à versão original do Brakes algumas melhorias, como a redução da sobrecarga imposta pela inserção de *bytecode* do código da aplicação, a migração iniciada por uma entidade externa (i.e. o agente pode solicitar a captura do estado de execução da aplicação), além de tornar mais flexível o uso do MAG/Brakes por seus usuários. Um comparativo entre o arcabouço MAG/Brakes e outros arcabouços de migração transparente de código de *threads* Java, bem como testes de avaliações de desempenho realizadas sobre o mesmo podem ser encontradas em [Lopes and Silva 2005a], [Lopes and Silva 2005b] e [Lopes 2006].

2.3. Mecanismo de tolerância a falhas de aplicações

Tolerância a falhas é uma característica essencial em ambientes de grade. Uma vez que a grade atua como um sistema massivamente paralelo, a perda de tempo computacional deve ser evitada. De fato, a probabilidade de ocorrerem erros é grande, uma vez que, na grade, muitas aplicações executarão longas tarefas que podem requerer vários dias de computação.

A infraestrutura de tolerância a falhas do MAG [Lopes and Silva 2006] contempla atualmente falhas do tipo *fail-stop* (i.e. colapso de nós e processos). Para tanto, uma sociedade de agentes foi desenvolvida para prover tolerância a falhas às aplicações que executam na grade, sendo responsáveis por detectar e tratar estas falhas de maneira autônoma. Futuramente pretende-se dar suporte a outras categorias de falhas, como por exemplo, falhas na rede.

[3]A *Java Platform Debugger Architecture* (JPDA) é parte da especificação padrão da máquina virtual. Através da JPDA as informações de execução das aplicações podem ser acessadas em modo de depuração. Isto pode ser explorado para prover migração transparente.

A recuperação de aplicações no MAG é baseada na abordagem de recuperação por retorno (i.e. *checkpointing*), onde o estado de execução das aplicações é salvo periodicamente em um armazém estável de dados. Assim, se uma falha vier a ocorrer, as computações podem ser recuperadas a partir de seu último salvo no armazém estável. Além disso, está sendo atualmente implementado no MAG o suporte a uma outra importante técnica de tratamento de falhas de aplicações: a replicação de processos. Nesta técnica, várias cópias da mesma aplicação (e com o mesmo conjunto de dados de entrada) são submetidas simultaneamente para execução na grade. Assim, a falha de uma réplica da aplicação não acarreta perda de tempo computacional já realizado pela aplicação, uma vez que as outras réplicas da mesma aplicação continuarão a executar sem problemas.

Com a implementação da técnica de replicação de processos será possível fornecer ao usuário da grade um esquema flexível para o tratamento de falhas de aplicações. Esta flexibilidade deriva da combinação das duas técnicas fornecidas pelo MAG, resultando em quatro diferentes abordagens para o tratamento de falhas: reinício da aplicação (sem *checkpointing* ou replicação), *checkpointing* sem replicação, replicação sem *checkpointing* e replicação com *checkpointing*. Em um primeiro momento, o uso de cada uma destas técnicas deverá ser configurado pelo usuário através da interface de submissão de aplicações, enquanto que posteriormente pretende-se tornar a grade adaptável, fazendo com que cada *MagAgent* decida qual a melhor estratégia a utilizar para sua aplicação. Esta decisão pode ser amparada por fatores relativos à aplicação e ao ambiente da grade.

O mecanismo de tolerância a falhas do MAG é composto por três agentes de software: o *StableStorage*, o *AgentRecover* e o *ExecutionManagementAgent* (EMA). O *StableStorage* representa o armazém estável que armazena o *checkpoint* de todas as aplicações em execução na grade. Sua implementação atual é centralizada e é executada no nó gerenciador do aglomerado. O *AgentRecover* é responsável por iniciar o processo de recuperação de aplicações, caso uma falha de colapso de nó venha a ocorrer. Este componente é criado sob demanda pelo GRM (*Global Resource Manager*, veja a Seção 2.1), que é responsável por detectar falhas de colapso de nós. O EMA fornece três principais funcionalidades para o mecanismo de tolerância a falhas do MAG: (1) ele fornece informações a respeito das execuções de aplicação (como os argumentos de linha de comando passados à aplicação e o seu estado atual de execução), utilizadas pelo processo de recuperação para restaurar a aplicação como os mesmos parâmetros que tinha antes da falha; (2) ele mapeia cada execução de aplicação com o nó em que executa, podendo assim descobrir que aplicações executavam em um dado nó que falhou; e (3) os dados que o mesmo armazena são utilizados na geração de estatísticas que amparam decisões relativas às técnicas de tratamento de falhas empregadas.

O custo computacional introduzido pelo mecanismo de tolerância a falhas no ambiente de execução do MAG foi avaliado por meio de testes realizados com uma aplicação denominada GLCM (*Gray Level Co-occurrence Matrix*) [Haralick et al. 1973], um conhecido método de análise de texturas. Os resultados obtidos através dos experimentos mostram que houve um aumento em torno de 8.93% no tempo total de execução da aplicação GLCM em relação à sua execução normal. Este aumento deve-se à captura e salva periódica do *checkpoint* da mesma, dos quais 2.37% devem-se ao mecanismo de captura do estado de execução (o arcabouço MAG/Brakes foi utilizado para este fim), enquanto que os outros 6.56% devem-se à transferência de dados entre o *MagAgent* e

o *StableStorage*. Este custo pode ser considerado aceitável para várias aplicações, dados os benefícios providos pelo mecanismo de tolerância a falhas de aplicações. Uma análise aprofundada dos experimentos e seus resultados, bem como comparações com outros importantes trabalhos relacionados a tolerância a falhas de aplicações em grades de computadores, podem ser encontrados em [Lopes and Silva 2006] e [Lopes 2006].

3. Grade Pervasiva

Uma das promessas da computação em grade é fornecer acesso a recursos computacionais de alto desempenho de forma simples e transparente, analogamente ao que ocorre com o acesso à rede elétrica (*power grid*). Isto significa que usuários de grades de computadores devem poder utilizar os serviços providos pela mesma a partir de casa, do trabalho, e até mesmo durante sua locomoção. Entretanto, para que esta promessa se torne realidade, é necessário que a infraestrutura da grade forneça um mecanismo através do qual os usuários possam ter acesso aos serviços da mesma, independentemente de localização física. Será utilizado o termo *grade pervasiva* para denotar este mecanismo.

O desenvolvimento de uma infraestrutura de grade pervasiva envolve uma série de desafios introduzidos pela mobilidade de seus usuários. Esta infraestrutura pode, no entanto, ser dividida em dois diferentes mecanismos, de acordo com o tipo de mobilidade que deverá ser suportada pela infraestrutura da grade: (1) usuários nômades e (2) usuários móveis. A mobilidade nômade é caracterizada pelo deslocamento de usuários / dispositivos através de limites institucionais [Kleinrock 1997, McKnight et al. 2004], permanecendo desconectados enquanto se locomovem, e.g. um usuário que esteja utilizando um acesso *dial-up* em uma localidade deve desconectar-se antes de mudar para outra localidade, reconectando-se novamente no destino. Ao contrário do que ocorre com usuários nômades, usuários móveis não devem ter suas conexões interrompidas durante sua locomoção [McKnight et al. 2004], sendo estas conexões mantidas através de tecnologias de comunicação sem fio, e.g. um usuário acessando a Internet através de um *Smart-Phone* utilizando tecnologia GPRS, representa um usuário móvel.

O mecanismo de grade pervasiva do MAG (chamado *PervMAG*) propõe a extensão da arquitetura básica do *middleware* MAG de forma a permitir a utilização dos serviços da grade por parte de usuários móveis e nômades. Para dar suporte a estes usuários foram criados dois mecanismos que representam as duas frentes deste projeto:

- *Mecanismo de suporte a usuários móveis*: através deste mecanismo, usuários de dispositivos de computação móvel podem interagir com o MAG para utilizar seus serviços, podendo a grade funcionar como uma extensão do poder computacional de seus dispositivos. Na versão atual, a comunicação entre os dispositivos e a infraestrutura da grade se dá através da tecnologia de comunicação sem fio IEEE 802.11x e utiliza-se dispositivos PalmOS;
- *Mecanismo de suporte a usuários nômades*: este mecanismo prevê que os usuários possam interagir com a infraestrutura da grade através da Internet, utilizando para isto uma interface web. Assim, sempre que o usuário desejar se conectar à grade, poderá fazê-lo através de qualquer computador que tenha acesso à web.

Através destes mecanismos é possível interagir com o MAG para solicitar serviços. Entre estes serviços encontram-se os de submissão, acompanhamento e coleta dos resultados de execuções de aplicações. As arquiteturas dos mecanismos de acesso à grade por usuários móveis e nômades serão detalhadas nas seções a seguir.

3.1. Suporte a usuários móveis

Dispositivos de computação móvel normalmente apresentam restrições de hardware que impedem seus usuários de realizar tarefas que exijam um alto poder de processamento ou uma grande capacidade de armazenamento de dados, restringindo as classes de aplicações que podem ser executadas pelos mesmos. Estas limitações tornam os usuários de dispositivos móveis um emergente segmento de clientes de grades, uma vez que estas podem funcionar como uma extensão dos recursos computacionais destes dispositivos, adicionando-lhes novas capacidades e funcionalidades [González-Castaño et al. 2002].

O integração desses dois novos paradigmas apresenta desafios inerentes à comunicação sem fio e às características dos dispositivos de computação móvel, como baixa largura de banda, conectividade intermitente, heterogeneidade de dispositivos e carência de recursos computacionais [Phan et al. 2002]. Para tratar estes desafios e torná-los transparentes à infraestrutura da grade foi adotado o modelo *cliente / agente / servidor* proposto por Pitoura et al. [Pitoura and Samaras 1998]. Nesta arquitetura, o *agente* exerce o papel de um *proxy* responsável por realizar a interação entre os *clientes móveis* e o *servidor* (a grade), tornando os desafios da computação móvel transparentes ao *servidor*.

A arquitetura do mecanismo de suporte à usuários móveis do MAG é composta pelos seguintes componentes:

- **Mobile Application Submission and Control Tool (MASCT)**: interface gráfica que permite aos usuários de dispositivos móveis requisitar os seguintes serviços da grade: submeter aplicações, acompanhar o andamento da execução de aplicações, receber notificações e visualizar os resultados das computações finalizadas. Durante períodos de desconexão ou baixa conectividade, o MASCT armazena em *log* as requisições de execução efetuadas por seus usuários, sendo enviadas à grade assim que a conexão é reestabelecida;
- **Mobile Proxy Agent**: representa o *agente* da arquitetura *cliente / agente / servidor*. O *MobileProxyAgent* é responsável por tornar os dispositivos móveis e os problemas relacionados à comunicação sem fio transparentes aos componentes da grade. O *MobileProxyAgent* recebe as requisições de cada MASCT executando nos dispositivos móveis e as encaminha para execução na grade. Notificações de aplicações finalizadas são encaminhadas ao *MobileProxyAgent* para serem, então, enviadas aos dispositivos móveis que as requisitaram. Caso o dispositivo esteja desconectado esta notificação será armazenada em um banco de dados (mantido pelo componente *DataBaseAgent*) até que a conexão de rede do dispositivo seja novamente estabelecida;
- **Mobile Local Resource Manager (MLRM)**: monitora os recursos do dispositivo móvel (e.g. energia da bateria e memória disponível) e informa ao *MobileProxyAgent* o estado atual dos mesmos. Isto tem como objetivo tornar o *MobileProxyAgent* ciente do estado de conectividade e de disponibilidade de recursos de cada dispositivo para permitir a utilização de adaptação dinâmica de conteúdo nos arquivos de saída das aplicações executadas pela grade. Além disso, o *MobileProxyAgent* é responsável por informar ao MASCT a respeito das condições de rede;
- **Mobile DataBase Agent (MDBA)**: mantém uma base de dados que associa cada execução de aplicação com o dispositivo que a requisitou. O DBA é também res-

ponsável por armazenar características dos dispositivos portáteis. São armazenadas características estáticas, como: o identificador do dispositivo, a profundidade de cores, a dimensão da tela e o poder de processamento; bem como as seguintes características dinâmicas: o endereço de rede do dispositivo, o estado atual da conexão (conectado ou desconectado) e a quantidade de energia da bateria.

3.2. Suporte a usuários nômades

Muitas vezes o acesso à redes sem fio não está disponível, o que torna inviável a utilização da arquitetura de suporte a usuários móveis do MAG. Torna-se então necessário que existam outras maneiras de interagir com a infraestrutura da grade, de forma a garantir acesso ubíquo à mesma. Segundo Kindberg et al. [Kindberg and Barton 2001], a web é a infraestrutura básica para fornecer suporte à usuários nômades, uma vez que pode ser acessada a partir de um enorme (e crescente) número de lugares. A arquitetura de suporte a usuários nômades do MAG é centrada na utilização da tecnologia de serviços web, e é composta pelos seguintes componentes:

- **ASCTWeb ou *Application Submission and Control Tool for Web***: ferramenta visual executada em navegadores que possibilita aos usuários registrarem as aplicações para a execução na grade, acompanhar o estado da execução das aplicações e obter os resultados das computações finalizadas. É uma interface web desenvolvida com a tecnologia JSP (*Java Server Pages*), semelhante à interface do ASCT disponibilizado pelo *middleware* Integrade. É através desta interface que usuários nômades interagem com a grade;
- **MagWS ou *MAG Web Service***: exerce o papel de um *proxy* da grade, cuja interface pública disponibiliza os serviços oferecidos pelo *middleware* MAG. Fornece a interface de comunicação entre a web e a grade, recebendo requisições do AsctWeb e repassando-as à grade (e vice-versa).

4. Grade de Dados

Atualmente um grande volume de informações é gerado em vários domínios de aplicação, originando grandes coleções de dados. Estas coleções encontram-se geograficamente dispersas, assim como os seus usuários. A combinação dessas coleções, a distribuição geográfica de recursos e usuários e também a computação intensiva sobre dados, demandam mecanismos sofisticados de acesso e compartilhamento. Estes mecanismos devem ser capazes de tratar questões como: falta de interoperabilidade entre tecnologias de sistemas de banco de dados (XML, relacional, arquivos), segurança, replicação, transferência rápida e confiável, publicação e descoberta dos dados.

Para discutir essas questões, a computação em grade especializou-se, objetivando fornecer uma infraestrutura para manipular grandes volumes de dados [Foster et al. 2001]. Esta especialização é chamada de *grade de dados* (*datagrid*). Chervernak et. al descreve uma arquitetura geral, enfatizando dois serviços básicos [Chervenak et al. 2001]: (a) *serviço de metadados* e (b) *serviço de dados*. O serviço de metadados fornece os mecanismos necessários para a publicação de dados, de forma que usuários possam descobrí-los através de seus metadados; já o serviço de dados engloba mecanismos de acesso, gerenciamento e transferência de dados, objetivando fornecer transparência quanto às características distribuídas do ambiente.

A arquitetura do serviço de metadados do MAG, chamado MagCat [Sousa et al. 2006], é composta pelos seguintes agentes:

- *CatalogManager*: realiza operações sobre o repositório de metadados (adição, remoção, atualização e consulta);
- *SchemaManager*: responsável pela extensibilidade do esquema de metadados para diferentes domínios de aplicação;
- *SearchAgent*: executa consultas em repositórios distribuídos de metadados;
- *RequestMonitor*: monitora as operações executadas por usuários do MagCat;
- *ReplicationManager*: responsável pela replicação e sincronização de réplicas dos metadados, utilizando as informações coletadas pelo *RequestMonitor* para decidir quando e onde criar réplicas;
- *SecurityAgent*: é responsável pela autenticação e autorização usando mecanismos como o *Community Authorization Service* (CAS) [Pearlman et al. 2002].

A implementação atual do serviço de metadados consiste nos agentes *CatalogManager* e *SchemaManager*, os quais provêem as funcionalidades de publicação e descoberta de metadados e extensão de esquema de metadados, respectivamente. Estes agentes foram projetados de forma a permitir um alto grau de independência do sistema de armazenamento de metadados utilizado. Atualmente é fornecido suporte ao serviço de diretórios LDAP e ao sistema de banco de dados relacional MySQL. Outras importantes características como distribuição, replicação, sincronização de réplicas e segurança são alcançadas através do sistema de armazenamento de metadados.

O serviço de dados do MAG encontra-se em uma versão preliminar. O agente *DataManager* é responsável pelo fornecimento deste serviço, possibilitando o armazenamento e a recuperação dos dados em um repositório de arquivos. Este agente executa no nó gerenciador de cada um dos aglomerados da grade, permitindo o acesso a bases distribuídas. Uma nova versão deste serviço está atualmente sendo projetada de forma a permitir o uso de técnicas de replicação dos dados armazenados nos aglomerados.

O modelo de dados definido para o MagCat enfatiza a extensibilidade e agrupamento de objetos relacionados. A unidade conceitual para representação da informação é um *arquivo*, que pode ser agrupado em *coleções*. Um especialista de um dado domínio pode agrupar os arquivos em *visões* de acordo com suas necessidades. Cada arquivo, coleção e visão tem um identificador único. Cada conceito no modelo de dados tem um conjunto de atributos básicos, mas o especialista do domínio pode adicionar atributos específicos do domínio de sua aplicação.

5. Conclusões e Trabalhos Futuros

Questões como gerenciamento e alocação eficiente de recursos distribuídos, escalonamento dinâmico de tarefas de acordo com a disponibilidade de recursos, tolerância a falhas de nós e aplicações individuais, mobilidade de código, gerenciamento eficiente da execução de aplicações, comunicação entre computações cooperantes executando em nós distintos e diversos aspectos relacionados à segurança, constituem importantes desafios para a construção de um *middleware* de grade.

Considerando algumas características relativas à tecnologia de agentes móveis, como cooperação, autonomia, reatividade e mobilidade, pode-se concluir que este paradigma simplifica potencialmente o desenvolvimento da infraestrutura da grade. Como

prova de conceito, este artigo descreveu o MAG, um *middleware* de grade que explora o paradigma de agentes como uma forma de superar diversos desafios relativos ao desenvolvimento de um *middleware* de grade.

O MAG executa aplicações na grade carregando dinamicamente o código da aplicação em um agente móvel. O agente do MAG pode ser dinamicamente realocado entre nós da grade através de um mecanismo de migração transparente de aplicações denominado MAG/Brakes, permitindo que nós não dedicados sejam utilizados como parte da infraestrutura da grade. O MAG também fornece um mecanismo de tolerância a falhas de aplicações. Este mecanismo é essencial em ambientes de grades, dado que evita a perda de tempo computacional realizado durante longos períodos. Atualmente, o mecanismo de tolerância a falhas do MAG trata somente falhas de colapso de nós, e é baseado na abordagem de *checkpoint*, em que o estado de execução da aplicação é periodicamente salvo em um armazém estável. Assim, se uma falha ocorrer, as computações podem ser recuperadas a partir de seu último *checkpoint*. Finalmente, foi descrita a infraestrutura de grade pervasiva implementada no contexto do *middleware* MAG. Esta infraestrutura permite que os usuários móveis e nômades possam ter acesso à grade através de uma rede sem fio e através da web, respectivamente. Estes mecanismos são um primeiro passo em rumo à promessa de acesso ubíquo da grade, isto é, poder utilizar seu poder computacional a qualquer hora e em qualquer lugar.

Algumas direções futuras do projeto MAG são: investigar abordagens adaptativas para o mecanismo de tolerância a falhas, de forma a prover mecanismos dinâmicos para a detecção e a recuperação de falhas; o desenvolvimento de mecanismos de proteção contra aplicações hostis e defeituosas; o desenvolvimento de mecanismos de balanceamento de carga para melhor aproveitar o mecanismo de migração forte do MAG; prover suporte à aplicações paralelas do tipo MPI, bem como migração e tolerância a falhas de processos MPI; personalização da plataforma JADE de forma a remover serviços que não sejam utilizados pelos agentes do MAG, economizando recursos de memória e processamento.

6. Agradecimentos

O MAG faz parte dos projetos FlexiGrid e Integrade2, financiados pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), processos 50.6689/2004-2 e 55.0094/2005-9, respectivamente. Agradecemos ainda a Estevão Freitas e Gilberto Cunha pelo fundamental apoio empregado no desenvolvimento deste projeto.

Referências

Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., and Tuecke, S. (2001). The Data Grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications*, 23:187–200.

Foster, I., Kesselman, C., and Tueke, S. (2001). The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputing Applications*.

Goldchleger, A., Kon, F., Goldman, A., Finger, M., and Bezerra, G. C. (2004). Integrade: Object-oriented grid middleware leveraging idle computing power of desktop machines. *Concurrency and Computation: Practice & Experience. Vol. 16, pp. 449-459*.

González-Castaño, F. J., Vales-Alonso, J., Livny, M., Costa-Montenegro, E., and Anido-Rifón, L. (2002). Condor grid computing from mobile handheld devices. *SIGMOBILE Mobile Computing Communications*.

Haralick, R. M., Shanmugam, K., , and Dinstein, I. (1973). Textural features for image classification. *IEEE Trans. Syst., Man, Cybern., SMC-3*, pages 610–621.

Illmann, T., Kargl, F., Krueger, T., and Weber, M. (2000). Migration in Java: problems, classifications and solutions. In *MAMA'2000*, Wollongong, Australia.

Kindberg, T. and Barton, J. (2001). A Web-based nomadic computing system. *Computer Networks, Elsevier*, Vol. 35(No. 4):443–456.

Kleinrock, L. (1997). Nomadicity: anytime, anywhere in a disconnected world. *Mobile networks and applications*, 1(4):351–357.

Lopes, R. F. (2006). MAG: uma grade computacional baseada em agentes móveis. Master's thesis, Universidade Federal do Maranhão, Sã Luís, MA, Brasil. In Portuguese.

Lopes, R. F. and Silva, F. J. d. S. (2005a). Migration Transparency in a Mobile Agent Based Computational Grid. In *Proceedings of the 1st WSEAS International Symposium on GRID COMPUTING*, pages 31–36, Corfu, Greece.

Lopes, R. F. and Silva, F. J. d. S. (2005b). Strong Migration in a Grid based on Mobile Agents. *WSEAS Transactions On Systems*, Volume 4(Issue 10):1687–1694.

Lopes, R. F. and Silva, F. J. d. S. (2006). Fault Tolerance in a Mobile Agent Based Computational Grid. In *4th International Workshop on Agent Based Grid Computing. IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'06)*, Singapore. IEEE Computer Society Press.

Lopes, R. F., Silva, F. J. d. S., and Sousa, B. B. d. (2005). MAG: A Mobile Agent based Computational Grid Platform. In *In Proceedings of the 4th International Conference on Grid and Cooperative Computing (GCC 2005)*, Lecture Notes in Computer Science, Beijing. Springer-Verlag.

McKnight, L. W., Howison, J., and Bradner, S. (2004). Wireless grids: Distributed resource sharing by mobile, nomadic, and fixed devices. *IEEE Internet Computing*, pages 24–31.

Pearlman, L., Welch, V., Foster, I., Kesselman, C., and Tuecke, S. (2002). A community authorization service for group collaboration.

Phan, T., Huang, L., and Dulan, C. (2002). Challenge: Integrating mobile wireless devices into the computational grid.

Pitoura, E. and Samaras, G. (1998). *Data Management for Mobile Computing*. Kluwer Academic Publisher.

Sousa, B. B. d., Silva, F. J. d. S., Teixeira, M. M., and Filho, G. C. (2006). MagCat: An Agent-based Metadata Service for Data Grids. In *4th International Workshop on Agent Based Grid Computing. IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'06)*, Singapore. IEEE Computer Society Press.

Truyen, E., Robben, B., Vanhaute, B., Coninx, T., Joosen, W., and Verbaeten, P. (2000). Portable support for transparent thread migration in java. In *ASA/MA*.

Sessão Técnnica 3 /
Technical Session 3

Sistema de Monitoramento/Gerência de Recursos e de Segurança para Grades Computacionais Baseadas no *Globus ToolKit*

Luis Rodrigo de Oliveira Gonçalves, Fábio de Oliveira Fagundes, Bruno Schulze, Letícia Braga Loureiro de Sá, Thais Cabral de Mello

[1]Laboratório Nacional de Computação Científica - LNCC
CEP: 25651-075 – Petrópolis – Rio de Janeiro – Brasil

{lrodrigo,fagundes,schulze,leticiasa,thaiscm}@lncc.br

Abstract. *This document presents a monitoring tool for the resources and security of Grids based on the Globus Toolkit. Different from other solutions, it avoids the installation of additional software, since all the collection process is carried out by scripts, which are sent using the communication channel provided by the Globus libraries. Scripts will also be used to provide the necessary mechanisms for the security adjustments.*

Resumo. *Este documento apresenta a descrição de uma solução para o monitoramento de recursos e da segurança de Grades Computacionais, implementadas através do uso do Globus Toolkit. Ao contrário de outras soluções de monitoramento/gerenciamento existentes, a solução proposta é baseada no modelo descrito em [GONÇALVES 2005], que permite a gerência, o monitoramento e o ajuste do ambiente sem a necessidade de instalação de novos aplicativos, visto que todo processo de coleta/ajuste será realizado por meio de scripts, enviados aos nós utilizando-se o canal de comunicação provido pelo próprio Globus.*

1. Introdução

Devido ao crescente uso de Grades Computacionais e ao fato da ferramenta *Globus ToolKit* ser considerada um padrão para a construção deste tipo de ambiente, há a necessidade do desenvolvimento de uma solução que permita o monitoramento e a gerência da segurança de ambientes com esta configuração.

Baseando-se nesta necessidade, está sendo construída uma solução que visa prover aos usuários informações atuais e históricas sobre o ambiente permitindo, desta forma, que durante o processo de submissão e execução de tarefas ocorra um melhor aproveitamento dos recursos disponíveis. Esta solução destina-se, também, a facilitar o processo de gerenciamento do *middleware* da Grade, bem como a manutenção da segurança do ambiente computacional.

A solução proposta é focada em um ambiente onde os nós possuem o sistema operacional *Linux* instalado e baseada no modelo de verificação citado na Seção 4, descrito em [GONÇALVES 2005] e no protótipo *SiMPaSeG* [FAGUNDES 2005], citado na Seção 5. Por estes motivos a implementação será totalmente baseada em ferramentas não proprietárias, conseqüentemente esta implementação pode ser facilmente migrada para outros ambientes *Unix-Like*, que não sejam baseados no sistema operacional *Linux*. A

interface com o usuário é baseada em um *website*, por meio do qual, dependendo do seu nível de acesso, o usuário poderá obter informações do ambiente e, caso possua nível de acesso administrativo, ele poderá realizar a gerência do ambiente provido pelo *Globus*, bem como a análise e o ajuste de segurança.

A ferramenta é segmentada em dois níveis: o *front-end*, que fornece uma interface direta do sistema com os usuários e o *back-end*, responsável pela coleta, análise de dados e ajuste do ambiente. Esta segmentação permite ao sistema possuir dois tipos de acesso: um à interface e outro ao sistema distribuído, deste modo todos os trabalhos submetidos pelos administradores serão executados sobre os direitos de uma única conta local, a qual deverá ter acesso privilegiado aos nós da Grade.

Por utilizar o *middleware* de Grade para realizar toda a coleta de informações a solução não exige a instalação prévia de aplicativos nas máquinas a serem monitoradas/gerenciadas, de forma similar ao *NetLogger* [TIERNEY 2002]. Esta abordagem permite reduzir a complexidade do processo de configuração e manutenção do ambiente.

Para facilitar seu entendimento, este documento é organizado da seguinte forma: Seção 2 apresenta as principais características de um ambiente de Grade Computacional; a Seção 3 apresenta o *Globus ToolKit*, bem como descreve seus principais componentes; na Seção 4 são apresentados os componentes básicos do modelo utilizado como base da implementação da solução proposta; a Seção 5 apresenta o primeiro protótipo de ferramenta para verificação de segurança implementado para Grades Computacionais utilizando-se o modelo descrito por [GONÇALVES 2005] e apresentado na Seção 4; na Seção 6 é descrita a solução proposta para o monitoramento e gerência dos recursos e da segurança de uma Grade baseada no *Globus Toolkit*; a Seção 7 apresenta alguns trabalhos relacionados à solução proposta; a Seção 8 apresenta os primeiros resultados obtidos e a Seção 9 conclui o artigo.

2. Grade Computacional

As Grades fornecem uma visão transparente do sistema, como se o conjunto de recursos que o formam (*hardware* e *software*) fosse um único e poderoso computador, sendo mais distribuídas, diversas e complexas que outras plataformas.

Seu objetivo inicial era disponibilizar poder computacional sob demanda, ou seja, seria necessário apenas a conexão de um computador à rede para que o mesmo já pudesse utilizar e disponibilizar recursos aos demais nós. As Grades Computacionais são plataformas heterogêneas, compartilhadas e complexas, utilizadas para execução de aplicações paralelas ou não, seu uso possibilita a alocação de uma enorme quantidade de recursos a uma aplicação com um custo muito menor que alternativas convencionais de processamento [CIRNE 2002].

Segundo [BUYYA 2005], a principal diferença entre as Grades Computacionais e os *Clusters* está relacionada ao modo como é feito o gerenciamento dos recursos, ou seja: se o compartilhamento de recursos é gerenciado por um único sistema global, sincronizado e centralizado, então é um *Cluster*. Mesmo que os recursos da Grade estejam geográficamente distribuídos e espalhados por vários domínios administrativos, os proprietários desses recursos possuem autonomia para fazer o gerenciamento local.

A Grade pode ser encarada como sendo uma evolução do *Cluster*, dado que os

recursos deste podem ser compartilhados pela Grade, da mesma forma ela é mais heterogênea, complexa e distribuída.

Para facilitar a classificação de um ambiente computacional como sendo uma Grade, foi criada a *Grid Checklist* [FOSTER 2002], que define as três características básicas que estes sistemas devem possuir:

1. Os recursos coordenados não são sujeitos a um controle centralizado, visto que uma Grade integra/coordena recursos e usuários residentes em diferentes domínios administrativos.
2. Deve utilizar padrões abertos, interfaces e protocolos de propósito geral, já que ela deve alcançar escalabilidade e interoperabilidade entre diferentes padrões de *hardware* e *software*, a fim de realizar funções fundamentais como: autenticação, autorização, descobrimento e controle de acesso aos recursos.
3. Deve prover o mínimo em qualidade de serviços, a Grade deve permitir que os recursos disponibilizados possam ser usados de forma coordenada, de modo a atingir diferentes qualidades de serviço, como por exemplo: tempo de resposta, vazão, disponibilidade, segurança e/ou a co-alocação de múltiplos recursos. O fornecimento de vários níveis de qualidade de serviço permite à Grade se adequar às necessidades mais complexas dos usuários.

3. *Globus Toolkit*

Vários sistemas para suporte à computação em Grade surgiram nos últimos anos, dentre aqueles que envolvem esforços acadêmicos o projeto de maior impacto foi o *Globus Toolkit* (GT), que teve sua versão 1.0 lançada no ano de 1998 e desde então vem se tornando o padrão de *facto* na construção de Grades Computacionais [FAGUNDES 2005].

Globus Toolkit consiste em um conjunto de serviços distintos, porém integrados, que facilitam a computação em Grade, seus serviços podem ser usados para submissão e controle de aplicações (*GRAM*); descoberta de recursos (*MDS*); movimentação de dados e segurança na Grade (*GSI*). Estes e outros componentes do Globus podem ser observados na Figura 1.

A segurança e a autenticação são dois aspectos que dificultam o uso de Grades Computacionais, pois neste tipo de ambiente existe a necessidade de autenticação dos usuários em diferentes domínios administrativos. Para resolver esse problema existe o serviço *GSI (Globus Security Infrastructure)* do *Globus*, que viabiliza o *login* único na Grade.

As Grades baseadas no *Globus* não possuem um escalonador que controla todo o sistema, todavia ele fornece o serviço *GRAM*, que funciona como uma interface única, utilizada para submeter, monitorar e controlar tarefas de forma independente do escalonador de recursos. Visando agilizar ainda mais este processo o *Globus* disponibiliza o *MDS* (*Metacomputing Directory Service*), um serviço de informação sobre os recursos do ambiente.

Para prover os mecanismos de transferência de dados entre os nós da Grade o *Globus* dispõe de um serviço chamado *GridFTP*, que implementa uma extensão do protocolo *FTP* tornando-o mais adequado para as necessidades da computação em Grade.

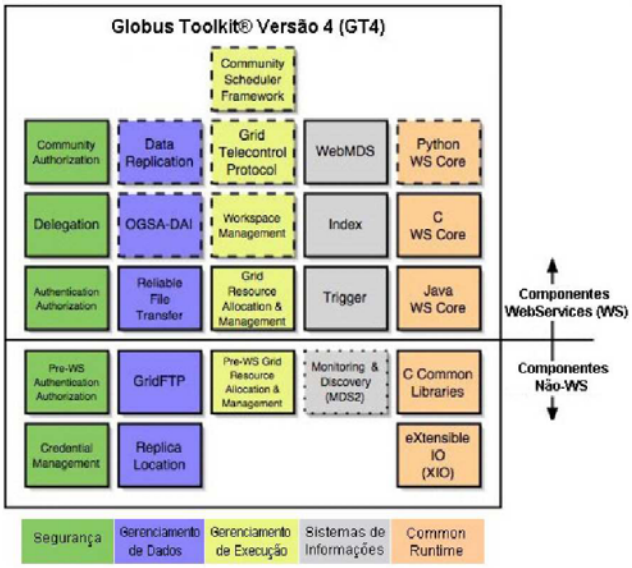

Figura 1. Visão geral dos componentes do *Globus Toolkit*

4. Modelo de Verificação

Em fevereiro de 2005 foi apresentado um modelo que descreve os componentes e as fases necessárias para se realizar a verificação automática do nível de aderência de um ambiente à norma nacional de segurança da informação [GONÇALVES 2005]. Este modelo propõe um sistema segmentado em duas partes: o *front-end* e o *back-end*. O *front-end* representa a parte do sistema através do qual o processo de verificação/ajuste do ambiente pode ser gerenciado, já o *back-end* é formado pelos componentes responsáveis pelo processo de coleta de informações a serem utilizadas na análise, e pelo processo de ajuste.

Em um ambiente de redes de computadores convencional o *back-end* é composto pelos *scripts* de coleta, pelo Agente Primário de Verificação (APV) e pelo Agente de Coleta de Dados (ACD). O objetivo destes agentes é promover a coleta *on-loco* de dados baseada na execução de *scripts*. Contudo, parte desta funcionalidade já é provida pelo *Globus ToolKit*, através do *GRAM* e do *GridFTP*.

Na implementação da ferramenta de análise e ajuste baseada no modelo, em um ambiente de Grade provido pelo *Globus*, é necessário apenas uma interface para o processo de submissão, a execução dos *scripts* de coleta e a implementação de um controle de coleta ao invés de uma implementação completa dos Agentes de Coleta e de Verificação.

No entanto, devido à forma de funcionamento do *GSI* alguns processos de coleta podem ser comprometidos, já que determinados *scripts* necessitam de privilégios de administrador. Todavia como pode ser observado nas seções 5 e 6, ambos os protótipos implementados adotaram soluções específicas para contornarem este problema.

5. Sistema de Monitoramento Passivo de Segurança de Grade Computacional (SiMPaSeG)

Este foi o primeiro protótipo implementado utilizando o modelo citado na Seção 4 e aplicado ao ambiente de Grade Computacional, o objetivo deste sistema é viabili-

zar a verificação da conformidade do ambiente há quinze controles da *NBR ISO/IEC 17799:2005* (3.1.1, 4.3.2, 4.3.3, 7.2.1, 7.2.2, 7.2.3, 7.3.1, 7.4.3, 7.5.2, 7.5.3, 7.5.4, 7.5.5, 7.5.7, 7.5.8 e 7.7.3) [NBR]. A verificação é possível graças ao envio e execução dos *scripts* correspondentes aos nós da Grade, estes *scripts* foram escritos de forma a não necessitarem de acesso privilegiado ao sistema.

Este sistema é formado por quatro módulos distintos: o primeiro realiza a varredura da Grade, através das informações publicadas no *MDS* do *Globus*; o segundo realiza a enumeração de portas e *Fingerprint* do sistema operacional identificando, desta forma, os nós e os serviços ativos; o terceiro faz a verificação de conformidade, ou seja, realiza o envio e a execução dos *scripts* de coleta de dados nas máquinas da Grade; e o quarto e último módulo realiza o processamento/análise dos resultados obtidos pela execução dos *scripts*. [FAGUNDES 2005].

O acesso aos relatórios é feito através de um portal *web* implementado utilizando-se o *Apache Tomcat*, o qual fornece acesso aos dados logo após o processo de autenticação. O processo de autenticação usado baseia-se no fornecimento do *login name* e a senha associados ao certificado de usuário do *Globus*, na implementação deste processo foi utilizado o *toolkit Java CoG Kit* [LASZEWKSI 2003].

Devido à forma como o sistema foi implementado qualquer usuário pode iniciar um processo de análise concomitantemente com vários outros processos de análise estarem ocorrendo ao mesmo tempo. A análise é realizada por meio de *scripts* criados levando em consideração que os usuários teriam acesso restrito ao sistema.

Mesmo realizando uma análise, onde a visibilidade pode ser comprometida, este sistema proveu os meios necessários para validar a aplicabilidade do modelo de verificação a um ambiente de Grade, bem como abriu caminho para a criação do sistema descrito na Seção 6, o qual permite a execução de uma análise mais profunda do ambiente.

6. Solução Proposta

Ao contrário da solução anterior, que realizava apenas um monitoramento passivo da segurança do ambiente da Grade, esta realiza: o monitoramento das máquinas que compõem a rede; o monitoramento do estado das máquinas da Grade; a gerência do *Globus* e da segurança do ambiente.

Assim como o sistema citado na Seção 5, este faz uso do modelo de verificação definido em [GONÇALVES 2005], contudo esta implementação expande sua utilização, uma vez que fornece a possibilidade de validação de controles que não são parte da NBR ISO/IEC 17799:2005 [NBR], bem como permite a criação de novas políticas de segurança a serem aplicadas ao ambiente.

Os Agentes de Coleta de Dados (ACDs) definidos no modelo de verificação, em um ambiente distribuído construído utilizando-se o *Globus*, são os nós da Grade, visto que o próprio *middleware* fornece os meios para o envio e execução dos *scripts* de coleta e ajuste.

Desta forma, como mostra a Figura 2, para a implementação do modelo em um ambiente com o *Globus* instalado são necessário os *scripts* e o Agente Primário de

Verificação, responsável pelo envio destes *scripts* aos nós através do *GRIDFTP* e pela solicitação da execução dos mesmos através do *GRAM*.

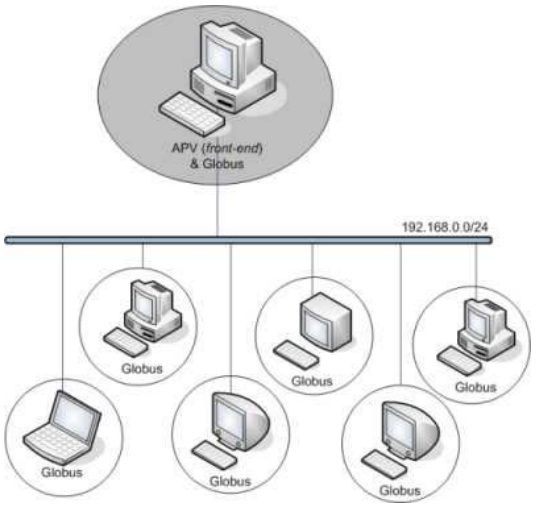

Figura 2. Rede Hierárquica de Comunicação – Grade

Tanto para o processo de envio e execução dos *scripts* de coleta quanto para o dos *scripts* de ajuste utilizar-se-á os recursos fornecidos pelo *Globus Toolkit*, conseqüentemente deve haver uma separação entre o controle de acesso ao *front-end* do sistema (sobre o qual o usuário poderá gerenciar/monitorar o ambiente), da parte do *back-end*, ou seja, da parte do sistema que irá realizar o processo de envio e execução dos *scripts*.

Esta segmentação se deve ao fato de que um usuário comum não possui, e nem deve ter, o nível de acesso necessário para prover os ajustes do ambiente. Caso todos os usuários possuíssem este nível de acesso qualquer um deles poderia, facilmente, comprometer seu funcionamento.

Logo, mesmo existindo vários usuários habilitados para o uso do *front-end*, todo o processo de análise e ajuste, realizado pelo *back-end*, será executado no ambiente sobre uma única conta local privilegiada, a qual será mapeada em um único usuário do *Globus*. Esta medida reduz a possibilidade da perda da confiabilidade, integridade e continuidade dos serviços fornecidos pela Grade.

Como já foi mencionado, esta solução possui quatro funcionalidades básicas:

Monitoramento dos nós da Rede : visa, através do uso de histórico, realizar um acompanhamento da atividade das máquinas que compõem a rede na qual a Grade está apoiada. Dentre as informações obtidas destacam-se: o estado do nó (ativo/inativo), as portas *TCP/IP* abertas, os serviços disponibilizados pelo nó, assim como seu sistema operacional e seu *uptime*.

Monitoramento dos nós da Grade : visa determinar e acompanhar, através do uso de um histórico, o estado dos serviços do *GRAM*, do *GridFTP* e do *MDS* em cada nó da Grade.

Gerência da segurança dos nós da Grade : Por funcionar de forma automática e quase sem interferência do usuário, a análise do ambiente proverá um elevado nível de

precisão, visto que o usuário não terá acesso aos dados antes deles serem processados, e por prover um ajuste semi-automático, que depende da confirmação do administrador. O sistema remove a necessidade do conhecimento das rotinas de segurança exigidas pelo ambiente. Na implementação desta funcionalidade serão utilizados *scripts* desenvolvidos para o *shell bash*, assim, a análise e ajuste do ambiente será realizado com o uso das próprias ferramentas existentes no ambiente.

Gerência dos serviços do *Globus* : baseado na mesma estrutura da gerência de segurança esta funcionalidade provê os meios para configurar os serviços disponibilizados pelo *Globus ToolKit*, inclusive o mecanismo de ajuste fino de data e hora.

Logo, a solução proposta dispõe de duas classes de serviços: o monitoramento e a gerência. Os serviços de monitoramento estão acessíveis a quaisquer usuários da Grade. As informações disponibilizadas visam facilitar o processo de submissão e execução de *jobs* na Grade, por outro lado, os serviços de gerência, tanto do *Globus* quanto da segurança, somente estarão disponíveis aos usuários devidamente cadastrados no sistema, pois durante o processo de gerência ações de nível privilegiado serão executadas nas máquinas da Grade. Outro fator que motiva o controle de usuários é a possibilidade do registro de suas atividades, facilitando a detecção do autor de operações que poderiam comprometer, ou que comprometeram, a integridade do ambiente.

6.1. Monitoramento de Recursos

Ambos os processos de monitoramento são executados de forma passiva, ou seja, não há a necessidade do usuário iniciar qualquer tipo de procedimento específico. Os processos de monitoramento são iniciados durante a carga do sistema operacional da máquina onde a ferramenta está instalada e continua ativo até que máquina seja encerrada.

Assim como ocorre no *SiMPaSeG* [FAGUNDES 2005], o monitoramento dos nós da rede é feito através de uma varredura de rede e pela coleta do *fingerprint* de cada nó. Com estas informações é possível determinar o sistema operacional, a versão do *kernel*, o tempo de atividade e os serviços disponíveis nas máquinas da rede [ARKIN 2005].

Já o monitoramento dos nós da Grade proposto neste documento, é realizado por meio de consultas periódicas ao servidor de *MDS* do ambiente, assim como ao módulo do *MDS* disponível em cada nó da Grade. Os principais dados obtidos em cada coleta são: estado do nó, do serviço de *MDS* local, do serviço *GRAM* e do serviço *GRIDFTP*; família do processador; quantidade de processadores; velocidade do processador em Hz; velocidade do processador em *BogoMIPS*; sistema operacional; disponibilidade de memória e a disponibilidade de espaço em disco.

Além de prover as informações sobre o estado atual da rede e da Grade a ferramenta permite aos usuários consultar o histórico do ambiente, dentre as informações que podem ser obtidas destacam-se: *uptime* médio de cada máquina da rede; utilização média de processador; utilização média de memória; e a quantidade média de nós ativos. Estas informações podem ser agrupadas por datas e/ou por intervalos de tempo (manhã, tarde, noite), possibilitando, assim, que o usuário acompanhe a atividade do sistema por um tempo previamente determinado pelo administrador. Consequentemente, durante uma submissão de tarefas poder-se-á escolher as máquinas de acordo com seu comportamento ao longo de um determinado tempo e não apenas pelo estado atual das mesmas.

6.2. Gerenciamento dos serviços da Grade

Apesar de ser relativamente simples o processo de instalação do *Globus Toolkit*, a gerência de um ambiente de Grade envolve muito mais do que a instalação do seu *middleware*, há a necessidade de se manter os relógios das máquinas sincronizados, de se realizar o controle de acesso aos recursos do ambiente e de validar os certificados, dentre outros fatores.

Para facilitar o processo de manutenção da Grade está incluída na ferramenta um módulo que permite realizar algumas ações básicas, contudo críticas, para a administração do ambiente, dentre elas destacam-se:

Sincronização dos Relógios : verifica a sincronização do relógio interno de cada um dos nós da Grade e quando necessário realiza sua sincronização segundo um servidor específico de *NTP*. O fornecimento deste recurso se justifica pela sensibilidade de várias aplicações distribuídas aos desvios de data e hora, principalmente aquelas que necessitam manter algum tipo de sincronismo.

Verificação da Validade dos Certificados das Máquinas : como o *Globus* faz uso certificados para autenticar os nós da Grade, há a necessidade da verificação da validade dos mesmos, visto que uma vez expirada a validade de seu certificado o nó da Grade deixa de atender às solicitações de *jobs* vindo das demais máquinas.

Verificação da Validade dos Certificados dos Usuários : assim como cada nó da Grade possui um certificado criado utilizando criptografia assimétrica os usuários também o possuem, só que neste caso, o certificado é utilizado na criação do *proxy* que permite a submissão/execução dos trabalhos. Quando estes certificados expiram os usuários deixam de ter acesso aos recursos da Grade. Para evitar que isto ocorra a ferramenta permite verificar validade dos certificados dos usuários e logo que estes expirarem ou estiverem por expirar um alerta é gerado e enviado ao administrador, assim como para o dono do certificado.

Administração dos *Mapfiles* do Ambiente : visando facilitar o processo de submissão de trabalhos entre nós de Grades pertencentes a domínios administrativos independentes e para prover um meio fácil de controle de acesso, o *Globus* usa um arquivo texto que realiza o mapeamento de um certificado de usuário em uma conta local, todavia cada máquina deve possuir seu próprio *mapfile* e nem todas as contas mapeadas em uma máquina são mapeadas em outra. O sistema permite o ajuste destes arquivos de forma remota, sendo possível construir um *mapfile* padrão para o ambiente, criar um *mapfile* a ser atribuído a um grupo de máquinas, realizar a inclusão e/ou a exclusão de linha(s) em uma ou várias máquinas. Esta funcionalidade evita que o administrador tenha que acessar cada nó da Grade para realizar as alterações manualmente.

As ações necessárias à execução dos ajustes serão realizadas por *scripts*, ou seja, não haverá necessidade da instalação de novos *softwares* para a realização dos mesmos, visto que serão utilizadas as funcionalidades já disponíveis no próprio ambiente, assim o *Globus* fornecerá os mecanismos para a os ajustes em seus próprios componentes.

6.3. Gerenciamento da segurança dos nós da Grade

A finalidade deste módulo é prover os mecanismos necessários para realizar a aderência do ambiente a uma política de segurança específica ou à norma nacional de segurança da informação. Para alcançar este objetivo a ferramenta permite ao administrador realizar

o cadastro da política bem como de seus controles, mas por outro lado, ela já fornece vários controles da NBR ISO/IEC 17799:2005 [NBR] que serão aplicados por padrão ao ambiente. Assim, mesmo que o administrador não tenha uma política definida os controles básicos serão aplicados.

Assim como os *scripts* de coleta utilizados na implementação descrita em [GONÇALVES 2005] e no protótipo *SiMPaSeG* [FAGUNDES 2005], os novos scripts foram construídos utilizando *shell script*.

Como mostra a Figura 3, o processo de ajuste/análise do ambiente pode ser decomposto em seis fases, na primeira ocorre à seleção dos nós a serem analisados, em seguida deve haver a seleção dos controles a serem verificados, tal seleção pode ser feita através da escolha de uma política previamente criada ou pela seleção da análise padrão (NBR ISO/IEC 17799:2005).

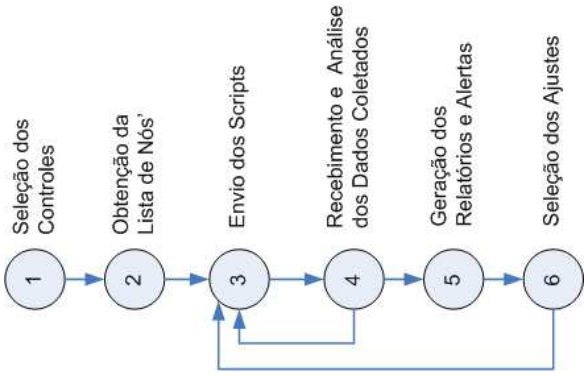

Figura 3. Fases do processo de ajuste/análise

Uma vez executado o *script* em um determinado nó, os dados obtidos são enviados de volta ao Agente Primário de Verificação (APV). Estes serão analisados e posteriormente gerará relatórios de conformidades e alertas de não conformidades. O processo de envio e execução dos *scripts* é repetido até que não haja mais controles a serem enviados. Encerrada a análise de um nó, o próximo da lista é selecionado e o processo de envio e execução recomeça, este só encerrará quando o último controle tiver sido analisado no último nó da lista. A Figura 4 apresenta um trecho do código do *script* que faz a verificação do controle 7.5.4 da NBR ISO/IEC 17799:2005 .

```
LOGIN_DEFS=0
if [ -f /etc/login.defs ]
then
  LOGIN_DEFS=1
  LOGIN_DEFS_PERMISSAO=`verific /etc/login.defs 0644 0 0`
  echo "login.defs:permissao:$LOGIN_DEFS_PERMISSAO"
  LOGIN_DEFS_PASS_MAX_DAYS=`egrep -v ^# /etc/login.defs
          | grep -i ^PASS_MAX_DAYS | awk '{print $2}'`
  echo "login.defs:MAX_DAYS:$LOGIN_DEFS_PASS_MAX_DAYS"
  LOGIN_DEFS_PASS_MIN_DAYS=`egrep -v ^# /etc/login.defs
          | grep -i ^PASS_MIN_DAYS | awk '{print $2}'`
  echo "login.defs:MIN_DAYS:$LOGIN_DEFS_PASS_MIN_DAYS"
fi
```

Figura 4. Código do Controle 7.5.4 da NBR ISO/IEC 17799:2005

Com base nas informações fornecidas pelo sistema o administrador poderá solicitar a aplicação dos ajustes de segurança. O ajuste do ambiente deve ocorrer de forma semi-automática, pois em determinados ambientes alguns pontos tidos como falhas de segurança são necessários para o funcionamento de outras ferramentas disponíveis, assim, a correção automática sem a confirmação do responsável pelo ambiente poderia levar a paralisação de serviços essenciais ao dia-a-dia dos usuários.

Todavia, para que a gerência do ambiente seja possível deve ocorrer o mapeamento da credencial do APV em todos os nós da Grade (feito por um usuário com acesso privilegiado) caso contrário algumas coletas de dados não serão possíveis, bem como a execução da maioria dos ajustes de segurança, já que estas alterações normalmente devem ser aplicadas à regiões críticas do sistema operacional e ao conjunto de *daemons* presente na máquina.

Deste modo, àquele que utiliza o sistema não necessita de conhecimento prévio sobre a aplicação dos mecanismos de segurança necessários a aderência do ambiente à NBR e/ou à política de segurança customizada.

7. Trabalhos Relacionados

Este trabalho propõe a implementação de uma ferramenta que atende a duas áreas bem distintas, o processo de monitoramento/gerência de recursos e o processo de gerência da segurança do ambiente da grade. Dentre os trabalhos relacionados a gerência de segurança destacam-se:

***COBRA - Consultive, Objective and Bi-functional Risk Analisis* :** esta ferramenta faz uso de questionários eletrônicos para prover a verificação do nível de aderência de um ambiente à Norma Internacional de Segurança da Informação (ISO/IEC 17799). Apesar de ser extremamente útil, ela tem sua capacidade de verificação limitada ao nível de conhecimento do usuário que responde aos questionários [C&A].

***Checkup Tool* :** esta solução foi desenvolvida pela Módulo, para contornar a limitação dos questionários eletrônicos fornecidos pelo sistema *COBRA*. Esta ferramenta faz uso de coletores escritos na linguagem de programação LUA, que devem ser executados manualmente em cada nó da rede a ser verificado/analisado. Com base nas informações coletadas serão gerados os relatórios de conformidades. Os coletores do *Checkup Tool* foram desenvolvidos tendo como base as soluções para servidores de rede da *Microsoft* [GALVAO 2004].

Protótipo Tamanduá-Mirim : este foi a primeira ferramenta desenvolvida tendo como base o modelo de verificação citado na Seção 4. Esta solução foi projetada para operar em um ambiente Linux e permitir que coletores, desenvolvidos em *shell scripts*, fossem enviados e executados nos Agentes de Coleta de Dados. Uma vez coletadas as informações, elas devem ser retornadas ao APV, que gera os relatórios de conformidade. Esta implementação permite uma coleta de dados *on-loco* e sem a interferência dos administradores do ambiente. Evitando assim, a inclusão de possíveis ruídos no processo de coleta e elevando consideravelmente a qualidade da análise. Entretanto, este protótipo requer que o ACD seja instalado e executado em todos os nós a serem analisados [GONÇALVES 2005] .

SiMPaSeG : assim como o protótipo anterior, esta ferramenta baseia-se no modelo de verificação citado na Seção 4. Esta solução foi a primeira a implementar a verificação de controles da NBR ISO/IEC 17799:2001 em um ambiente de Grades Computacionais. No entanto, ao contrário do protótipo original, esta implementação realizava uma verificação passiva, como citado na Seção 5, que não necessita da instalação de novos aplicativos no ambiente, nem da execução manual dos coletores [FAGUNDES 2005].

Quanto ao processo de monitoramento dos recursos de uma Grade, destaca-se o *NetLogger*, que permite realizar o monitoramento dos recursos disponíveis no ambiente da Grade através da instalação e execução de sensores nas máquinas a serem observadas. Estes sensores fazem uso do protocolo *LDAP* para permitir o registro e publicação de informações, que serão posteriormente armazenadas em um banco de dados que possua suporte à linguagem *SQL* [TIERNEY 2002].

Apesar do *NetLogger* permitir o monitoramento de recursos disponíveis em máquinas que não possuem o *middleware* da Grade instalado, ele não permite a gerência dos recursos disponibilizados pelo *Globus*. Neste ínterim, não é de conhecimento dos autores a existência de outra ferramenta, além da proposta por este trabalho, que permita a gerência e administração do ambiente provido pelo *Globus Toolkit*.

8. Resultados Obtidos

Até o presente momento somente os monitores e alguns coletores foram implementados, contudo já foi possível obter informações relevantes ao desenvolvimento da ferramenta.

Cruzando as informações fornecidas pelo monitoramento da rede com as fornecidas pelo monitoramento dos nós da Grade foi observado que muitas máquinas, mesmo possuindo todo o *Globus Toolkit* instalado e configurado, não apareciam na listagem de nós ativos na Grade, contudo as portas correspondentes aos serviços do *Globus* estavam "abertas" e aceitando conexões. Testes realizados posteriormente mostraram que o problema era no serviço prestado pelo servidor de *MDS* do ambiente, que ao ser consultado não apresentava as informações de alguns nós, mas quando se realizava uma consulta ao *MDS* local os nós retornavam todas as informações que deveriam ser publicadas pelo servidor.

Dentre os coletores implementados destacam-se aqueles relacionados ao gerenciamento do *mapfile* das máquinas. Com a execução manual destes coletores foi possível a criação de um *mapfile* padrão, o qual foi aplicado às máquinas servidoras. A remoção das linhas duplicadas dos *mapfiles* e a criação de um segundo modelo que foi aplicado às estações de trabalho da rede onde os *jobs* enviados ao *Globus* disputam recursos com os processos dos usuários.

9. Conclusões

Apesar desta ferramenta ser formada por soluções que já foram parcialmente implementadas de forma isolada pelo *SiMPaSeG* [FAGUNDES 2005] e pelo *NetLogger* [TIERNEY 2002], o diferencial da idéia proposta está no fato de que não é necessário a instalação de novos aplicativos nas máquinas da Grade, como ocorre com o uso do *NetLogger*.

Além de evitar a instalação de novos aplicativos em cada nó, esta nova solução permite que através de uma interface única os usuários possam ter acesso a várias informações relevantes sobre a carga e disponibilidade de recursos do ambiente.

Esta solução ainda permite ao administrador identificar as vulnerabilidades e aplicar ajustes finos de segurança ao ambiente, os quais não seriam normalmente possíveis de serem aplicados com tanta facilidade se realizados de forma manual, nem mesmo com o uso do protótipo apresentado por [FAGUNDES 2005].

Logo, com o mínimo de conhecimento sobre o ambiente da Grade, sobre o *Globus ToolKit* e sobre o funcionamento de políticas de segurança é possível que o administrador, através do uso desta solução, mantenha um controle fino sobre toda a estrutura e recursos fornecidos pelo ambiente.

As próximas fases do desenvolvimento da solução proposta incluem: o término da tradução dos controles utilizados pelo *SiMPaSeG*, de forma a permitir uma coleta mais abrangente do ambiente, visto que a implementação faz uso de coletores que requerem acesso privilegiado ao sistema; a seleção e tradução de outros controles da NBR ISO/IEC 17799:2005; a inclusão de controles de outras normas/*checklists*/manuais de segurança ao conjunto básico de controles da ferramenta e o término da implementação do *front-end* da solução.

Referências

ARKIN, O. (2005). A remote active os fingerprinting tool using icmp.

BUYYA, R. (2005). Special theme issue - grid computing: Making the global cyberinfrastructure for escience a reality. In *CSI Communications, Computer Society of India*.

C&A. Systems Security Ltd, COBRA. (2002) consultative, objective & bi-functional risk analysis, iso compliance analyst, release 3.1.8b.

CIRNE, W. (2002). Grids computacionais: Arquiteturas, tecnologias e aplicações.

FAGUNDES, F. d. O. (2005). Elaboração de uma politica de segurança para grades computacionais. Master's thesis, IME - Instituto Militar de Engenharia.

FOSTER, I. (2002). What is the grid? a three point checklist.

GALVAO, M. (2004). Módulo investe na automação do check-up tool.

GONÇALVES, L. R. d. O. (2005). Um modelo para verificação, homologação e certificação de aderência à norma nacional de segurança de informação - nbr iso/iec 17799. Master's thesis, COPPE Sistemas - Universidade Federal do Rio de Janeiro.

LASZEWKSI, G. v. (2003). The java cog kit user manual.

NBR. Associação Brasileira de Normas Técnicas (2005), NBR ISO/IEC 17799:2005 - Tecnologia da Informação - Código de Práticas para a Gestão da Segurança da Informação.

TIERNEY, B. (2002). Netlogger: A toolkit for distributed system performance tuning and debugging.

NextComp - Molecular Dynamics Application for Long-Range Interacting Systems on a Computational Grid Environment

Marcelo P. de Albuquerque[1], Alexandre Maia de Almeida[1], Leonardo Haas Peçanha Lessa[1], Márcio P. de Albuquerque[1], Nilton Alves[1], Luis Gregorio Moyano[1], Constantino Tsallis[1,2]

[1]Centro Brasileiro de Pesquisas Físicas (CBPF/MCT)
Rua Dr. Xavier Sigaud, 150, 22290-180 - Rio de Janeiro – RJ – Brasil

[2]Santa Fe Institute
1399 Hyde Park Road – Santa Fe – NM 87501 – USA

{marcelo, amaia, llessa, mpa, naj, moyano}@cbpf.br, tsallis@santafe.edu

Abstract. *This paper presents the ongoing activities of the NextComp project. This project has as main objective to investigate the validity of nonextensive statistical mechanics for systems with long-range interactions, in particular the Hamiltonian Mean Field model. This will be achieved through molecular dynamics (MD) simulations by a parallel program taking advantages from Charm++ language. Development and tests are carried out in a Linux cluster environment, being our interest focused on a grid environment in order to overcome the limiting factors of the simulations.*

1. Introduction

We are developing a parallel, object-oriented, molecular dynamics program for high performance simulation, referred to as NextComp, in order to analyze many-body infinite-ranged Hamiltonian systems. NextComp employs the message-driven execution capabilities of the Charm++ runtime system, allowing for parallel scaling on workstation clusters and grid environments [Albuquerque, 2005].

Our molecular dynamics (MD) program simulates a physical model called *Hamiltonian Mean Field* (HMF) which consists on N classical planar rotators, where each rotator i has as dynamical variables an angle θ_i and its conjugate momentum v_i and is defined by its Hamiltonian function [Antoni 1995]

$$H = \sum_{i=1}^{N} \frac{v^2_i}{2} + \frac{1}{N}\sum_{i,j=1}^{N}[1-\cos(\theta(i)-\theta(j))] \qquad (1)$$

The evolution of this system is governed by its Hamilton equations (derived from Equation 1) and this set of equations may be numerically integrated, thus making possible to know any physical observable at any time.

As computing and network technologies continue to improve high performance calculus, statistical physics domain takes an interest in use MD simulations on long-range interaction systems. Such simulations may easily exceed 10^7 elements having, additionally, a very high number of time steps as well as many realizations in order to achieve bounds which provide a desired level of statistical confidence.

The computing resources necessary to perform MD simulations typically diverge when calculating long-range interactions. It remains a great challenge to carry

out high performance of MD software because it involves the simulation of several small elements, where the computation related to each element depends globally on *all* other elements.

Another difficulty is that the physical limits of interest are commonly very high (more precisely, one is forced to deal with infinite limits). This is of great significance to the NextComp Project, for reasons that will be explained in more detail in Section 2, where we present the Hamiltonian Mean Field model and its relation with the formalism of nonextensive statistical mechanics. Section 3 shows the sequential computational problem and the parallelization strategy used to speedup our MD calculus. In section 4 we make a performance analysis presenting two concrete experiments. We close our paper with conclusions and a brief outlook.

2. The Physical Problem

Classical physics, and particularly statistical mechanics, studies systems formed by elements that interact through forces. Usually, these forces have a dependency with the distance between any two elements, *r*. In most cases, the force is strong when the inter-particle distance is small, and weak when the elements are far apart.

Depending on the intensity of these forces in the long distance limit (i.e., when $r \to \infty$), this interaction may be classified as *short* or *long range interaction*. More precisely, in a system with dimension *d*, and a force between elements that behaves asymptotically (i.e., when $r \to \infty$) as $r^{-\alpha}$, the system is long-range if $\alpha \le d$ and short-range if $\alpha > d$. The limit case $\alpha = 0$ the system does not depend on the distance and it is called an infinite range system.

Examples of systems with long-range interactions [Dauxois 2002] are gravitational systems, Coulombian systems, dipolar systems, fractures, finite systems, etc. The properties of these systems still remain to be explained, even though their evident importance. The main challenge regarding these systems is the construction of a thermodynamics that may describe them correctly and may as well explain the similarities and differences with their short-range counterparts.

This is one of the main points of interest in nonextensive statistical mechanics [Tsallis 1988]. Nonextensive statistical mechanics is a formalism formulated by one of us (CT) in 1988, that generalizes the usual Boltzmann-Gibbs (BG) statistical mechanics. This formalism is based in a generalization of the conventional entropy $S = k \sum p \ln p$ that includes a parameter *q* (that generally depends on the dynamics of the problem, or on its universality class), $S_q = k\,(1 - \sum_i p_i^q)\,(q-1)^{-1}$ where $S_q \to S$ when $q \to 1$.

An example of a long-range system is the Hamiltonian Mean Field (HMF), a system formed by planar classical rotators.

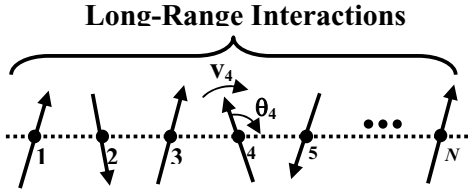

Figure 1: The Molecular dynamics uses a very large array of *N* planar rotators.

In Figure 1 we show schematically the model. The *N* rotators are symbolically depicted as black dots, and that are able to move only over the big circle. Rotators are

considered to pass through each other without collisions (which is completely equivalent to perfect elastic collisions). Each rotator *i* has an associated angle θ_i that depends in time, and therefore for any two rotators *i* and *j*, the angle between them $(\theta_i - \theta_j)$ also depends in time. Moreover, each rotator *i* has an associated angular momentum v_i (here, without loss of generality, we are assuming unitary inertial moment).

The interaction force between rotators *i* and *j* is proportional to the angle difference

$$F_{i,j} \propto sin(\theta(i) - \theta(j)) \tag{2}$$

The total force in each rotator determines its time dependence. In this model the force in each rotator is influenced by every other rotator

$$F_i = \frac{1}{2N} \sum_{j=1}^{N} sin(\theta(i) - \theta(j)) \tag{3}$$

The fact that the sum includes every rotator means that the interaction range is infinite (any rotator has an influence on any other). This simple model reflects many realistic characteristics of systems with long-range interactions such as galaxies or plasma gas [Dauxois 2002].

The thermodynamic limit of this model may be analytically solved in the microcanonical ensemble. This means that, given a specific energy (a constant in the system), the value of any mean macroscopic observable such, as the temperature, may be predicted for equilibrium. But it is known that, for certain values of initial conditions, the system may be trapped in states where the mean microscopic quantities stay approximately constant for long periods of time with different values than those predicted by the BG theory. This happens, for example, when initial conditions correspond to *waterbag* initial conditions, widely used in the literature [Anteneodo et al. 1998]. This type of initial conditions consists in $\theta_i = \theta_0 \quad \forall i = 1 \ldots N$, (i.e., all angles aligned and momenta taken arbitrarily from a uniform distribution, $v_i \in [-c, c] \quad \forall i = 1 \ldots N$, where *c* is a constant. This constant is properly defined fixing the total specific energy $E(v(i), \theta(i)) = Ne$. For the angles, without loss of generality, we choose $\theta_0 = 0$.

Using this *waterbag* initial conditions, quasistationary states appear for certain values of the specific energy *e*, slightly below *e*=0.75. In these quasistationary states, the temperature, defined as

$$T = \frac{\langle 2K \rangle}{N} = \frac{1}{N} \sum_{j=1}^{N} v(i)^2 \tag{4}$$

where $\langle \cdot \rangle$ means average between rotators, is almost constant in time and has a lower value than the one expected at equilibrium , $T_{QSS} < T_{BG}$ (see Figure 2). This result means that the time and size limits are not commutable ($N \to \infty \quad t \to \infty \neq t \to \infty \quad N \to \infty$), a main conjecture of nonextensive statistical mechanics. Furthermore, the duration t_{QSS} of the quasistationary state (defined, for example, as the time in which occurs the inflection point when the temperature relaxes to equilibrium) grows with the system size *N*, indicating that they are indeed relevant in the thermodynamical limit ($t_{QSS} \to \infty$ when $N \to \infty$). Consequently, our purpose is to simulate the evolution of this system for large

values of N to verify the applicability of nonextensive statistical mechanics to this model.

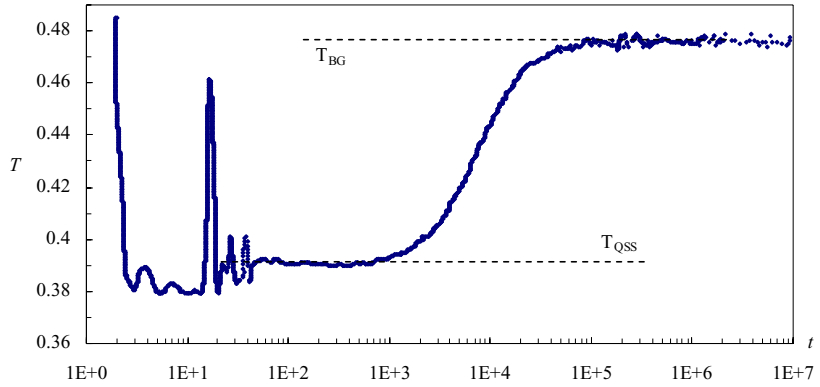

Figure 2: Temperature evolution shows a quasistationary plateau followed by a relaxation to final equilibrium.

In order to do this, we numerically simulate the Hamiltonian equations of the system. This set of differential equations define the way the rotators move

$$\left.\begin{aligned} \frac{d}{dt}\theta(i) &= v(i) \qquad\qquad 1 \le i \le N \\ \frac{d}{dt}v(i) &= m_y \cos(\theta(i)) - m_x \sin(\theta(i)) \end{aligned}\right| \tag{5}$$

where $\bar{m} = \frac{1}{N}\sum_{j=1}^{N}[\cos(\theta(j)), \sin(\theta(j))]$ is the magnetization of the system.

The total specific energy e of this system is a conserved quantity, i.e., constant at any time. To solve this set of differential equations numerically, they must be discretized, and this may have as consequence a poor total energy conservation, yielding an incorrect dynamics (i.e., the wrong time evolution). Thus, we use a special algorithm to solve sets of differential equations, the symplectic Yoshida integrator [Yoshida 1990], that guarantees a good conservation of the total specific energy and therefore correct from the physical point of view.

Additionally, the system presents statistical fluctuations in the value of its macroscopic observables (such as its temperature) due to the random character of the initial conditions (note that this is not caused by the Hamilton equations, which are deterministic). In order to reduce these fluctuations, we take averages from several realizations of the same simulation (i.e., different microscopic initial conditions consistent with equal macroscopic parameters, as for example, total specific energy).

3. NextComp Parallelization Strategy

The NextComp MD creates, as stated in Section 2, a set of N planar rotators (N>>1) where each rotator has associated a pair of state variables: its angle θ and its momentum v (see Figure 1). Numerical integration of the Hamilton equations (5) provides the state of the rotator at each time step t.

Figure 3 shows a simple diagram of the sequential NextComp MD algorithm (NextComp-MD-S). The first procedure includes initializations steps and a simulation loop (that takes account for different statistical realizations). In the simulation loop,

there is an appropriate initialization of the pair (θ, v), and preliminary iterations that represent a transient time. Then we start the integration loop calculating the dynamics.

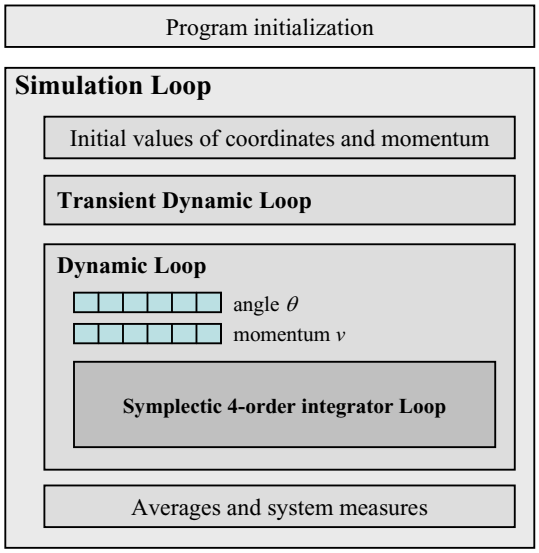

Figure 3: Diagram of the serial NextComp MD algorithm.

In this dynamical loop we use the symplectic fourth-order integration method proposed by Yoshida in order to conserve the total energy of the system for long time runs. This symplectic integration uses a four loop as follows in each time step t

```
1. s = 0
2. while s < 4
     2.1. θ(t+1,n) = f(v(t)) ∀ N
     2.2. Calculate Magnetizations m_{x,y}(t+1) = f(θ(t+1,n)) ∀ N
     2.3. Calculate interactions forces F(t+1) = f(m_{x,y}(t), θ(t+1,n)) ∀ N
     2.4. v(t+1,n) = f(F(t+1), θ(t+1,n)) ∀ N
     2.5. s = s + 1;
3. Calculate the kinetic energy of the system
```

Only after the fourth step the calculation of θ(t+1) and v(t+1) are complete. Finally some physical properties are averaged and measured.

To parallelize the NextComp MD (NextComp-MD-π) algorithm, we used the same approach as the NAMD project for Biomolecular Simulations on thousands of processors [Kale 2002]. NAMD scalability challenge was tackled adopting message-driven approaches.

NextComp dynamic components are implemented in the Charm++ parallel language [Charm 2006]. Charm++ implements an object-based message-driven execution model. In Charm++ applications, there are collections of C++ objects, which communicate by remotely invoking methods on other objects by messages. More precisely, the problem can be decomposed into objects, and since we decide the granularity of the objects, we can control the degree of parallelism. Objects encapsulate states, Charm++ objects being only allowed to directly access their own local memory. Access to other data is only possible via asynchronous method invocation directed to other objects.

Programs written in Charm++ consist in parallel objects called chares that communicate with each other through asynchronous message passing. When a chare

receives a message, the message triggers a corresponding method (or entry points –EPs) within the chare object to handle the message asynchronously. Further, chares may be organized into one or more indexed collections called chare arrays. Messages may be sent to individual chares within a chare array or to the entire chare array simultaneously.

Charm++'s parallel objects and data-driven execution adaptively overlap communication and computation, and hide communication latency: when an object is waiting for some incoming data, entry functions of other objects whose data are ready to execute.

For the NextComp-MD-π, we first decomposed the coupled rotators in objects where each one can only access a group of (θ, v), G. We have defined a rotator class and instantiate it as parallel chare array in a charm environment. This class was designed to use Yoshida symplectic integrator in a parallel algorithm. The first stage of the algorithm computes the angle θ and makes a reduction in $m_{x,y}$, (defined in Equation (5)), i.e. collect and sum all $m_{x,y}$ from all objects and distribute the result to all rotators objects in order to start the second stage (see Figure 4).

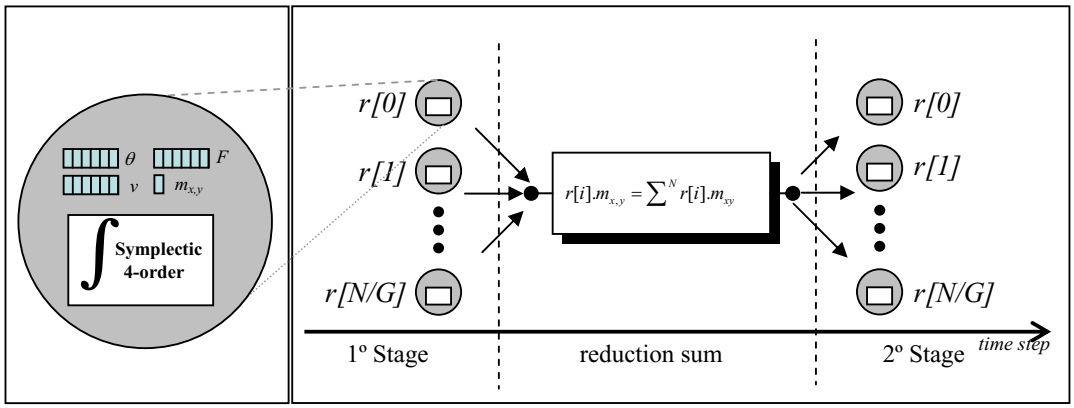

Figure 4: Diagram of the 4-order Yoshida integrator. (a) Rotator objects. (b) Parallel implementation and the synchronization stage for computation the $m_{x,y}$ sum.

The symplectic integration is the most important hotspot of the program. The parallel version needs to synchronize processes in order to compute $m_{x,y}$. NextComp-MD-π is a tightly coupled parallel program, meaning that all processes needs to exchange data at regular intervals, leading to communication latency.

The statistical mechanics point of view is focused generally in both large N and t limits. In particular, in this model (which has infinite range interactions, i.e. every pair of rotators interacts) most (common) integration algorithms will scale as $O(N^2)$, or, at most, $O(N \ln(N))$ may be achieved using optimizations taking advantage of internal symmetries [Anteneodo et al 1998]. In consequence, computational times might be unreachable, indicating that this model is a appropriate candidate for grid computing.

4. Performance Analysis

Performance is of paramount importance in parallel programming. The NextComp-MD-π development purpose should be both to solve the physical problem faster than its sequential version, and to deal with a larger physical system that could not be attained by the previously one.

We conducted three sets of experiments in the CBPF SSolar Linux Cluster [SSolar 2006] and in the NCSA[1] Xeon Linux Cluster. The first experiment accomplished was the measurement of the total execution time and thus the speedup of the system, leading to the optimal grain size computation. The second experiment corresponds to the analysis of the object execution time and the performance distribution for all object entry points. We also monitor the processors activities inspecting the program time line. Finally, the third experiment presents the method used to diminish the processors idle times.

A valuable tool to analyze in detail the parallel performance is Charm++'s *Projections*, a Java-based visualization tool (PVT) [Projections 2006]. PVT consists in an efficient automatic tracing and an interactive graphic analysis system.

All the experiments run with fixed $N=10^4$ rotators. Our choice of size N was only with the purpose of measuring performance and validates the NextComp-MD-π version. Note that N should be several orders of magnitude greater than 10^4 and the number of time steps should be large enough to verify the application of the nonextensive theory. More precisely, should be large enough to reflect the relaxation to the BG regime that, as explained in section 2, occurs at later times as N gets larger.

4.1. First Experiment

The execution time is the first metric of performance, and measures how long the NextComp-MD program runs a fixed number of time steps. Through the execution time one can compute the speedup as a function of the number of processor used in parallel calculus.

Since the heaviest computation procedure happens in the rotator-object due to the fourth-order integrator we analyzed its grain size. It was done by controlling the amount of computation per object, i.e. ranging G form 1 to *N*. In this experiment we set P= [1, 5, 10 and 20] for the number of processor and N/P as the number of rotator-objects instantiated. Table 1 presents the execution time as a function of P and G.

NextComp-MD-π — Execution Time (s)

G	P=1	P=5	P=10	P=20
1	215.50	39.92	20.59	21.90
10	38.04	8.56	6.95	8.37
20	28.27	7.54	6.22	7.77
50	23.45	6.97	5.94	7.52
100	22.73	6.90	5.82	7.28
250	21,42	6.58	5.84	7.09
500	23.51	6.81	5.64	7.13
1000	22.18	6.69	**5.62**	8.35
2000	22.45	6.43	8.29	10.69
5000	20.90	12.87	14.95	17.99
10000	20.82	23.24	25.53	29.12

Table 1: Amount of computation per object size G analysis for N=10000 rotators. The best configuration for the NextComp-MD-π is P=10 and G=1000.

[1] The NCSA proposal is related to the NextComp Project — "*Molecular Dynamics for Long-Range Interacting Systems and its Possible Connection with Non-Extensive Mechanics Theory*". NCSA Proposal Number: PHY060015.

The serial performance of the system corresponds to P=1 and G=10000, where the execution time was about 20.82 s. Table 1 shows that the best configuration of the system was achieved when G=1000 and P=10. In this condition, the speed up was ≈ 3.70 and the processor efficiency, calculated by equation (6), is $Efficiency_{10}$=37%.

$$Efficiency_P = Speedup(P) / P \quad (6)$$

Another measure of performance, similar to the efficiency, is the efficacy [Goshal 1991], useful for determining the best cost-effective number of processors for a particular application

$$Efficacy_P = Efficiency_P \; Speedup(P) \quad (7)$$

An important property of the efficacy is that its maximum corresponds to an optimal system operating point [Eager 1989]. When a new processor is added to the computation with a resulting increase in efficacy, then this gain is measured by the cost of adding it. When, on the contrary, the efficacy diminishes, then the cost of the addition outweighs the potential performance gain. For G=1000, the maximum efficacy is 1.94 for P=5. For G=2000 it is 2.10, also attained for P=5 processors (Table 2).

NextComp-MD-π Metrics

G	Speedup				Efficacy			
	P=1	P=5	P=10	P=20	P=1	P=5	P=10	P=20
1	0.10	0.52	1.01	0.95	0.01	0.05	0.10	0.05
10	0.55	2.43	2.99	2.49	0.30	1.18	0.90	0.31
20	0.74	2.76	3.35	2.68	0.54	1.52	1.12	0.36
50	0.89	2.99	3.50	2.77	0.79	1.78	1.23	0.38
100	0.92	3.02	3.58	2.86	0.84	1.82	1.28	0.41
250	0.97	3.16	3.57	2.93	0.94	2.00	1.17	0.43
500	0.89	3.06	3.69	2.92	0.78	1.87	1.36	0.43
1000	0.94	3.11	**3.70**	2.49	0.88	1.94	1.37	0.31
2000	0.93	3.24	2.51	1.95	0.86	2.10	0.63	0.19
5000	1.00	1.62	1.39	1.16	0.99	0.52	0.19	0.07
10000	1.00	0.90	0.82	0.71	1.00	0.16	0.07	0.03

Table 2: Speedups and efficacy for several configurations of the system.

Figure 5 highlights the behavior of the execution time and the amount of computation per rotator-object (G) for NextComp-MD-π. In (a) we show that for each number of processors used in the simulations there is a minimum G, i.e. the best configuration of the system for each P. In Figure 4 (b) we put in evidence that when we add a processor to the calculation there is also a better configuration. This is expected due to the amount of time spent in communication and due to the increase of related activities with growing number of objects.

4.1. Second Experiment

We perform analysis of the object execution times for different number of entry points (EP's). Most object execution times are 54.56± 4.91 μs corresponding to the EP that computes the fourth-order integrator.

The timelines in Figure 6 show a detailed view of a NextComp-MD-π realization. It illustrates the parallel task distribution over ten processors during the runtime of the program between 5.25 s and 5.30 s. We can observe the simultaneous use

of all processors. However, the timelines graph also points out that the parallel tasks have sections of idle time. This idle time is due to the frequent periods of communication because of the synchronization events of the fourth-order integrator.

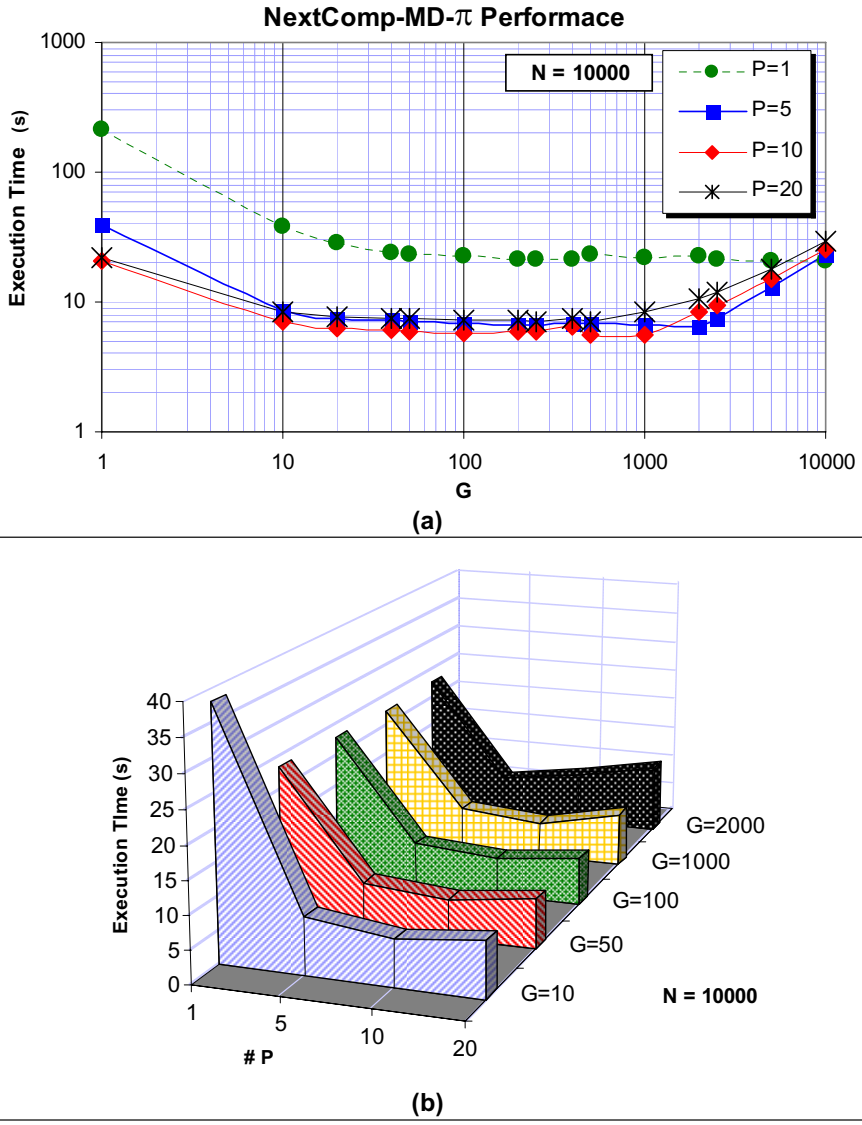

Figure 5: (a) Dependence of the execution time on the amount of computation per rotator-object G. (b) Execution time as a function of number of processors P. Note the lose of performance due to task communication when P goes from #10 to #20.

The reduction made at every time step to compute the magnetization $m_{x,y}$ produces this dependence of data synchronization. This idle time is possibly associated to the load imbalance and can affect the execution time of NextComp-MD-π.

4.3. Third experiment

In fact, to estimate the parameters of physical interest in our problem we need to generate and analyze several realizations (i.e., random samples) and obtain a statistical average of the results. These realizations are controlled by the simulation loop explained in section 3 (Figure 3). We measured the execution time for four simulations loops of NextComp-MD-π as a function of P and G. The results are shown in Figure 7. This is a

very interesting situation due to the emergence of independent tasks for each simulation. It allows Charm++ runtime system to enhance task distribution and use processors idle time. The best configuration achieved of the system in these circumstances corresponds to G=200 and P=10. As expected, the speedup increases up to ≈ 5.03 and processor efficiency increases to $Efficiency_{10,4} \approx 50\%$.

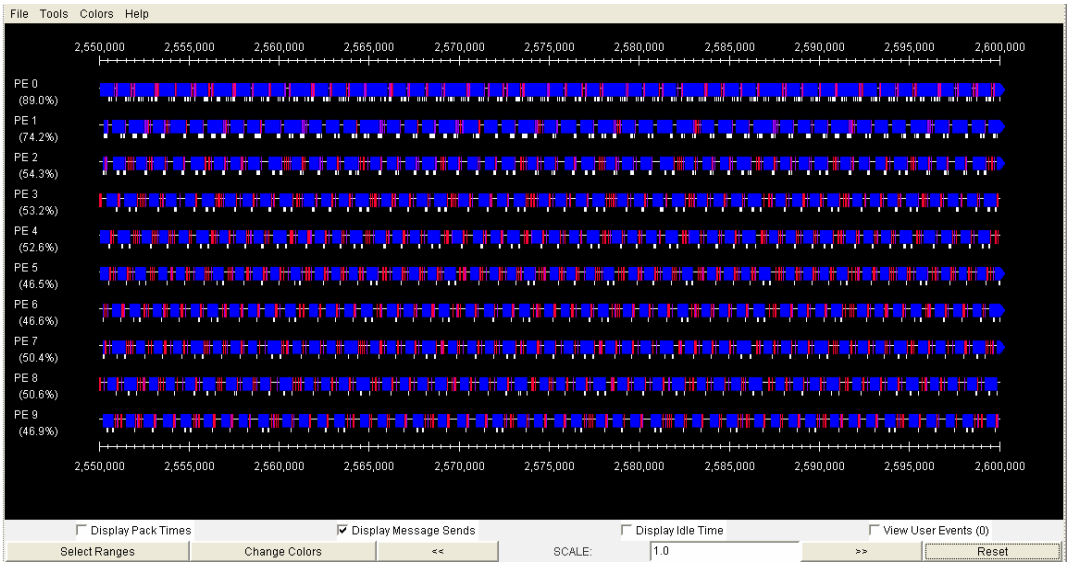

Figure 6: Charm++ Java *Projections* Timeline Tool analyzing tracedata from NextComp-MD-π. The figure presents the runtime on NCSA Xeon Linux Cluster on a window of Δ_t=40ms from 2.55 s to 2.95 s.

5. Conclusions

In this paper we presented a typical problem of computational physics that explores a system model for which no solution is known. When modeling a physical system in a computer program we have to worry about its correct mathematical representation and also with program performance. Under the statistical mechanical point of view, investigations generally focus in both large identical elements and large time limits. Computational time can be unreachable, indicating that this kind of problem is a suitable candidate for parallel computing and grid deployment.

For this reason we started the NextComp project, i.e. the development of a parallel algorithm for molecular dynamics simulation, which leads to a complex program structure. NextComp applications need to be capable of quick scalability on machines with thousands of processors and hence we decided to use message-driven objects, implemented in the Charm++ runtime system. NextComp in parallel is very challenging because of the Hamiltonian Mean Field model compels a tightly coupling application.

In this work, we performed some experiments for performance analysis. In the first one we obtained the metrics for the NextComp-MD-π; for the second one we show graphically the simultaneous processing of the program. The best parallel configuration achieved runs five times faster than its serial version. We can therefore confidently state that the results reported in this work are conservative and indicative of true performance for a production run. Nevertheless, the explored techniques show good parallel scalability allowing observations of very large physical systems.

However, we expect to achieve a better speedup by optimizations of NextComp-MD-π code. One goal is a new design of the parallel symplectic fourth-order integrator exploring the advantages of internal symmetries of the physical problem. The other one could be the use of the load balancing available in the Charm++ system that provides a variety of mechanisms and strategies to distribute tasks. Load balancing is a critical factor to achieve optimal performance in parallel applications. NextComp-MD is an iterative computation consisting in a combination of a number of time steps and calculus iterations. As the symplectic fourth-order integrator is subject to a barrier synchronization point, it forces the slowest task to determine the overall performance. Furthermore, in NextComp-MD-π the load of each individual objects is correlated with the communication patterns of consecutive iterations. In this case, Charm++ can migrate objects to create a reasonable computation and communication balance, and it also employs continuous-monitoring strategies to fine-tune load balance. Consequently, we believe that the distributing work among tasks may minimize the idle time.

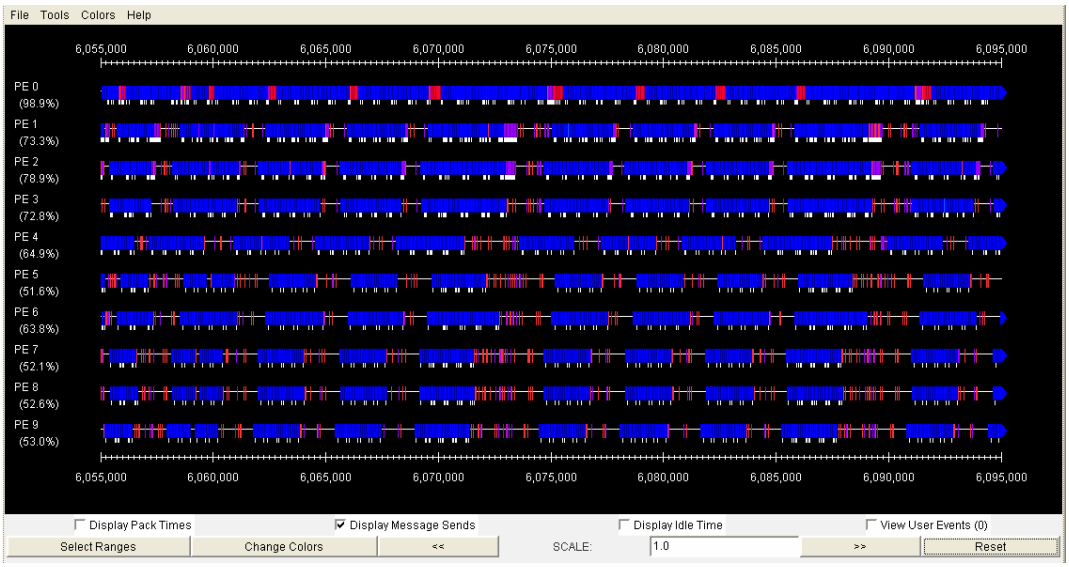

Figure 7: Timeline for NextComp-MD-π running four realizations in parallel on the NCSA Xeon Linux Cluster. Runtime on a window of Δ_t=40ms from 6.05 s to 6.09 s. The CPUs usages (left column – PE #) are higher showing a better utilization of all processors.

We intend to perform further experiments to investigate several areas with the NextComp-MD-π version described in this paper. We intend to verify the communication latencies by measuring the time it takes to send and receive data over a HPC with dedicated high performance switches (e.g. in CBPF SSolar and NCSA Xeon Linux Clusters).

Finally, and perhaps most importantly, our future work in using parallel message-driven objects is the use of NextComp-MD-π running across multiple clusters in a Grid environment, e.g. the INTEGRIDADE Linux Cluster across the RNP Giga Network. The INTEGRIDADE Project uses Globus [Globus 2006] to create "virtual organizations" sharing computational resources owned by different real organizations. A mechanism for providing latency tolerance presents serious challenges for deployment of parallel programs in Grid environments. Masking the effects of wide-area latency is critical for achieving good performance and actually most work involving deployment of tightly-coupled applications on computational Grids has been focused on algorithm-level modifications.

6. Acknowledgment

This work was supported by the Brazilian National Council of Scientific and Technological Development (CNPq/MCT). The NextComp Project is also part of the "INTEGRIDADE - Middleware Development for Computational Grids over the infrastructure for advanced research network", of RNP (Brazilian National Education and Research Network) GIGA Project - in collaboration with National Laboratory for Scientific Computing (LNCC) and Fluminense Federal University (UFF) in Brazil and the National Center for Supercomputing Applications (NCSA) in USA.

7. References

Albuquerque, Marcelo P.; Albuquerque, M. P.; Alves, N.; Peixoto, D.; Moyano, G. L.; Tsallis. C.; Baldovin, F.; Giupponi, G. (2005); "*Dinâmica Molecular em Ambiente Computacional*", In: Annals of "III Workshop on Computational Grids and Applications".

Kale L. V., Phillips J. C., Zheng G. and Kumar S. (2002); "*NAMD: Biomolecular Simulation on Thousands of Processors*". In Proceedings of Supercomputing, The SCxy Conference series, Baltimore, Maryland, November 2002. ACM/IEEE.

Charm (2006), "*The CHARM++ Programming Language Manual – Version 5.8*" Parallel Programming Laboratory Group of the University of Illinois at Urbana-Champaign; http://charm.cs.uiuc.edu/manuals/html/charm++/manual.html, April.

Projections (2006), "*Projections Manual – Version 2.1*", Parallel Programming Laboratory at University of Illinois at Urbana-Champaign, http://charm.cs.uiuc.edu/manuals/html/projections/manual.html, April

Goshal D, Serazzi G., and Triphath S. (1991), "*The processor working set and its use in scheduling multiprocessor systems", IEEE Transactions on Software Engineering*, 17(5):443-453.

Eager D. L., Zahorjan J., and Lazowska E. D. (1989), "*Speedups versus efficiency in parallel systems", IEEE Transactions on Software Engineering*, 38:406-423.

SSolar (2006), "*CBPF SSolar Linux Cluster*", http://mesonpi.cat.cbpf.br/ssolar, April.

Globus (2006), "*The Globus Toolkit is an open source software toolkit used for building Grid system*", http://www.globus.org, April.

T. Dauxois, S. Ruffo, E. Arimondo and M. Wilkens; (2002); eds., "*Dynamics and Thermodynamics of Systems with Long Range Interactions"*, Lecture Notes in Physics Vol. 602, Springer (2002), and references therein.

C. Tsallis (1988), *J. Stat. Phys*. **52**, 479; "*Nonextensive Mechanics and Thermodynamics",* in Non-Extensive Entropy Interdisciplinary Applications, edited by M. Gell-Mann and C. Tsallis, (Oxford University Press, Oxford, 2004).

M. Antoni and S. Ruffo (1995), *Phys. Rev. E* **52**, 2361.

C. Anteneodo and C. Tsallis (1998), Phys. Rev. Lett. 80, 5313; F. Tamarit and C. Anteneodo, *Phys. Rev. Lett.* **84**, 208 (2000); M.C. Firpo and S. Ruffo, *J. Phys. A* **34**, L511 (2001); V. Latora, A. Rapisarda, and C. Tsallis, *Phys. Rev. E* **64**, 056134 (2001);

H. Yoshida (1990), *Phys. Lett. A* **150**, 262; F. Neri (1988), "*Lie algebras and canonical integration"*, Department of Physics, University of Maryland, preprint.

Sand Castle: infra-estrutura segura para hospedar dinamicamente aplicações em grades computacionais

Leonardo Mattes, João Antonio Zuffo

Laboratório de sistemas integráveis – Universidade de São Paulo (USP)
Caixa Postal 15.064 – 91.501-970 – São Paulo – SP – Brazil

{leo,jazuffo}@lsi.usp.br

Abstract. *The grid paradigm aims a better exploitation of the computational resources in a distributed and flexible systems, however its utilization brings new challenges regarding security. One of the foreseen grid functionalities creates dynamic jobs and services, as well as operation bringings flexibility to the system that should be exploited for installations of "malicious" applications. The present work presents Sand Castle, a tool that aims to host and control Java applications in grid computing environments, and, once it is hosted, its applications has to be monitored and controlled in according to security polices, that has been established by the grid environment.*

Resumo. *O paradigma grades computacionais objetiva um melhor aproveitamento de recursos computacionais por meio de sistemas distribuídos e flexíveis. Sua utilização, contudo, traz novos desafios do ponto de vista da segurança. Uma das funcionalidades previstas para as grades computacionais permite que tarefas e serviços possam ser instanciados remotamente de forma dinâmica, operação que além de trazer maior flexibilidade ao sistema, pode ser explorada para a instalação de aplicações "maliciosas". O presente trabalho apresenta Sand Castle, ferramenta que tem por objetivo hospedar e controlar aplicações Java em grades computacionais, de forma que as mesmas, uma vez instanciadas em um determinado host, tenham suas ações monitoradas e controladas por políticas de segurança, estabelecidas para o ambiente da grade computacional.*

1. Introdução

O paradigma grade computacional [Foster 2005] [Bavier et al. 2004] faz referência a sistemas, cuja a principal característica é o compartilhamento de recursos e serviços de forma integrada e dinâmica entre múltiplos domínios lógicos e tecnológicos, no intuito de proporcionar um melhor aproveitamento e gerenciamento dos recursos disponíveis. Em contra partida, por apresentar um modelo de maior cooperação e compartilhamento de recursos, a grade computacional traz desafios de segurança ainda não totalmente equacionados [Welch 2003]. Entre as novas funcionalidades propostas dentro de uma grade, a capacidade de instanciação e controle de tarefas e serviços remotos de forma dinâmica [Keahey et al. 2003], além de proporcionar maior flexibilidade, pode permitir a instalação de aplicativos "maliciosos" ou com falhas de segurança, exigindo assim a existência de mecanismos para controlar as ações das aplicações instanciadas.

Outro aspecto importante em relação à segurança em grades computacionais, diz respeito a mudanças na infra-estrutura de serviços de rede. Os projetos GLOBUS [Foster 2005] e EuroGrid [EuroGrid] apontam o desenvolvimento de ferramentas de grade computacional

Sumário / Summary

Comitê de Programa do WCGA 2006

Alba Melo (UnB)
Alexandre Sztajnberg (UERJ)
Antônio Tadeu A Gomes (LNCC)
Artur Ziviani (LNCC)
Bruno Schulze (LNCC)
Carlos Becker Westphall (UFSC)
Celso Mendes (UIUC)
Claudio Geyer (UFRGS)
Edison Ishikawa (IME-RJ)
Edmundo Madeira (UNICAMP)
Elias Duarte Jr. (UFPR)
Francisco Brasileiro (UFCG)
Inês Dutra (UFRJ)
Jairo Panetta (INPE/CPTEC)
Luciano Gaspary (UNISINOS)
Luis Carlos Trevelin (UFSCar)
Marcelo Albuquerque (CBPF)
Michael Stanton (RNP e UFF)
Noemi Rodriguez (PUC-Rio)
Osvaldo Carvalho (UFMG)
Philippe Navaux (UFRGS)
Rossana M de Castro Andrade (UFC)
Simone Martins (UFF)
Tereza Cristina Carvalho (USP)
Thais Vasconcelos Batista (UFRN)
Vinod Rebello (UFF)
Walfredo Cirne (UFCG)

Comitê de Programa do WCGA 2006

Alba Melo (UnB)
Alexandre Sztajnberg (UERJ)
André Detsch (UNISINOS)
Angelo Perkusich (UFCG)
Antônio Tadeu A Gomes (LNCC)
Artur Ziviani (LNCC)
Ayla Débora Dantas de Souza (UFCG)
Bruno Schulze (LNCC)
Carlos Westphall (UFSC)
Celso Mendes (UIUC)
Claudio Geyer (UFRGS)
Edison Ishikawa (IME-RJ)
Edmundo Madeira (UNICAMP)
Eliane Araújo (UFCG)
Francisco Carvalho (UFC)
Francisco H de Carvalho Júnior (UFC)
Inês Dutra (UFRJ)
Luciana Lima (LNCC)
Luciano Gaspary (UNISINOS)
Luis Carlos Trevelin (UFSCar)
Luiz Bittencourt (UNICAMP)
Marcelo Albuquerque (CBPF)
Michael Stanton (RNP e UFF)
Milena Pessoa M Oliveira (UFCG)
Nicolas Maillard (UFRGS)
Noemi Rodriguez (PUC-Rio)
Osvaldo Carvalho (UFMG)
Rossana M de Castro Andrade (UFC)
Simone Martins (UFF)

Realização

Comitê de Organização

Coordenação Geral
Carlos Alberto Maziero, PUCPR
Elias Procópio Duarte Jr., UFPR
Keiko Verônica Ono Fonseca, UTFPR

Coordenação do Comitê de Programa
Joni da Silva Fraga, UFSC

Coordenação de Tutoriais
Raimundo José de Araújo Macêdo, UFBA

Coordenação de Minicursos
Edmundo Roberto Mauro Madeira, UNICAMP

Coordenação de Palestras e Painéis
Edmundo de Souza e Silva, UFRJ

Coordenação do Salão de Ferramentas
Thaís Vasconcelos Batista, UFRN

Coordenação de Workshops
Emílio Carlos Gomes Wille, UTFPR

Coordenação de Publicação
Mauro Fonseca, PUCPR

Comitê Consultivo
Rossana Maria de Castro Andrade, UFC
Joaquim Celestino Júnior, UFC
José Augusto Suruagy Monteiro, UNIFACS
José Neuman de Souza, UFC
Maria Janilce Bosquiroli Almeida, UFRGS
Lisandro Zambenedetti Granville, UFRGS
José Marcos Silva Nogueira, UFMG
Luci Pirmez, UFRJ NCE

Organização Local
Anelise Munaretto, UTFPR
Pedro Torres (POP-PR)
Silvia Avila

Mandato 2005 - 2009
Ana Carolina Salgado (UFPE)
Ricardo de Oliveira Anido (UNICAMP)
Jaime Simão Sichman (USP)
Daniel Schwabe (PUC/RJ)
Marcelo Walter

Suplentes - Mandato 2005-2009
Robert Carlisle Burnett (PUCPR)
Ricardo Reis (UFRGS)
José Valdeni de Lima (UFRGS)
Raul Sidnei Wazlawick (UFSC)

Laboratório Nacional de Redes de Computadores – LARC

Diretor do Conselho Técnico Científico:
Guido Lemos de Souza Filho (UFPB)

Vice-Diretor do Conselho Técnico-Científico:
José Neuman de Souza (UFC)

Diretor executivo:
Luci Pirmez (UFRJ)

Vice-Diretor executivo:
Lisandro Zambenedetti Granville (UFRGS)

Membros Institucionais
MEC-SESU, USP, PUC-RJ, UNICAMP, UFRGS, UFRJ, UFMG, UFPE, INPE, UFPB, LNCC, IME, UFSC, CEFET-PR, UFC, UFF, UFSCAR, CEFET-CE, UFRN, UFES, UFBA, UNIFACS, UECE.

Promoção

Sociedade Brasileira de Computação – SBC

Diretoria

Presidente
Cláudia Maria Bauzer Medeiros (UNICAMP)

Vice-Presidente
José Carlos Maldonado (ICMC - USP)

Diretoria Administrativa e Finanças
Carla Maria Dal Sasso Freitas (UFRGS)

Diretoria de Eventos e Comissões Especiais
Karin Breitmann (PUC-Rio)

Diretoria de Educação
Edson Norberto Cáceres (UFMS)

Diretoria de Publicações
Marta Lima de Queirós Mattoso (UFRJ)

Diretoria de Planejamento e Programas Especiais
Virgílio Augusto Fernandes Almeida (UFMG)

Diretoria de Secretarias Regionais
Aline dos Santos Andrade (UFBA)

Diretoria de Divulgação e Marketing
Altigran Soares da Silva (UFAM)

Diretoria de Regulamentação da Profissão
Roberto da Silva Bigonha (UFMG)

Diretoria de Eventos Especiais
Carlos Eduardo Ferreira (USP)

Conselho

Mandato 2003-2007
Flávio Rech Wagner (UFRGS)
Siang Wu Song (USP)
Luiz Fernando Gomes Soares (PUC/RJ)
Ariadne Maria B. Carvalho (Unicamp)
Taisy Silva Weber (UFRGS)

Primeira - na biblioteca Axis, foi inserido o código da Figura 5, no método "invoke()" da classe org.apache.axis.client.Call, para que a mesma passe a exercer a checagem de controle de acesso, antes de invocar um serviço web;

```
GridSecuriryManager seg = (GridSecuriryManager)System.getSecurityManager();
segcheckWebService(transport.url, operationName.getNamespaceURI() ,
operationName.getLocalPart() );
```

Figura 5. Requisição de controle de acesso à serviços web.Os parâmetros passados para o método " checkWebService" são respectivamente o endereço do serviço web , o serviço e sua porta.

Segunda - na classe "org.globus.secutiry.GridSecuriryManager", além de ter sido implementado o método para controle de acesso a serviços web "checkWebService", foi desabilitado o controle de acesso à conexões tcp/udp para requisições vindas da classe "org.apache.axis.client.Call". A Figura 6 traz o código do método "checkConnect" .

```
public void checkConnect(String host, int porta) {
if (inClass("org.apache.axis.client.Call")) // sem controle para  requisiçoes vindas do axis
.       return  ;
if ( !pegaPermisaoSock(host, porta) )
          throw new SecurityException("Not conect to :" + host + porta);
}
```

Figura 6. Controle de conexões tcp/udp, desabilitado para requisições vindas da biblioteca Axis.

Ao não realizar o controle de acesso a conexões tcp/udp, para requisições vindas da biblioteca Axis, o gerente de segurança permite que o controle de acesso seja realizado somente no nível dos serviços web. Contudo, caso um programa utilize uma biblioteca Axis sem as alterações realizadas, nenhum controle é exercido. No caso da plataforma do implementada a classe "org.globus.secutiry.GridSecuritManagerLoad" é responsável em carregar a biblioteca Axis junto ao diretório de bibliotecas do GT4.

6.3 Gerente de tarefas GRAM Java

O serviço WS GRAM do GT4 conta com *scrips* Perl para preparar o ambiente, iniciar a execução e manter informações sobre o status das tarefas e serviços instanciados remotamente. No intuito de integrar o WS GRAM com o Sand Castle foram realizadas mudança no script "/usr/local/globus-4.0.1/lib/perl/Globus/GRAM/JobManager/fork.pm" para que o mesmo passe a exercer também as seguinte funções:

- a partir do arquivo descritor de tarefas no formato "rsl", extrair a propriedade "project" para definir o perfil de segurança a ser utilizado;
- verificar junto ao arquivo "grid-mapfile", se o usuário requisitor possui permissões de utilizar o perfil definido;
- armazenar as políticas de segurança do perfil determinado em um arquivo local que deverá ser utilizado pelo gerente de segurança do Sand Castle;

- iniciar corretamente a classe “org.globus.secutiry.GridSecuritManagerLoad”, para que a mesma gerencie a execução da tarefa dada.

7. Teste operacional

Para uma melhor compreensão dos benefícios do Sand Castle, esta seção apresenta um teste operacional, onde uma tarefa Java será instanciada remotamente por meio do serviço WS GRAM com suporte a Sand Castle na tentativa executar uma serie de operações mediante a utilização de dois perfis de segurança distintos.

Para a realização do estudo dois perfis de aplicações foram cadastrados junto a uma VO de teste: o “teste.escrita” e o “teste.web”, sendo atribuído ao usuário de teste “testeusu”, a permissão de utilizar ambos. Conforme demonstrado na Figura 7, as políticas de segurança do teste.escrita permitem ler e escrever no diretório “/tmp” e utilizar o serviços tcp/udp no host 192.168.0.77 e na porta 7. Já as políticas para o “teste.web”, representadas pela Figura 8, atribuem autorização para utilizar o serviço web “Teste” na porta “Echo” a partir do endereço “http://192.168.0.39:8080/axis/” .

```
FilePermission /tmp/* read,write
SocketPermission 192.168.0.77 7 resolve,connect
```

Figura 7. Permissões do perfil de aplicações “teste.escrita”

```
WebservicePermission http://192.168.0.39:8080/axis/  Teste  Echo
```

Figura 8. Permissões do perfil de aplicações “teste.web”

Do lado do usuário, foi desenvolvido uma aplicação que tenta escrever um arquivo no diretório “/tmp/”, abrir uma conexão tcp com o endereço “192.168.0.77” na porta 7 e por último utilizar o serviço “Teste” na porta “Echo”, a partir do endereço http://192.168.0.39:8080/axis/. No intuito de testar os diferentes perfis de segurança, foram escritos dois arquivos descritores de tarefas no formato “rsl”. O primeiro (teste.escrita.rsl) representado pela Figura 9, atribui a propriedade “project” como “teste.escrita”. O segundo (teste.web.rsl), arquivo idêntico ao primeiro a não ser pela propriedade “project” atribuída como “teste.web”.

```
<job>
   <executable>java</executable>
    <argument>org.globus.secutiry.TestePermisao</argument>
    <project>teste.escrita</project>
     <fileStageIn> <transfer>
          <sourceUrl>gsiftp://192.168.0.44:2811/home/user/testeGlobus.jar</sourceUrl>
          <destinationUrl>file:///${GLOBUS_USER_HOME}/testeGlobus.jar</destinationUrl>
          </transfer> </fileStageIn>
     </transfer> </fileStageIn>
</job>
```

Figura 9. Arquivo “teste.escrita.rsl” descritor de tarefa.

Como o parâmetro “project” do descritor de tarefas define qual o papel de segurança será utilizado - ao utilizar o programa “globusrun-ws” para enviar as tarefas descritas segundo

os arquivos “teste.escrita.rsl” e “teste.red.rsl” - espera-se obter acessos de segurança diferentes para mesma aplicação, fato que pode ser observado nas Figuras 10 e 11, que demonstram o resultado do envio das duas tarefas.

```
[testeusu@curupira ~]$ globusrun-ws -submit -s -f  teste.escrita.rsl
Delegating user credentials...Done.
Submitting job...Done.
Job ID: uuid:2a2ctyy3e-c19b-11dr-a932-00080dc03d44 Termination time:
04/02/2006 16:38 GMT
Current job state: Active
Current job state: CleanUp-Hold

Inicio do teste de controle de acesso !
Dados escritos no arquivo /tmp/GridCliente.out com sucesso!
Dados enviados a 192.168.0.771:7 com sucesso !
Falha ao utilizar WebService em http://192.168.0.39:8080/axis/services/
Fim do teste de controle de acesso !

Current job state: CleanUp
Current job state: Done
Destroying job...Done.
Cleaning up any delegated credentials...Done.
```

Figura 10. Resultado do envio da tarefa Java utilizando o descritor “teste.escrita.rsl”.

Como pode-se observar ao utilizar o perfil “teste.escrita” a tarefa instancia obteve êxito em escrever no diretório “/tmp/ “ e abrir uma conexão com “192.168.0.77” na porta 7, porém não conseguiu utilizar o serviço web.

```
[leo@curupira ~]$ globusrun-ws -submit -s -f  teste.web.rsl
Delegating user credentials...Done.
Submitting job...Done.
Job ID: uuid:2a2c733e-c19b-11da-a932-00080dc03e66
Termination time: 04/02/2006 16:18 GMT
Current job state: Active
Current job state: CleanUp-Hold

Inicio do teste de controle de acesso !
Falha ao escrever no arquivo:/tmp/GridCliente.out
Falha ao tentar conectar com o host : 192.168.0.771:7
WebService em http://192.168.0.39:8080/axis/services/  utilizado com sucesso!
Fim do teste de controle de acesso !

Current job state: CleanUp
Current job state: Done
Destroying job...Done.
Cleaning up any delegated credentials...Done.
```

Figura 11. Resultado do envio da tarefa Java utilizando o descritor “teste.web.rsl”.

Ao contrário da primeira, a tarefa que utilizou o perfil “teste.web”, esta não obteve autorização para escrever no diretório “/tmp/” e nem para abrir um a conexão tcp, além de conseguir utilizar corretamente o serviço web .

8. Conclusões e trabalhos futuros

O paradigma grades computacionais objetiva um melhor aproveitamento dos recursos computacionais existentes. No entanto o mesmo exige novos requisitos em relação à segurança, particularmente a instanciação remota e controle de tarefas e serviços, que podem representar uma grande vulnerabilidade para o sistema.

A ferramenta Sand Castle, tem como objetivo a instanciação segura de aplicativos (tarefas ou serviços) Java em grades computacionais. Seguindo um modelo onde, atribui-se perfis de segurança as aplicações instanciadas, que são colocadas em uma ferramenta do tipo caixa de areia flexível, que tem a função de abalizar as políticas do perfil utilizado para impedir a realização de ações não permitidas, o que reduz os possíveis danos a serem causados ao sistema por aplicações maliciosas.

Em relação aos trabalhos estudados, o Sand Castle se destaca por combinar a utilização de uma ferramenta tipo caixa de areia flexível, capaz de realizar controle de acesso a serviços web e utilizar a infra-estrutura grade computacional para políticas de segurança. Tal combinação permite com que o Sand Castle monitore e limite as operações das aplicações instanciadas de forma flexível e orientadas por políticas, que são armazenadas, distribuídas e cadastradas de forma única e descentralizada em uma VO.

Futuros trabalhos devem estudar a viabilidade do Sand Castle como ferramenta de segurança, ponderando a respeito de funcionalidades, possíveis vulnerabilidades e desempenho. Outro possível tema para novos estudos é a integração do Sand Castle com as diferentes ferramentas de grade computacionais existentes.

Referências

AXIS, biblioteca Java para serviços web. Disponível em http://ws.apache.org/axis/ acesso em 3/4/2006.

Anderson, D., Cobb, J., Korpela E., Lebofsky, M. (2002). SETI@home: an experiment in public-resource computing. Communications of the ACM, Volume 45 , Número 11.

Bavier, A., Bowman, M., Chun, B., Culler, D., Muir S., Peterson, L., Roscoe, T., Spalink, T. e Wawrzoniak, M. (2004.) Operating System Support for Planetary-Scale Network Services. NSDI'04, San Francisco, USA.

Czajkowski K., Ferguson, D., Foster,I, The WS-Resource Framework, March 5, 2004.

Dodonov, E., Quaini J. e Guardiã H. (2004). GridBox: Securing Hosts from Malicious and Greedy Applications. In *Middleware for Grid Computing 2004*, Toronto.

EuroGrid, projeto de infra-estrutura para grade computacional da Europa. Disponível em http://www.eurogrid.org/ acesso em 12/10/2005.

Entropia, Plataforma de grade computacional. Disponível em http://www.entropia.com/ acesso em 3/4/2006.

Foster, I (2005). Globus Toolkit Version 4: Software for Service-Oriented Systems. In I*FIP International Conference on Network and Parallel Computing*, Springer-Verlag LNCS 3779, pp 2-13.

Herzog A. and Shahmehri A. (2002) Using the Java Sandbox for Resource Control. In *7th Nordic Workshop on Secure IT Systems*. Karlstad, Sweden. Pages: 135-147.

Keahey, K., Ripeanu M. e Doering K. (2003). Dynamic Creation and Management of Runtime Environments in the Grid. In *Workshop on Designing and Building Web Services (GGF 9)*, Chicago, IL.

Lorch, M. e Kafura, D. (2004) The PRIMA Grid Authorization System, Springer Netherlands, Journal of Grid Computing, Volume 2, Número 3.

Nabrzyski, J. and Schopf, J. (2003). Grid Resource Management, Kluwer Publishing.

Pistoia, M., Reller F. and Gupta D. (1999). Java 2 Network Security, Prentice-Hall, 2nd Edition.

RSL, Globus *Resource Specification Language*. Disponível em http://www.globus.org/toolkit/docs/2.4/gram/rsl_spec1.html acesso em 3/4/2006.

SOAP, Simple Object Access Protocol. Disponível em http://www.w3.org/TR/soap/ acesso em 1/2/2006.

Welch V., Siebenlist, F., Foster, I., Bresnahan, J., Czajkowski, K., Gawor, J., Kesselman, C., Meder, S., Pearlman, L., e Tuecke S. (2003). Security for Grid Services. In *Twelfth International Symposium on High Performance Distributed Computing (HPDC-12)*, IEEE Press.

Welch V., Barton, T. Kate Keahey and Siebenlist F. (2005). Attributes, Anonymity, and Access: Shibboleth and Globus Integration to Facilitate Grid Collaboration. In Proceedings of the 4th Annual PKI R&D Workshop.

Integração de Ambientes Heterogêneos de Grades Computacionais

Alessandra C. G. Nascimento[1], Marcio N. Miranda[1,2]

[1]Escola de Eng. Elétrica e de Computação – Universidade Federal de Goiás (UFG)
Pça Universitária S/Nº – Setor Universitário – 74.605-220 – Goiânia, GO -- Brazil

[2]Instituto de Informática – Universidade Federal de Goiás (UFG)
Caixa Postal 131 – 74001-970 – Goiânia, GO – Brazil

alessandragarcia2004@yahoo.com, miranda@inf.ufg.br

Abstract. *Grid computing provides integration of different environments, creating a unified system, where organizations might share, manage and access different resources regardless where they are localized. However, the lack of integration among different grid computing systems does not provide total exploitation of resources. This work investigates and develops an interface for integration of two heterogeneous grid computing environments: Globus and InteGrade. Such integration increases the computational power, transparently. Users will be able to submit jobs through both grid environments and applications will be executed into Globus, or into InteGrade or into both systems.*

Resumo. *A computação em grade integra diferentes ambientes, criando um sistema unificado, onde as organizações podem compartilhar, gerenciar e acessar diferentes recursos, a despeito de sua localização física. Porém, a falta de integração entre as diferentes ferramentas existentes não permite um completo aproveitamento dos recursos ociosos. Este trabalho se propõe a investigar e desenvolver uma interface para a integração de dois ambientes heterogêneos de grades computacionais: Globus e InteGrade. Esta integração aumenta a capacidade computacional de forma transparente para o usuário. Os usuários estarão aptos a submeter suas aplicações a amdos os ambientes, as quais poderão ser executadas no Globus, no InteGrade ou em ambos.*

1. Introdução

Atualmente, no lugar de um único computador pertencente a uma instituição, existem muitas estações de trabalho pertencentes a pessoas físicas. Esta propriedade pessoal distribuída, apesar de conveniente para os seus usuários, é menos eficiente do que a propriedade institucional concentrada em um único computador constantemente ocupado. A maioria das estações pessoais costuma ficar inativa nos períodos em que seu proprietário está ausente, mas, por outro lado o alto custo de um supercomputador torna essa solução cada vez menos viável. As grades computacionais [Foster 2001][Baker 2000][Foster 1997] surgem para integrar estes recursos ociosos e estão se tornando cada vez mais abrangentes. Deixaram de ser um simples método de agregação de recursos amplamente distribuídos em larga escala, com o foco apenas na comunidade de

computação de alto desempenho (grades de alto desempenho), para serem uma infra-estrutura de integração de sistemas orientada a serviço (grades de serviços).

As novas possibilidades de aplicação das grades computacionais incentivaram a convergência das tecnologias em grades computacionais com tecnologias comerciais como os *Web Services* [Foster 1999].Um dos propósitos de emprego das grades computacionais é possibilitar aos usuários de informática o acesso transparente a aplicações e a uma grande quantidade de recursos computacionais heterogêneos e geograficamente distribuídos.

Várias ferramentas têm sido criadas para implementar plataformas de execução de aplicações em grade, entre as quais pode-se destacar o Globus [Globus 2006], o InteGrade [Integrade 2006] , o Condor [Condor 2005], o OurGrid [Ourgrid 2005] , o Legion [Legion 2005] , entre outras.

Apesar de permitir comunicação e troca de informações entre computadores, as tecnologias convencionais que compõem a Internet atual não disponibilizam abordagens integradas que permitam o uso compartilhado e coordenado de recursos pertencentes a várias instituições na realização de computações mais complexas. Neste cenário, é comum encontrar uma grande quantidade de recursos computacionais ociosos ou sub-utilizados convivendo com uma demanda crescente por capacidade computacional por parte de aplicações de alto desempenho (tais como aquelas relacionadas às áreas da saúde, do seqüenciamento genético e da física de alta energia), demanda esta que poderia ser suprida através do aproveitamento desses recursos ociosos. Fornecer uma infra-estrutura que torne isto uma realidade é justamente o objetivo principal da computação em grade.

A computação em grade insere conceitos para a virtualização [Foster 1997][Foster 1999] de recursos computacionais, que é um requisito fundamental para a computação sob demanda (utilização de recursos computacionais ociosos disponíveis) e também permite a formação de organizações virtuais para a realização de projetos temáticos baseados em ambientes de computação distribuída. Organizações virtuais são grupos de atuação em um determinado tema podendo estar geográfica e institucionalmente dispersos.

Com o objetivo de suprir a falta de integração entre os recursos computacionais de várias instituições e integrar ferramentas e sistemas de ambientes de grades computacionais existentes nos centros de pesquisas e universidades, este trabalho propõe uma abordagem de integração de ambientes de grades computacionais heterogêneas, que combina ambientes que trabalham com a arquitetura OGSA *(Open Grid Services Architecture)* [OGSA 2005] e ambientes que trabalham com a tecnologia CORBA [OMG 2003], orientada a objetos. Deseja-se executar aplicações sobre um ambiente de grade computacional heterogêneo de maneira transparente, sem se preocupar com alteração de código fonte e alocação de recursos computacionais.

Este trabalho está organizado da seguinte forma: a seção 2 descreve sucintamente os ambientes de grades computacionais que compõem o estudo de caso

deste trabalho. A seção 3 detalha a arquitetura de integração. Na seção 4 são apresentados os resultados obtidos e na seção 5 são realizados os comentários finais.

2. Ambientes de Grades Computacionais

Uma das características da computação em grade é a sua capacidade de lidar com a heterogeneidade dos equipamentos que a compõem. As grades devem ser capazes de se comunicar entre si. Esta integração é desafiadora, quando se considera que grades diferentes são controladas por organizações independentes, que não compartilham a mesma administração, as mesmas políticas ou as mesmas ferramentas.

2.1 Globus Toolkit

O Globus Toolkit [Foster 2001][Pitanga 2003] é um conjunto de ferramentas e bibliotecas de *software* que dão suporte à arquitetura e às aplicações em grade. É um projeto desenvolvido pelo *Argone National Laboratory* (ANL) e pela *University of Southern California.* Os componentes do conjunto de ferramentas Globus são apresentados na figura 1. Os principais serviços presentes na infra-estrutura são: *Globus Resource Allocation Manager* (GRAM)*, Monitoring and Discovery Service* (MDS)*, Reliable File Transfer* (RFT) *e Grid Security Infrastructure* (GSI)*.*

O GRAM é responsável pela administração de um conjunto de máquinas, além da alocação de recursos e gerenciamento de tarefas. Ele fornece uma interface única que permite submeter, monitorar e controlar tarefas remotamente, de maneira independente do escalonador de recursos. O MDS fornece suporte à organização das informações e sistemas de diretórios da grade. Armazena informações como sistemas operacionais utilizados, memória disponível, espaço em disco, etc. É formado por dois componentes principais: o *Grid Resource Information Service* (GRIS) e o *Grid Index Information Service* (GIIS). O GRIS é responsável por obter as informações da máquina onde está sendo executado. O GIIS é um *framework* para a construção de diretórios customizados, reunindo em um único servidor as informações de vários GRISs. O RFT é o serviço responsável pela transferência segura de arquivos entre as máquinas da grade. O GSI é um serviço de segurança cujo objetivo é permitir que, uma vez autenticado, o usuário receba um a credencial que o permita acessar os recursos sem a necessidade de se autenticar novamente.

O Globus Toolkit 4 (GT4) [Globus 2006] mostrado na figura 2, apresenta uma evolução em relação às versões anteriores. Ele é baseado no conceito de *Grid Services*, uma expansão do conceito de *Web Services*. Os principais conceitos e padrões nos quais está baseado o GT4 são: OGSA, OGSI (*Open Grid Software Infrastructure*) [OGSI 2005] e *Web Services*.

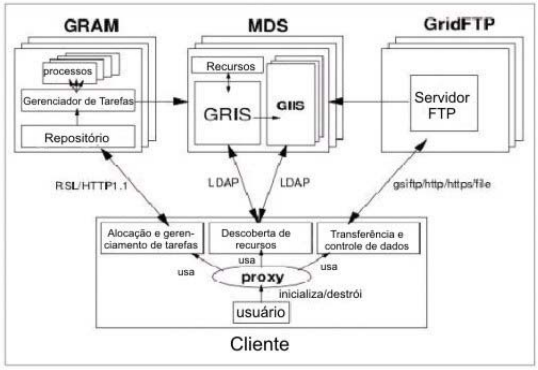

Figura 1 – Componentes do Globus Toolkit

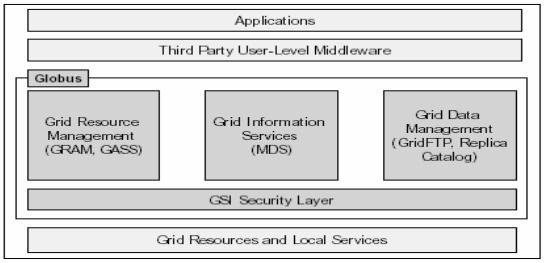

Figura 2 - Arquitetura do Globus Toolkit 4

2.2 InteGrade

O InteGrade [Finger 2003] é um *middleware* desenvolvido pela equipe do IME-USP, que permite a implantação de grades sobre recursos computacionais não dedicados. As principais características do InteGrade incluem: arquitetura orientada a objetos, a utilização do CORBA como infra-estrutura de comunicação, o oferecimento de suporte a aplicações de paralelismo não trivial e a coleta e utilização de padrões de uso das máquinas, de maneira a melhorar o escalonamento A unidade estrutural básica do InteGrade, representado na figura 3, é o aglomerado (*cluster*). Um aglomerado é um conjunto de máquinas agrupadas de acordo com um determinado critério, onde cada uma é chamada de nó. O nó dedicado é uma máquina reservada à computação em grade. O nó compartilhado é aquele pertencente a um usuário que disponibiliza seus recursos ociosos à grade. Já o nó de usuário é aquele que tem capacidade de submeter aplicações para serem executadas na grade. Finalmente, o gerenciador de aglomerado é o nó onde são executados os módulos responsáveis pela coleta de informações e escalonamento, entre outros.

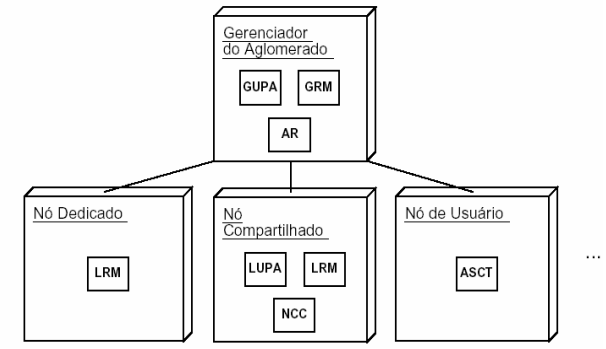

Figura 3 Arquitetura do InteGrade

Os módulos executados nos nós do InteGrade são:

- LRM (*Local Resource Manager*): módulo executado em cada máquina que cede recursos para a grade. Responsável por informar ao GRM a quantidade de recursos disponíveis na máquina, assim como receber requisições para execução de aplicações;

- GRM (*Global Resource Manager*): recebe informações sobre disponibilidade de recursos, enviadas pelos diversos LRM. Também é responsável pelo escalonamento das aplicações submetidas à grade;

- ASCT (*Application Submission and Control Tool*): ferramenta que permite que o usuário da grade registre aplicações no repositório de aplicações (AR), solicite execuções para o GRM, acompanhe o estado da execução e colete seus resultados;

- NCC (*Node Control Center*): módulo executado nas máquinas compartilhadas. Permite que o proprietário da máquina controle o compartilhamento de seus recursos com a grade. Atuando conjuntamente com o LRM, permite que o usuário defina períodos em que os recursos podem ou não ser utilizados, a fração dos recursos que pode ser utilizada ou quando considerar a máquina como ociosa;

- LUPA (*Local Usage Pattern Analyzer*): executado em máquinas compartilhadas, é o responsável pela análise e pelo monitoramento dos padrões de uso. A partir das informações periodicamente coletadas pelo LRM, o LUPA armazena longas séries de dados e aplica algoritmos de *clustering* [Foster 1999][OGSA 2005][OMG 2003], de modo a derivar categorias comportamentais do nó. Tais categorias serão utilizadas como subsídio às decisões de escalonamento, fornecendo uma perspectiva probabilística de maior duração sobre a disponibilidade de recursos em cada nó;

- GUPA (*Global Usage Pattern Analyzer*): auxilia o GRM nas decisões de escalonamento ao fornecer as informações coletadas pelos diversos LUPA. O GUPA pode funcionar de duas maneiras: se as informações fornecidas pelo LUPA não comprometem a privacidade de um usuário, o GUPA pode agir como aglutinador das informações disponibilizadas pelos LUPA. Entretanto, caso o perfil de uso gerado pelo LUPA comprometa a privacidade do proprietário de um recurso, os

padrões de uso nunca deixam o LUPA e o GUPA realizarem consultas específicas ao LUPA, podendo assim funcionar como *cache* das respostas fornecidas sob demanda pelos diversos LUPA;

- AR (*Application Repository*): armazena as aplicações a serem executadas na grade. Através do ASCT, o usuário registra a sua aplicação no repositório para posteriormente requisitar sua execução.

3. Arquitetura de Integração

A metodologia desenvolvida utiliza, como estudo de caso, ambientes de grades computacionais já implantadas e simulações em ambientes reais. Com esta finalidade são utilizadas as ferramentas Globus e InteGrade. O conceito principal é a utilização de um conjunto de padrões e protocolos de código aberto no mercado. Por exemplo, o Globus se baseia na arquitetura *OGSA* que possibilita a comunicação através de ambientes heterogêneos e geograficamente dispersos. Já o InteGrade trabalha com a arquitetura CORBA (*Common Object Request Broker Architecture*) que é uma arquitetura padrão para objetos distribuídos. A arquitetura CORBA define e implementa a estrutura necessária à comunicação entre aplicações distribuídas em diferentes plataformas, sistemas operacionais e linguagens de programação. Com o CORBA, uma aplicação cliente não precisa conhecer os detalhes de implementação do objeto que obterá de um servidor. Esta capacidade é fornecida pela utilização de uma interface comum, compartilhada para a passagem de informações.

Os serviços mais importantes na integração dos ambientes Globus e InteGrade são: segurança, submissão remota de tarefas e gerenciamento das informações e recursos disponíveis na grade computacional. Estes serviços são elementares e essenciais para manter uma grade computacional em funcionamento e servem de base para a construção de *grid services* mais avançados e complexos. As figuras 4 e 5 representam o modelo de integração entre os componentes dos sistemas Globus e InteGrade.

O usuário submete sua aplicação à grade computacional de maneira transparente sem se preocupar com que tipo de sistema irá controlar e gerenciar os recursos capacitados nesta execução. A grade computacional será composta pelos sistemas Globus e InteGrade, conforme mostrado na figura 4.

O usuário poderá submeter as aplicações via Globus pelo comando *globus-run* ou pelo InteGrade utilizando a ferramenta ASCT. O escalonador de aplicações decidirá como distribuir a aplicação na grade de acordo com os parâmetros informados pelo usuário. Como o Globus e o InteGrade possuem arquiteturas e protocolos diferentes, eles não conversam entre si. Para resolver este problema, foi criado um adaptador que funcionará como um *proxy* para transferir as aplicações de um lado para o outro na grade. As informações sobre os recursos disponíveis na grade poderão ser consultadas pelo componente MDS do Globus.

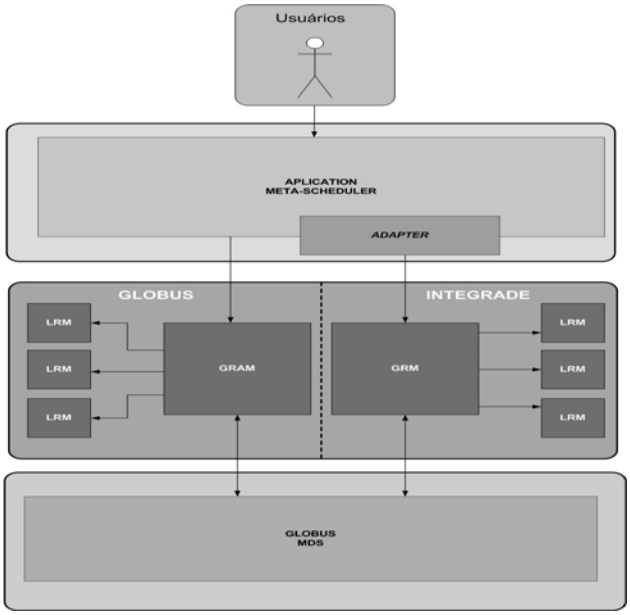

Figura 4 - Representação da arquitetura de integração do Globus com o InteGrade

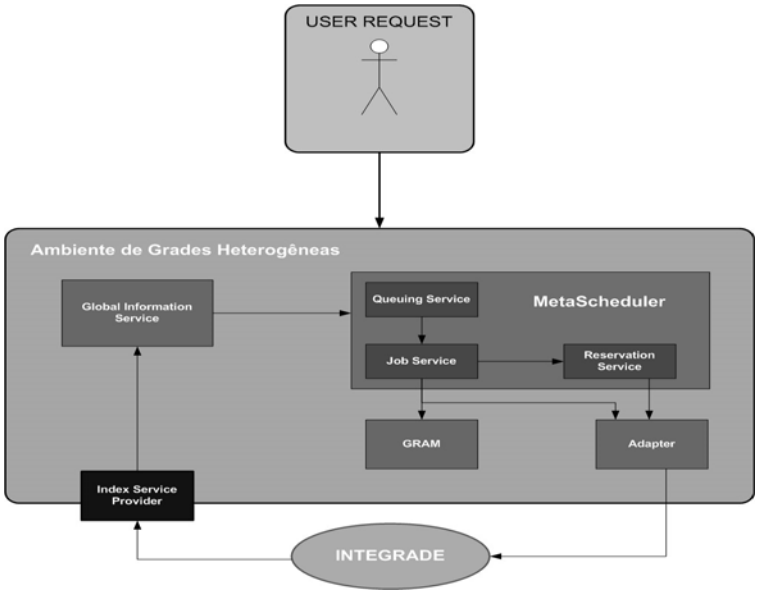

Figura 5 - Esquema de Integração dos componentes do Globus com o InteGrade

3.1 Funcionamento do Sistema de Integração

A integração do Globus e do InteGrade é dividida em 2 partes conforme ilustrado na figura 6:

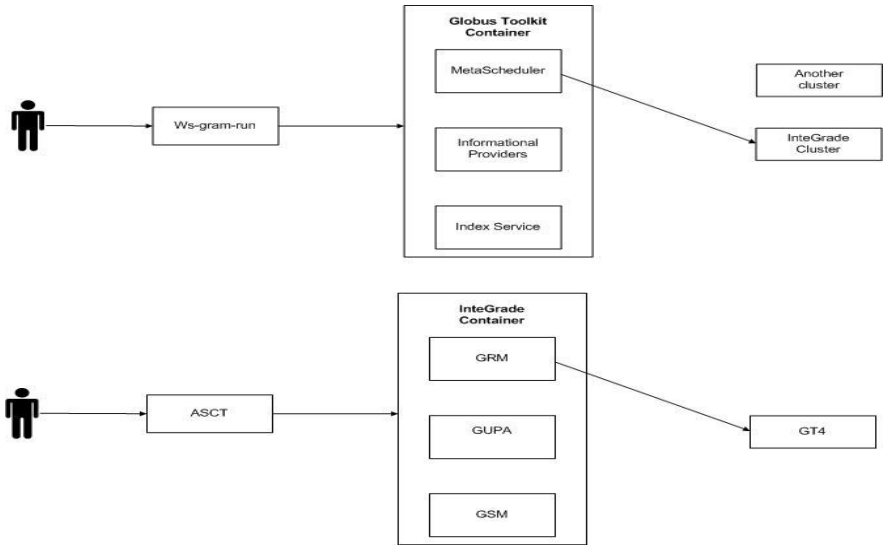

Figura 6 - Modelos de integração do Globus com o InteGrade

Na primeira parte, o usuário da grade computacional Globus submete tarefas para serem processadas na grade InteGrade. O Globus considera a grade InteGrade como um escalonador local, onde o GRM do InteGrade recebe requisições de processamento de tarefas em seus nós LRM.

O GRAM do Globus possui um conjunto de módulos que implementam interfaces para os escalonadores locais de recursos computacionais. Essas interfaces são responsáveis pelo gerenciamento de submissão de tarefas ao escalonador. O GRAM possui interface para vários tipos de escalonadores locais de outras grades computacionais. Esta flexibilidade para aceitar novos escalonadores locais utilizando o conjunto de módulos de interfaces já existentes, permite construir uma interface para o GRM do InteGrade.

A figura 7 mostra a arquitetura do InteGrade como um escalonador externo da grade computacional Globus. Dois novos componentes foram inseridos na arquitetura do Globus Toolkit para habilitar o InteGrade a processar tarefas enviadas pelo Globus. São eles: o InteGrade *adapter,* que é um módulo escrito na linguagem Perl, utilizando-se como referência a interface do Globus para escalonadores locais, e o SGE que permite a monitoração de tarefas enviadas do InteGrade para o Globus.

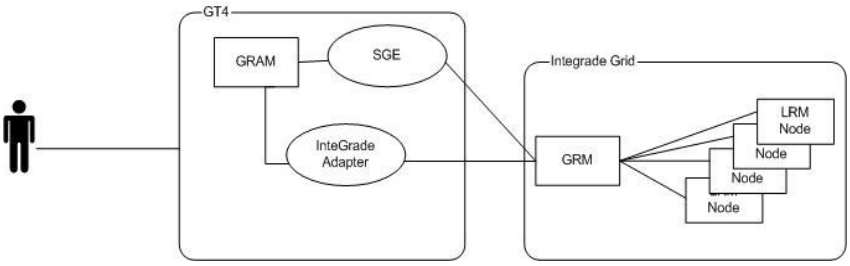

Figura 7 - O InteGrade sendo utilizado como um escalonador externo

Na segunda parte, o usuário do InteGrade submete tarefas para serem processadas no Globus (GT4), conforme ilustrado na figura 8. Para o escalonamento de recursos computacionais disponíveis, o GRM do InteGrade não considera apenas os seus nós LRM, pois o Globus será visto como um nó LRM do InteGrade, capaz de receber tarefas a serem executadas. Com esta finalidade, é criado um novo componente no InteGrade, denominado LRM Globus Node, que funciona como um *proxy*. Sua função será traduzir informações InteGrade em informações Globus. Este novo nó também fornece informações sobre o GT4 ao GRM. Esta abordagem de integração entre os dois sistemas evita mudanças na arquitetura ou nos códigos de seus componentes. Além disto, foi criada uma versão do ASCT que proporciona uma interface de linha de comandos para suprir a integração com o módulo Perl.

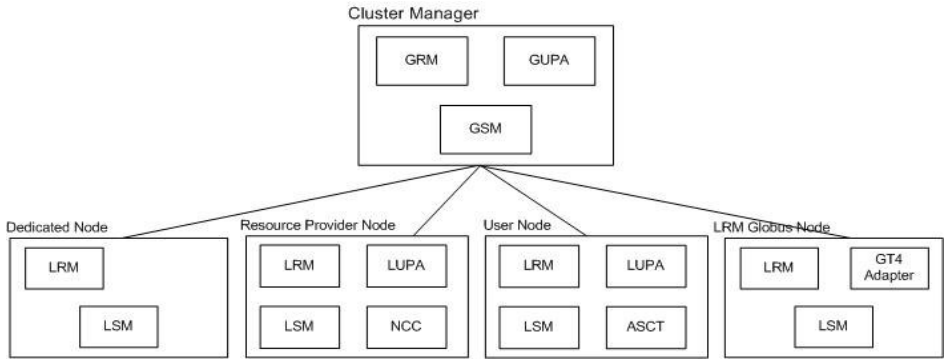

Figura 8 - Globus sendo utilizado como um componente do InteGrade.

4. Resultados

Para a obtenção dos resultados numéricos que são apresentados a seguir, foi desenvolvido um script em PERL do lado do Globus e um módulo em Java do lado do InteGrade. A seguir é realizada uma análise de desempenho das plataformas Globus e InteGrade, utilizando-se como medida de desempenho o tempo de execução da tarefa *fibonacci*(45) em até 4 máquinas. A figura 9 ilustra os resultados obtidos.

4.1 Avaliação do Globus e do InteGrade

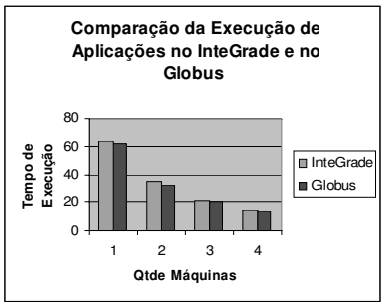

Figura 9: Quadro comparativo do desempenho entre o Globus e o InteGrade

O desempenho do InteGrade é um pouco menor do que o desempenho do Globus porque ele precisa registrar a aplicação no repositório de aplicações e cada LRM tem que ficar buscando o *bytecode* da aplicação sempre que necessitar destas informações. Já o Globus transfere o executável da aplicação por inteiro para cada *host* remoto que irá executar a tarefa.

4.2 Avaliação do Ambiente Integrado

A figura 10 mostra o tempo de execução da mesma aplicação no sistema integrado:

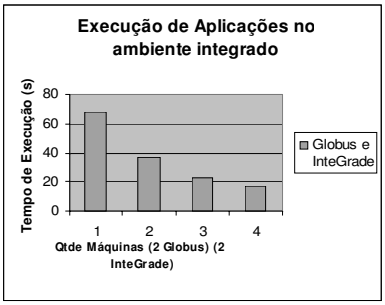

Figura 10: Análise de desempenho da execução da aplicação no ambiente integrado

A execução de aplicação no ambiente integrado tanto pelo lado do Globus quanto pelo lado do InteGrade não altera significativamente os tempos de execução. Existe uma demora maior em relação à execução em cada ambiente separado porque é preciso esperar que tanto o GRAM, do lado Globus, quanto o ASCT, do lado InteGrade, respondam à solicitação de disponibilidade para que executem as tarefas e isto acaba gerando um *overhead*. Isto acontece devido à relativa simplicidade da aplicação usada como exemplo. No entanto, na medida em que as aplicações forem mais complexas e necessitarem de maior quantidade de recursos, a influência deste *overhead* tende a diminuir e o desempenho do sistema integrado tende a aumentar. O *overhead* dependerá do tempo de resposta dos componentes dos sistemas em questão em aceitar a sua invocação para execução de tarefas.

5. Considerações Finais

O objetivo deste trabalho foi desenvolver uma interface para a integração de grades computacionais heterogêneas, utilizando como estudo de caso os ambientes de grade Globus e InteGrade. É definida uma arquitetura para a integração e são implementados módulos específicos, essenciais ao funcionamento dos dois sistemas, permitindo que eles funcionem de forma integrada.

Grande parte do interesse em integrar ambientes de computação em grade advém do potencial de atingir níveis de paralelismo inimagináveis em outras plataformas de execução de aplicações paralelas. Tais níveis de paralelismo podem se traduzir tanto em um desempenho maior para aplicações existentes quanto na possibilidade de executar aplicações inteiramente novas, com gigantescos requisitos de processamento e armazenamento. Dentre as principais motivações deste trabalho, é possível destacar a necessidade e o empenho em criar e desenvolver um ambiente de programação paralela de alto desempenho que possa ser executado de maneira simples, eficiente e transparente em quaisquer máquinas.

Os resultados obtidos a partir da implementação da arquitetura proposta na seção 3 mostram a viabilidade da integração entre as ferramentas Globus e InteGrade e as diversas vantagens obtidas com esta integração, a saber:

- O mapeamento de um novo componente no InteGrade (LRM-GT4) possibilita a criação de um novo módulo para o InteGrade, capaz de entender e interagir com as requisições do Globus. Assim é possível compartilhar recursos computacionais espalhados em outras localidades criando um ambiente colaborativo;

- A possibilidade do InteGrade utilizar os componentes MDS e GSI através da interação com o Globus. Desta forma, o InteGrade poderá usufruir de novas funcionalidades para disseminar e localizar novos recursos computacionais, além de gerenciar certificados e segurança;

- A integração e o intercâmbio de conhecimento entre pesquisadores de diferentes centros de pesquisas, que eventualmente possuam plataformas diferentes.

Como extensões deste trabalho, podem ser citados vários pontos a serem mais profundamente explorados como, por exemplo, a questão do balanceamento de carga, cujo algoritmo pode ser comparado a outros equivalentes e modificado de maneira a torná-lo mais eficiente. Outras extensões podem incluir o acesso remoto a arquivos, a possibilidade de migração de processos, a criação de um portal único de submissão de trabalhos e sistemas de monitoramento avançado de recursos.

Nesse trabalho não foram tratadas as dependências de *hardware,* que são de grande importância na configuração de um sistema computacional. O tratamento das dependências de *hardware* é extremamente necessário, pois sabe-se que, além de existir uma enorme variedade de *hardware* disponível no mercado, novos *hardwares* continuam a surgir e que estamos numa época em que a computação móvel e a

computação ubíqua está sendo cada vez mais estudada e requisitada. O tratamento do hardware será indicado para trabalhos futuros, não sendo tratado nesta integração.

Referências

Globus Project. 2006 Disponível em: <http://www.globus.org> Acesso em: Fevereiro/2006

InteGrade Project. 2006 Disponível em: <http://incubadora.fapesp.br/projects/integrade > Acesso em: Fevereiro/2006

Foster, I., Kesselman, C., Tueke, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of Supercomputing Applications, 15 (3), 200-222, 2001.

Baker, M., Buyya, R., Laforenza, D. The Grid: International Efforts in Global Computing. In Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet, July 31-August 6, 2000.

Foster, I., Kesselman, C. "Globus: A Metacomputing Infrastructure Toolkit", International Journal of Supercomputer Applications, 11(2): 115-128, 1997.

Foster, I. (ed.), Kesselman C (ed.). The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers: San Francisco, 1999.

Open Grid System Architecture. Disponível em: <http://www.globus.org/ogsa> Acesso em: Agosto/2005

OMG. The Common Object Request Broker: Architecture and Specification, rev. 3.0, Object Management Group, Needham-MA, USA, 2003.

Open Grid System Infrastructure. Disponível em: <http://www.globus.org/toolkit/draft-ggf-ogsi-gridservice-33_2003-06-27.pdf >. Acesso em: Agosto/2005

Condor Project. 2005 Disponível em:<http://www.cs.wisc.edu/condor/>. Acesso em: Agosto/2005.

Condor G Project. 2006 Disponível em: <http://www-128.ibm.com/developerworks/grid/library/gr-condorg1/>. Acesso em : Fevereiro/2006.

D-Grid Integration Project. 2006 Disponível em: < http://www.d-grid.de/index.php?id=61&L=1>. Acesso em : Fevereiro/2006.

Ourgrid Project. 2005. Disponível em:<http://www.ourgrid.org>. Acesso em: Agosto/2005

Legion Project. 2005. Disponível em: <http://www.cs.virginia.edu/%7Elegion>. Acesso em: Agosto/2005

Pitanga, M. Introduction to Grid Computing with Globus.
http://www.redbooks.ibm.com/redbooks/pdfs/sg246895.pdf: IBM Redbooks.

Finger, M., Goldchleger, A., Kon, F., Goldman, A., Bezerra, G. C. Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines. Journal of Concurrency and Computation: Practice and Experience, 2003. Vol. 16, pp. 449-459.

Sessão Técnnica 4 /
Technical Session 4

Modelo para Integração de Sistemas de Detecção de Intrusão através de *Grids* Computacionais

Paulo Fernando da Silva, Carlos Becker Westphall, Carla Merkle Westphall

Laboratório de Redes e Gerência (LRG) – Programa de Pós-Graduação em Ciências da Computação (PPGCC) – Universidade Federal de Santa Cataria (UFSC)
Caixa Postal 476 – 88040-970 – Santa Catarina – SC – Brasil

{paulo, westphal, carla}@lrg.ufsc.br

Resumo. *Este artigo apresenta um modelo para integração de IDSs usando Grids computacionais. O modelo proposto possibilita que IDSs heterogêneos trabalhem em conjunto formando um DIDS. São realizadas considerações e simulações sobre o modelo proposto demonstrando seu funcionamento.*

1. Introdução

Frente à diversidade de IDSs (Sistemas de Detecção de Intrusão) disponíveis no mercado, soluções para integração entre estes sistemas têm sido propostas por diversos grupos. Alguns motivos que justificam integração entre IDSs são descritos pelo grupo IDWG (*Intrusion Detection Working Group*) do IETF (*Internet Engineering Task Force*) em [WOOD 2002], entre eles temos: (i) a existência de vários IDSs no mercado com pontos fortes e fracos muito diferentes, tornando necessária a utilização de mais de um IDS em um mesmo ambiente; (ii) o fato de que intrusões geralmente envolvem redes de várias empresas ou várias redes de uma mesma empresa, protegidas por IDSs diferentes. Seria de grande utilidade fazer o relacionamento de um ataque distribuído por várias redes; (iii) a integração entre componentes de diferentes IDSs beneficiaria as pesquisas sobre detecção de intrusão e a migração destas para produtos comerciais.

Para realizar o relacionamento de informações de intrusão originadas em diferentes *hosts*, redes ou domínios, com o intuito de detectar ataques distribuídos, surgem no início da década de 90 os Sistemas de Detecção de Intrusão Distribuídos (*DIDS – Distributed Intrusion Detection Systems*) [SNAPP 1992]. Pesquisas relacionadas aos DIDSs ganharam rápido interesse, pois um IDS centralizado não consegue proteger informações sobre uma infra-estrutura distribuída [ZHANG 2005].

Um DIDS necessita de um alto grau de coordenação para realizar a detecção de ataques distribuídos. Para facilitar o desenvolvimento de ambientes distribuídos e altamente coordenados, surgem os *Grids* computacionais.

Os *Grids* computacionais visam o compartilhamento coordenado de recursos. As plataformas de *Grids* computacionais disponibilizam acesso remoto seguro, gerenciamento de alocação, configuração e consulta de informações de recursos, além de serviços para gerenciamento e transporte de dados entre aplicações [FOSTER 2001].

Segundo [FOSTER 2002], os *Grids* podem ser vistos como um conjunto de serviços agregados definidos pelos recursos que compartilham. Esta orientação a serviços dos *Grids* Computacionais é chamada de OGSA (*Open Grid Service*

Architecture). Na OGSA os serviços são especificados através de interfaces bem definidas, abertas, extensíveis e independentes de plataforma, o que contribui para o desenvolvimento de aplicações interoperáveis.

Este artigo propõe um modelo para integração de IDSs através de *Grids* computacionais. O modelo proposto fará com que IDSs heterogêneos trabalhem de maneira cooperativa, resultando em um DIDS sobre *Grids* (DIDSoG - *Distributed Intrusion Detection System on Grid*). No DIDSoG cada um dos IDSs integrados será visto pelos outros como um recurso, e acessado através dos serviços que disponibiliza.

Várias características apresentadas pelos *Grids* computacionais são convenientes à formação de um DIDSoG. Algumas destas características, descritas por [FOSTER 2002] são: a coordenação de recursos não centralizados; a utilização de protocolos e interfaces padronizadas; e a entrega com qualidade de serviço otimizada.

A orientação a serviços incorporada aos *Grids* (OGSA) permite a definição de interfaces que sejam adaptáveis a diferentes plataformas. Diversas implementações podem ser encapsuladas por uma interface de serviço, esta virtualização permite acesso consistente a recursos através de plataformas heterogêneas [FOSTER 2002].

A virtualização de serviços através de interfaces permite a sua utilização sem o conhecimento de como estes são implementados. Esta característica é de grande conveniência para a integração de IDSs, pois um mesmo serviço pode ser implementado por diferentes IDSs.

As plataformas de *Grids* computacionais podem ser usadas para implementar uma grande variedade de serviços. [FOSTER 2002] cita algumas características de serviços passíveis de implementação sobre *Grids*, entre eles: (i) serviços de gerência de dados distribuídos, que incluem serviços de acesso, replicação, localização e transação; (ii) serviços de *Workflow*, que implementam a execução coordenada de múltiplas aplicações sobre múltiplos recursos; (iii) serviços de auditoria, que têm por objetivo detectar fraudes ou intrusões; (iv) serviços de monitoramento, que implementam a descoberta de sensores em um ambiente distribuído e geram alertas em condições determinadas; (v) serviços de determinação de problemas em ambientes distribuídos, que implementam a correlação de informações de *logs* distribuídos.

As características descritas por [FOSTER 2002] são úteis à implementação do DIDSoG. Um DIDS com tal perfil necessita de serviços que gerenciem a localização e o acesso aos dados distribuídos dos diferentes IDSs. Os serviços de auditoria e monitoramento atendem a necessidades próprias dos DIDSs, como: armazenamento seguro, análise de dados com o objetivo de detectar intrusões, descoberta de sensores distribuídos, e geração de alertas. A correlação de *logs* distribuídos também é relevante à implementação do DIDSoG, pois a detecção de ataques distribuídos depende da correlação das informações de alerta geradas pelos diferentes IDSs que o compõem.

As próximas seções deste artigo estão organizadas da seguinte forma. A Seção 2 apresenta trabalhos correlatos. O modelo proposto é apresentado na Seção 3. A Seção 4 descreve o desenvolvimento e um estudo de caso. Resultados e discussão são apresentados na seção 5. Conclusões e trabalhos futuros na seção 6.

2. Trabalhos Correlatos

[KANNADIGA 2005] apresenta o DIDS DIDMA, que tem como características a flexibilidade, a escalabilidade, a independência de plataformas e a confiabilidade. A arquitetura DIDMA permite a análise distribuída de eventos e é facilmente extensível através do desenvolvimento de novos agentes. Como sugestão de trabalhos futuros, [KANNADIGA 2005] cita a possibilidade de integração com outros IDSs e o desenvolvimento de componentes de segurança.

A facilidade de extensão presente no DIDS DIDMA, bem como a sugestão de integração com outros IDSs estão presentes no DIDSoG. As características de flexibilidade, escalabilidade, independência de plataforma, confiabilidade e a sugestão de componentes de segurança, citadas por [KANNADIGA 2005], são atendidas no DIDSoG pela plataforma de *Grids*.

Técnicas mais eficientes para análise de grandes quantidades de dados em redes de larga escala, aplicadas a um DIDS baseado em *clustering***,** são apresentadas em [ZHANG 2005]. Como trabalhos futuros, [ZHANG 2005] sugere a integração entre IDSs heterogêneos, aumentando a variedade de técnicas de detecção de intrusão em um ambiente. O DIDSoG tem o objetivo de integrar IDSs heterogêneos, enquadrando-se na sugestão de trabalhos futuros de [ZHANG 2005].

[STERNE 2005] apresenta um DIDS de arquitetura hierárquica, onde informações são coletadas, agregadas, correlacionadas e analisadas à medida que sobem na hierarquia. A arquitetura é formada por componentes de monitoramento, correlação, detecção por estatísticas, detecção por assinaturas e respostas. Estes componentes cooperam entre si através dos vários níveis da hierarquia.

A integração realizada pelo DIDSoG também é organizada de maneira hierárquica. Cada IDS integrado ao DIDSoG disponibiliza suas funcionalidades em um nível da hierarquia e requisita funcionalidades de outros IDSs em outros níveis. A hierárquica apresentada por [STERNE 2005] reúne componentes homogêneos, enquanto que a arquitetura hierárquica apresentada pelo DIDSoG integra IDSs heterogêneos.

Propostas aliando *Grids* computacionais a IDSs são apresentadas em [LEU 2005a], [LEU 2005b], [TOLBA 2005] e [CHOON 2003].

[LEU 2005a] e [LEU 2005b] propõem a utilização do Globus Toolkit na detecção de intrusões, especialmente em ataques DoS (*Denial of Service*) e DDoS (*Distributed Denial of Service*), devido ao fato da grande quantidade de dados a serem processados para a detecção destes tipos de ataques. É apresentada uma arquitetura de processamento em duas fases. A primeira fase visa detectar ataques momentâneos, enquanto a segunda fase preocupa-se com ataques crônicos ou de longa duração.

IDSs ou DIDSs tradicionais geralmente são coordenados por um ponto central, característica que os torna mais vulneráveis a ataques. [LEU 2005a] destaca que IDSs desenvolvidos sobre plataformas de *Grids* são menos vulneráveis a ataques, devido à distribuição provida por tais plataformas.

Em [LEU 2005a] e [LEU 2005b] são utilizadas ferramentas para gerar vários tipos de ataques, incluindo TCP, ICMP e UDP flooding, e são apresentados

experimentos e resultados demonstrando vantagens na aplicação de *Grids* computacionais em IDSs.

O presente artigo também propõe o desenvolvimento de um DIDS sobre uma plataforma de *Grid*, porém o DIDS proposto pelo presente trabalho será formado pela integração de IDSs heterogêneos. O DIDS sobre *Grid* apresentado por [LEU 2005a] e [LEU 2005b] não prevê a integração com outros IDSs.

A característica de processamento em fases citada em [LEU 2005a] e [LEU 2005b] também é contemplada pelo DIDSoG, onde é possível a especificação de diversos níveis de processamento formados pela integração de IDSs heterogêneos.

O DIDS GIDA (*Grid Intrusion Detection Architecture*), cujo objetivo é a detecção de intrusões em ambientes de *Grid*, é apresentado em [TOLBA 2005]. Para o desenvolvimento do DIDS GIDA foi utilizado o simulador GridSim. Para simplificação do desenvolvimento [TOLBA 2005] utiliza sistemas de detecção homogêneos em seu DIDS. A possibilidade de aplicar sistemas de detecção heterogêneos em seu DIDS é citada por [TOLBA 2005] como tema de estudos futuros.

Outra proposta de um DIDS para *Grids* é vista em [CHOON 2003], onde é apresentada uma arquitetura para DIDS com tal característica. São também apresentados cenários demonstrando como um DIDS pode agir sobre ambientes de *Grid*.

O DIDSoG não tem por objetivo detectar intrusões especificamente em ambiente de *Grid*, como ocorre em [TOLBA 2005] e [CHOON 2003]. Porém, sendo o DIDSoG um IDS distribuído sobre o *Grid* e com capacidade de integração de IDSs específicos, o mesmo poderia ser organizado de modo a detectar tais intrusões. Poderia-se detectar ataques locais e distribuídos, através da integração de IDSs tradicionais; e ataques específicos de *Grids*, através da integração de IDSs com tal funcionalidade.

3. Modelo de Integração Proposto

O DIDSoG forma uma hierarquia de serviços de detecção de intrusão, esta hierárquica é organizada através de um vetor bidimensional definido por *Escopo:Complexidade*. Os IDSs que formam o DIDSoG podem ser alocados em diferentes níveis de escopo ou complexidade, dependendo de suas funcionalidades, da topologia do ambiente alvo e do resultado desejado.

A Figura 1 apresenta um cenário de funcionamento do DIDSoG formado por diferentes recursos de detecção de intrusão (coleta, agregação, correlação, análise, resposta e gerenciamento) providos por diferentes IDSs. Na Figura 1 pode-se observar o fluxo das informações e o relacionamento entre os níveis de escopo e complexidade.

Informações sobre o ambiente (*host*, rede ou aplicação) são coletadas pelos Sensores localizados nos usuários 1 e 2 do domínio 1. As informações são enviadas para Analisadores simples que atuam sobre informações de apenas um *host* (nível 1:1), e para recursos de agregação e correlação que atuam sobre informações de diferentes *hosts* dentro de um mesmo domínio (nível 2:1).

Dentro do primeiro nível de escopo, os Analisadores simples encaminham as informações para Analisadores mais complexos registrados nos próximos níveis de complexidade (nível 1:N).

Quando um Analisador detecta uma intrusão, este se comunica com recursos de Contra-Medidas e Monitoramento registrados em seu nível de escopo. Um Analisador pode solicitar a um recurso de Contra-Medidas que responda a um ataque detectado. Ou informar a um recurso de Monitoramento sobre um ataque em andamento, para que o administrador do ambiente tome providências.

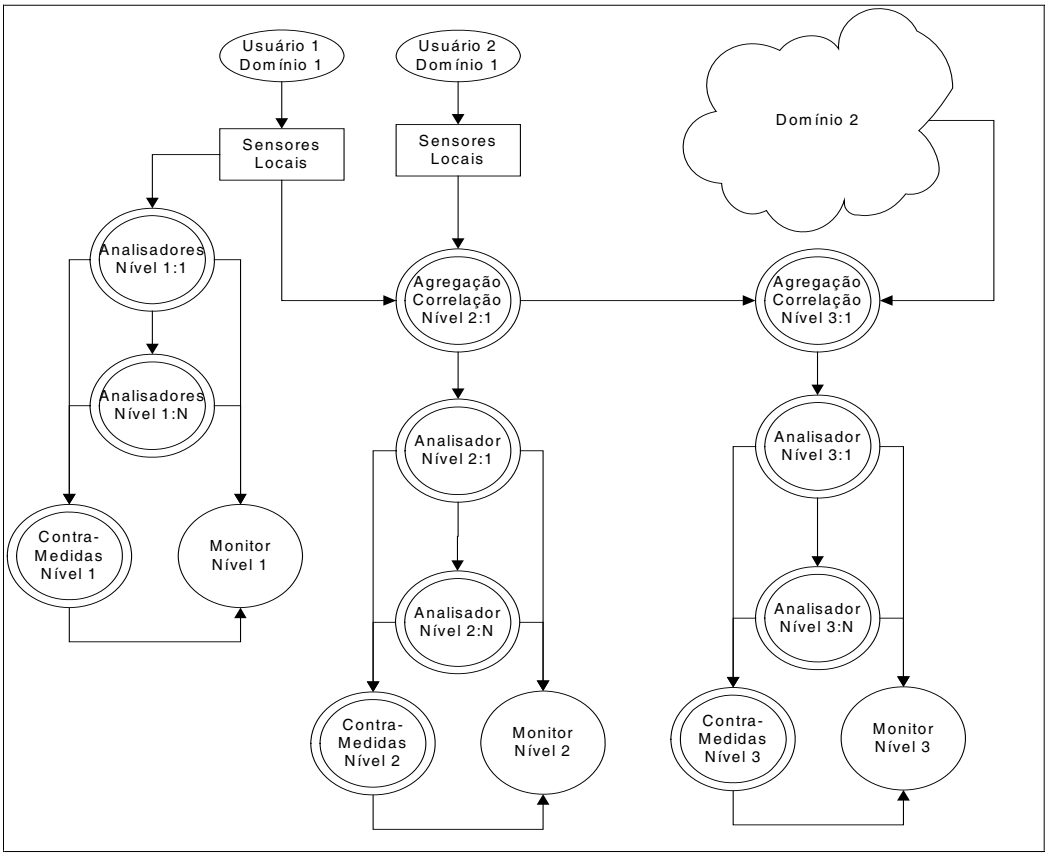

Figura 1 – Cenário de funcionamento do DIDSoG.

Dentro do segundo nível de escopo, recursos de agregação e correlação recebem informações dos Sensores dos diferentes usuários do domínio (usuário 1 e usuário 2). Estes recursos processam as informações recebidas e as encaminham para os recursos de análise registrados no primeiro nível de complexidade dentro do segundo nível de escopo (nível 2:1). As informações também são encaminhadas para os recursos de agregação e correlação registrados no primeiro nível de complexidade dentro do próximo nível de escopo (nível 3:1).

Os recursos de análise do segundo nível de escopo atuam de forma semelhante aos recursos de análise do primeiro nível, encaminhando as informações para recursos de análise mais complexos e acionando recursos de Contra-Medidas e Monitoramento em caso de ataques detectados.

No terceiro nível de escopo, recursos de agregação e correlação recebem informações já agregadas e correlacionadas dentro dos domínios 1 e 2. Estes recursos realizam a agregação e correlação das informações dos diferentes domínios e as enviam para os recursos de análise do primeiro nível de complexidade dentro terceiro nível de

escopo (nível 3:1). As informações também poderiam ser enviadas para agregação no próximo nível de escopo, caso houvessem recursos registrados em tal nível.

Os recursos de análise do terceiro nível de escopo atuam da mesma maneira que os recursos de análise do primeiro e segundo nível, com a diferença de atuarem sobre informações de vários domínios.

As funcionalidades dos recursos registrados em cada um dos níveis de escopo e complexidade podem variar de um ambiente para outro. O modelo permite o desenvolvimento de "N" níveis de escopo e complexidade.

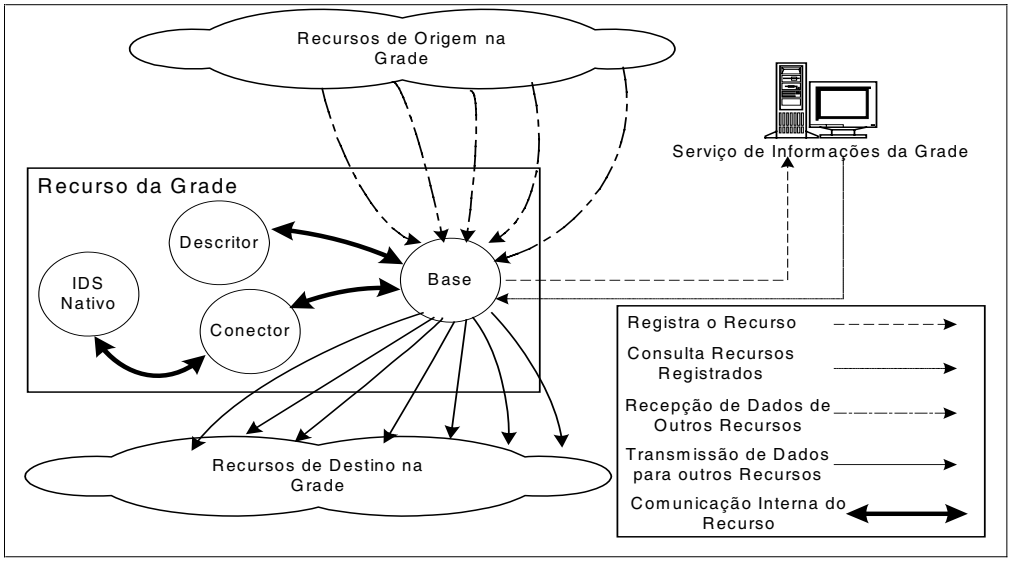

Figura 2 – Arquitetura de um Recurso participante do DIDSoG.

A Figura 2 apresenta a arquitetura de um recurso participante do DIDSoG. Inicialmente um determinado recurso se registra no Serviço de Informações da Grade (GIS – *Grid Information Service*) para que a sua existência seja visível a outros recursos participantes. Após registrar-se, o recurso solicita ao GIS informações sobre outros recursos de detecção de intrusão registrados.

Um determinado recurso do DIDSoG relaciona-se com outros recursos, recebendo dados dos Recursos de Origem, processando-os, e enviando os resultados aos Recursos de Destino, formando uma grade de recursos de detecção de intrusão.

Internamente, um recurso é formado por quatro componentes: Base, Conector, Descritor e IDS Nativo. O IDS Nativo corresponde ao IDS que está sendo integrado ao DIDSoG, este componente realiza o processamento sobre os dados recebidos dos Recursos de Origem, e gera novos dados a serem enviados aos Recursos de Destino.

Um componente IDS Nativo pode ser qualquer ferramenta que realiza algum processamento relacionado à detecção de intrusão, como ferramentas de análise, coleta, agregação, correlação, resposta ou gerenciamento.

O componente Descritor é responsável pelas informações que identificam um recurso e seus respectivos Recursos de Destino no DIDSoG. A Figura 3 apresenta o diagrama de classes das informações armazenadas pelo componente Descritor.

A classe ResourceDescriptor possui as classes Feature, Level, DataType e TargetRecources. A classe Feature representa as funcionalidades que um recurso possui. Os atributos featureType, name e version referem-se as funções oferecidas pelo componente IDS Nativo, seu nome e versão, respectivamente. A classe Level identifica o nível de escopo e complexidade em que o recurso atua. E a classe DataType informa o formato dos dados que o recurso aceita receber. A classe DataType pode ser especializada nas classes Text, XML e Binary. A classe XML contém o atributo DTDFile para especificar o arquivo DTD que valida o XML recebido.

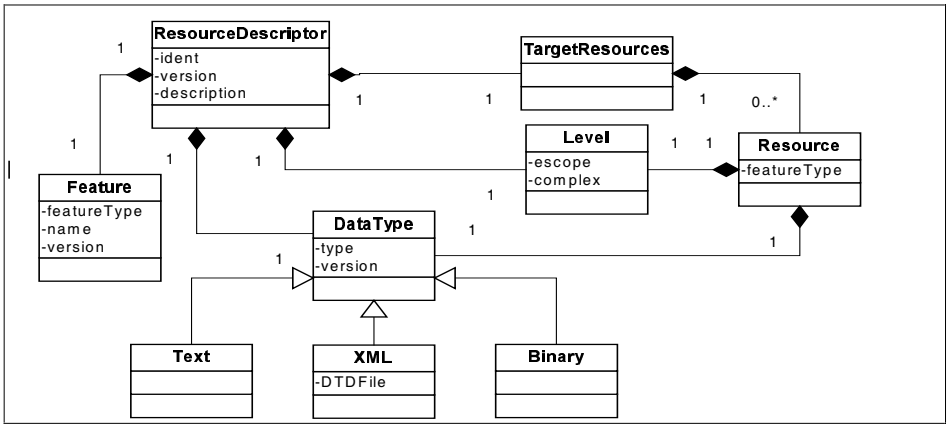

Figura 3 – Diagrama de classes do componente Descritor.

A classe TargetResources informa as características dos Recursos de Destino de um determinado recurso. Esta classe possui classes Resource agregadas a si. A classe Resource identifica as características de um Recurso de Destino. Esta identificação é feita através do atributo featureType e das classes Level e DataType.

Um determinado recurso consulta as informações dos Descritores de outros recursos, e compara estas informações com as especificadas em TargetResources para saber para quais recursos enviar o resultado de seu processamento.

O componente Base é responsável pela comunicação de um recurso com outros recursos do DIDSoG e com o Serviço de Informações da Grade. É este componente que realiza o registro do recurso e a consulta a outros recursos no GIS.

O componente Conector é o ponto de ligação entre o componente Base e o componente IDS Nativo. As informações que o componente Base recebe dos Recursos de Origem são repassadas ao componente Conector. O componente Conector é responsável por fazer as alterações necessárias nos dados para que os mesmos sejam compreendidos pelo componente IDS Nativo, e realizar a entrega destas informações ao mesmo, para processamento.

O componente Conector também tem a responsabilidade de coletar as informações processadas pelo componente IDS Nativo, e fazer as alterações necessárias para que elas possam novamente trafegar pelo DIDSoG. Após estas alterações o componente Conector entrega as informações para o componente Base, que as envia aos Recursos de Destino de acordo com as especificações do componente Descritor.

4. Desenvolvimento e Testes

Para o desenvolvimento e a realização de testes preliminares do modelo proposto foi utilizado o simulador GridSim Toolkit 3.3. A partir das classes do simulador GridSim, foram desenvolvidas classes para simular os recursos do DIDSoG e seus componentes internos.

A classe Simulation_User representa um usuário do DIDSoG, sua função é iniciar o processamento de um recurso Sensor, a partir de onde as informações coletadas serão encaminhadas a outros recursos do DIDSoG.

A classe DIDSoG_BaseResource corresponde ao componente Base da arquitetura de um recurso. DIDSoG_BaseResource relaciona-se com a classe DIDSoG_Descriptor, que representa o componente Descritor. A classe DIDSoG_Descriptor é criada a partir de um arquivo XML que especifica um descritor de recursos, conforme Figura 3.

Para cada IDS Nativo integrado ao DIDSoG, deve ser desenvolvido um componente Conector. O componente Conector é implementado criando-se uma classe derivada da classe DIDSoG_BaseResource. A nova classe criada terá todas as funções da classe DIDSoG_BaseResource (componente Base) e poderá implementar novas funcionalidades de acordo com as necessidades do IDS Nativo em que atua.

No ambiente de simulação foram integrados recursos de coleta de dados, análise, agregação/correlação e geração de respostas. Foram desenvolvidas classes para simular o processamento de cada um dos componentes IDS Nativos associados aos recursos.

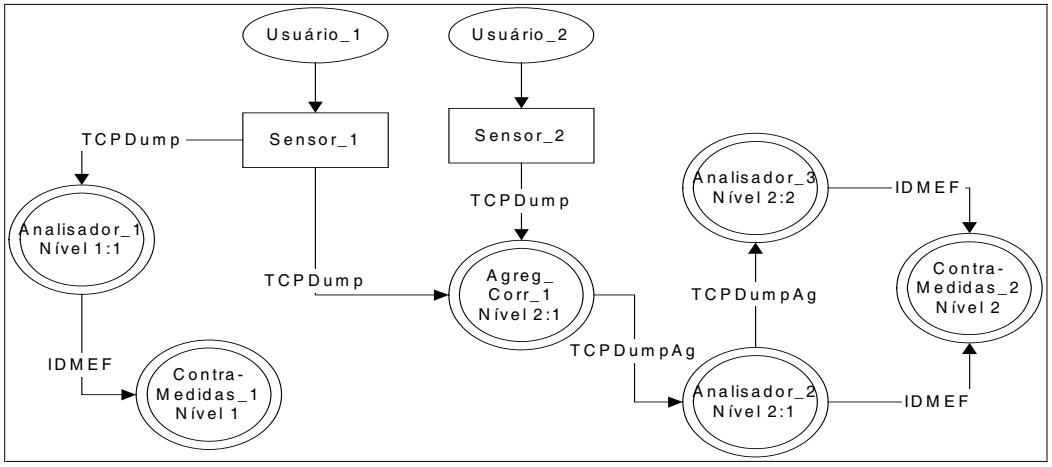

Figura 4 – Fluxo de execução da simulação.

Para cada IDS Nativo simulado, foi desenvolvida uma classe derivada de DIDSoG_BaseResource. Esta classe corresponde ao componente Conector do IDS Nativo e visa integrá-lo ao DIDSoG.

Através do componente Conector, foi selecionado um arquivo descritor XML para cada um dos recursos integrados. O relacionamento resultante entre os recursos integrados ao DIDSoG, de acordo com a especificação de seus respectivos descritores, é apresentado na Figura 4.

Os recursos Sensor_1 e Sensor_2 geram dados simulados no formato TCPDump. De acordo com a especificação de seus descritores os dados gerados são encaminhados aos recursos Analisador_1 e Agreg_Corr_1, no caso de Sensor_1; e Agreg_Corr_1, no caso de Sensor_2.

O IDS Nativo do recurso Analisador_1 foi desenvolvido de forma a gerar alertas para qualquer tentativa de conexão à porta 23. Os dados recebidos por Analisador_1 apresentaram tal característica, fazendo com que o mesmo gerasse um alerta IDMEF. O alerta gerado foi enviado ao recurso Contra-Medidas_1, onde foi emitido um aviso ao administrador sobre o alerta recebido.

O recurso Agreg_Corr_1 recebeu as informações geradas pelos sensores 1 e 2. Seu processamento consiste em realizar o relacionamento entre endereços IP de origem sobre os dados recebidos. As informações resultantes do processamento de Agreg_Corr_1 foram encaminhadas ao recurso Analisador_2.

O componente IDS Nativo do recurso Analisador_2 foi desenvolvido de forma a gerar alertas caso uma origem tente se conectar a uma mesma porta em vários destinos diferentes. Esta situação é identificada pelo Analisador_2 nos dados recebidos de Agreg_Corr_1, e um alerta no formato IDMEF é enviado ao recurso Contra-Medidas_2.

Além de gerar alertas no formato IDMEF, o recurso Analisador_2 também encaminhou os dados recebidos ao Analisador_3, no nível de complexidade 2. O componente IDS Nativo do recurso Analisador_3 gera alertas quando detectado o envio de mensagens ICMP de uma origem específica para vários destinos. Esta situação foi detectada nos dados recebidos de Analisador_2, e um alerta IDMEF foi enviado ao recurso Contra-Medidas_2.

O recurso Contra-Medidas_2 recebeu os alertas gerados pelos analisadores 2 e 3 e, de acordo com a implementação de seu componente IDS Nativo, avisos sobre os alertas recebidos foram emitidos ao administrador.

A simulação realizada exemplificou a arquitetura e o funcionamento do DIDSoG proposto. Foram gerados dados simulados, os quais alimentaram uma grade de detecção de intrusão composta por vários recursos distintos. Os recursos realizaram tarefas de coleta, agregação, análise e geração de respostas de maneira integrada.

5. Resultados e Discussão

O DIDSoG resulta em uma organização hierárquica de funcionalidades de detecção de intrusão organizada através de níveis de escopo e complexidade, formando uma grade de detecção de intrusão. As características de cada recurso na hierarquia são definidas através um descritor.

A organização hierárquica de escopo e complexidade confere ao modelo proposto um alto grau de flexibilidade. A partir de IDSs diferentes, pode-se moldar o DIDSoG de acordo com as necessidades de cada ambiente. Os descritores definem o fluxo de dados que se deseja obter no DIDS resultante.

Cada IDS Nativo é integrado ao DIDSoG através de um componente Conector. O componente Conector representa mais um ponto de flexibilidade do DIDSoG. Adaptações, conversões de tipos de dados e processamentos auxiliares que IDSs Nativos

necessitem são desenvolvidos no Conector. Filtros e geração de *logs* específicos para cada IDS Nativo ou ambiente também podem ser incorporados ao Conector.

Caso se deseje integrar novos IDSs em um ambiente já configurado, basta se desenvolver o Conector para o IDS desejado e se especificar o Descritor do recurso. Após a especificação do Conector e do Descritor, o novo IDS integra-se ao DIDSoG.

Através da definição de níveis de escopo, recursos podem atuar sobre dados de diferentes grupos de origens. Por exemplo, pode-se relacionar o escopo 1 a um determinado conjunto de *hosts*, o escopo 2 a outro determinado conjunto de *hosts*, e o escopo 3 com abrangência sobre os *hosts* dos escopos 1 e 2. Podem ser definidos níveis de escopos de acordo com a necessidade de cada ambiente.

Os níveis de complexidade permitem a distribuição do processamento entre vários recursos dentro de um mesmo escopo. Em uma tarefa de análise, por exemplo, a busca por ataques simples pode ser feita por recursos de complexidade 1, enquanto que a busca por ataques mais complexos, que demanda maior tempo de análise, pode ser feita por recursos de complexidade 2. Com isto, a análise dos dados é feita por dois recursos do ambiente.

A distinção entre níveis de complexidade também pode ser organizada de modo a integrar diferentes técnicas de detecção de intrusão. Pode-se definir o nível de complexidade 1 para análises baseadas em assinaturas, que são técnicas mais simples; o nível de complexidade 2 para técnicas baseadas em comportamento, que requerem maior poder computacional; e o nível de complexidade 3 para detecção de intrusão em aplicações, onde as técnicas são mais específicas e dependem de dados mais detalhados.

Tanto a divisão entre níveis de escopo, quanto em níveis de complexidade faz com que o processamento dos dados seja realizado em fases. Nenhum recurso possui conhecimento sobre o fluxo completo de processamento dos dados. Cada recurso conhece apenas o resultado de seu processamento e o destino a ser enviado tal resultado.

Recursos de maior complexidade devem ser ligados a recursos de menor complexidade. Com isto, mantém-se a estrutura hierárquica do DIDSoG, facilitando a sua expansão e integração com outros domínios de detecção de intrusão.

Realizando um relacionamento hierárquico entre os diversos analisadores escolhidos para um ambiente, o recurso sensor não fica sobrecarregado com a tarefa de enviar os dados a todos os analisadores. Existirá um analisador inicial (complexidade 1) para o qual os sensores enviarão seus dados, e este analisador encaminhará os dados para o próximo passo no fluxo de processamento.

Outra característica da organização hierárquica, é a facilidade de expansão e integração com outros domínios. Se for necessário adicionar um novo *host* (sensor) ao DIDSoG, basta relacioná-lo à primeira hierarquia de recursos. Se for necessário adicionar um novo analisador, que esteja em um escopo sobre vários domínios, basta relacioná-lo a outro recurso de mesmo escopo.

O DIDSoG permite que diferentes níveis sejam gerenciados por diferentes entidades. Por exemplo, o primeiro nível de escopo poderia ser gerenciado pelo usuário local do *host*. O segundo nível de escopo, abrangendo vários *hosts* de um domínio, poderia ser gerenciado pelo administrador do domínio. Uma terceira entidade poderia

ser responsável por gerenciar a segurança de vários domínios de maneira conjunta. Esta entidade atuaria no nível de escopo 3 de maneira independente das outras.

Com o modelo proposto para integração de IDSs em *Grids*, os diferentes IDSs de um ambiente (ou de vários integrados) atuam de maneira cooperativa e coordenada, melhorando o serviço de detecção de intrusão resultante, principalmente sob dois aspectos. Primeiro porque as informações de diferentes origens são analisadas de maneira integrada em busca de ataques distribuídos, e esta integração pode ser feita sob vários níveis de escopo. Segundo porque uma grande diversidade de técnicas de agregação, correlação, análise e resposta podem ser aplicadas a um mesmo ambiente, e estas técnicas podem ser organizadas sob vários níveis de complexidade.

6. Conclusão

Tendo em vista a necessidade de integração entre IDSs heterogêneos e a incompatibilidade das arquiteturas destes IDSs, foi proposto neste artigo um modelo para a formação de um DIDS composto pela integração de diferentes IDSs sobre uma plataforma de *Grids* computacionais (DIDSoG). No DIDSoG os IDSs componentes estão organizados sob a forma de uma grade de serviços de detecção de intrusão.

Cada IDS distinto é visto pelo DIDSoG como um recurso. Uma plataforma de *Grids* computacionais é usada para integração destes recursos. Esta tem a responsabilidade de prover requisitos básicos de comunicação, localização, compartilhamento e segurança necessários ao DIDSoG.

Os componentes da arquitetura do DIDSoG foram desenvolvidos sobre o simulador GridSim. Serviços de comunicação e localização do GridSim foram utilizados para realizar a integração entre componentes de diferentes recursos. A partir dos componentes da arquitetura, foram modelados diversos recursos formando uma grade de detecção de intrusão. A simulação demonstrou o funcionamento do modelo proposto. Foram lidos dados a partir dos recursos sensores e os dados alimentaram outros recursos da grade de serviços de detecção de intrusão.

Através da simulação pôde-se observar a integração de IDSs distintos. Foram integrados recursos com diferentes funcionalidades de detecção de intrusão (análise, correlação, agregação e resposta). Os serviços de comunicação e localização do GridSim, usados durante a simulação, demonstraram-se úteis ao desenvolvimento do DIDSoG, fornecendo a base para o desenvolvimento dos componentes da arquitetura e a posterior integração entre os componentes dos diferentes recursos.

Os componentes da arquitetura DIDSoG serviram de base para a integração dos recursos apresentados na simulação. Tais componentes forneceram as funcionalidades necessárias à integração dos recursos e a conseqüente formação de uma grade de serviços de detecção de intrusão.

Durante a simulação os diferentes IDSs cooperaram entre si de maneira distribuída, porém, com uma visão integrada dos eventos do ambiente, obtendo a capacidade de detectar ataques distribuídos. Esta capacidade demonstra que os IDSs integrados resultaram em um DIDS.

Alguns trabalhos correlatos apresentam características de cooperação entre componentes de DIDSs específicos. Outros apresentam o desenvolvimento de DIDS

sobre *Grids* computacionais ou a aplicação de IDSs aos *Grids* computacionais. Porém, todos trataram do relacionamento entre componentes homogêneos.

O modelo especificado, desenvolvido e simulado neste artigo demonstrou ser uma solução à questão de integração entre IDSs heterogêneos, possibilitando que sistemas de detecção de intrusão com funcionalidades diversas formem um único sistema resultante, coordenado e cooperativo.

O DIDSoG apresentado neste artigo possibilita o desenvolvimento de diversos trabalhos futuros, entre eles sugere-se: desenvolver o modelo em um ambiente real de execução; incorporar a utilização de serviços de segurança; e permitir a análise paralela dos dados por um mesmo IDS Nativo em diferentes *hosts*.

Bibliografia

Choon, O. T.; Samsudim, A. *Grid*-based Intrusion Detection System. The 9th IEEE Asia-Pacific Conference Communications, Setembro 2003.

Foster, I.; Kesselman, C.; Tuecke, S. The Physiology of the *Grid*: An Open *Grid* Service Architecture for Distributed System Integration. Draft Junho 2002. Disponível por http://www.globus.org/research/papers/ogsa.pdf. Acesso Fev. 2006.

Foster, Ian; Kesselman, Carl; Tuecke, Steven. The anatomy of the *Grid*: enabling scalable virtual organizations. International Journal of Supercomputer Applications, vol 5 no. 3, 2001.

Kannadiga, P.; Zulkernine, M. DIDMA: A Distributed Intrusion Detection System Using Mobile Agents. Proceedings of the IEEE Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, Maio 2005.

Leu, Fang-Yie, et al. Integrating *Grid* with Intrusion Detection. Proceedings of 19th IEEE AINA'05, Março 2005.

Leu, Fang-Yie, et al. A Performance-Based *Grid* Intrusion Detection System. Proceedings of the 29th IEEE COMPSAC'05, Julho 2005.

Snapp, S. R. et al. DIDS (Distributed Intrusion Detection System) - Motivation, Architecture and An Early Prototype. Proceeding of the Fifteenth IEEE National Computer Security Conference. Baltimore, MD, Outubro 1992.

Sterne, D.; et al. A General Cooperative Intrusion Detection Architecture for MANETs. Proceedings of the Third IEEE IWIA'05, Março 2005.

Tolba, M. F. ; et al. GIDA: Toward Enabling *Grid* Intrusion Detection Systems. 5th IEEE International Symposium on Cluster Computing and the *Grid*, Maio 2005.

Wood, M. Intrusion Detection message exchange requirements. Draft-ietf-idwg-requirements-10, Outubro de 2002. Disponível por http://www.ietf.org/internet-drafts/draft-ietf-idwg-requirements-10.txt. Acesso em 01 Março 2006.

Zhang, Yu-Fang; Xiong, Z.; Wang, X. Distributed Intrusion Detection Based on Clustering. Proceedings of IEEE International Conference Machine Learning and Cybernetics, Agosto 2005.

ch_hyperbone: Um Dispositivo para Execução de Programas MPI em Hipercubos Virtuais

Samuel L. V. Mello [2], Rafael Caiuta [2], Luis C. E. Bona [1], Elias P. Duarte Jr.[2], Keiko V. O. Fonseca [1]

[1] Universidade Tecnológica Federal do Paraná
CPGEI Av. Sete de Setembro 3165 Curitiba PR
{bona, keiko}@cpgei.cefetpr.br

[2]Universidade Federal do Paraná
Departamento de Informática Caixa Postal 19018 Curitiba PR
{slucas,caiuta,elias}@inf.ufpr.br

Abstract. *Several applications rely on distributed and parallel computing as a powerful tool for solving problems that require large amounts of resources. HyperBone is an overlay network based on virtual hypercubes that abstracts the heterogeneity of connected resources, providing message routing and delivery, and fault management. This paper presents the implementation of a software device that allows the execution of parallel programs based on the MPI standard on top of the HyperBone platform. Experimental results show the functionality of the device, the impact caused by the use of the hypercube topology on the throughput and evaluates the performance of an image rendering application.*

Resumo. *A computação paralela e distribuída é uma ferramenta poderosa para resolver problemas computacionais que demandam grandes quantidades de recursos computacionais. O HyperBone é uma rede overlay baseada em hipercubos virtuais que provê roteamento de mensagens, monitoramento de falhas e abstração da heterogeneidade dos recursos. Este trabalho apresenta a implementação de um dispositivo de software que permite a execução de programas paralelos e distribuídos baseados no padrão MPI sobre a plataforma HyperBone. São apresentados também resultados experimentais mostrando a funcionalidade do dispositivo, o impacto do uso da topologia de hipercubo na vazão atingida e no desempenho de uma aplicação de renderização de imagens.*

1. Introdução

O uso de conjuntos de processadores independentes interligados em rede executando programas paralelos e distribuídos é uma importante abordagem para a resolução de problemas que demandam grandes quantidades de recursos computacionais. Como exemplos de problemas que podem ser resolvidos usando esta abordagem podem ser destacados a computação científica, simulações dos mais diversos tipos, análise de genoma, renderização de imagens, processamento de sinais, entre outras [Foster and Kesselman 2003].

A Internet permitiu interconectar um grande número de recursos computacionais. Formar sistemas distribuídos de grande porte usando esses recursos é um desafio, os nodos não são permanentes, integrando-se e deixando continuamente o sistema, que está sujeito a falhas e recuperações também contínuas. Uma abordagem promissora para construir esses sistemas distribuídos de larga escala na Internet são as redes overlay [Amir and Danilov 2003]. As redes overlay propõem a criação de uma estrutura que

ofereça serviços como manutenção de rede, monitoração, segurança e alta disponibilidade que dificilmente poderiam ser providos diretamente pelo nível de rede.

As redes overlay podem utilizar diversas topologias para estruturarem seus nodoss. A topologia de hipercubos apresenta algumas características que são interessantes no processamento paralelo e distribuído como escalabilidade, simetria, diâmetro logarítmico, entre outras. A plataforma HyperBone [Bona et al. 2005] é uma rede overlay baseada em hipercubos virtuais, construídos utilizando recursos computacionais interligados em rede.

O HyperBone provê roteamento de mensagens no hipercubo, monitoramento de falhas e permite determinar quais os nodos mais estáveis ou com melhores características para realizar uma tarefa. O HyperBone é responsável por detectar falhas em nodos e enlaces e, em caso de falha, reconfigurar o sistema de modo a manter o hipercubo virtual. Estas características são importantes para permitir a criação de aplicações que utilizem grandes quantidades de recursos computacionais heterogêneos conectados pela Internet.

O MPI *(Message Passing Interface)* [Forum 1994] é um padrão que permite o desenvolvimento de programas paralelos e distribuídos. Ele foi criado em 1993 com o objetivo de ser um padrão amplamente usado para o desenvolvimento de programas baseados no paradigma de comunicação de troca de mensagens. O MPI foi especificado visando a portabilidade, eficiência e flexibilidade. Ele provê ao programador um conjunto de estruturas de dados e funções que abstraem aspectos de baixo nível da comunicação, permitindo que o programador concentre-se no problema a ser resolvido.

A biblioteca MPICH [Gropp et al. 1996] é uma das implementações mais populares do padrão MPI. Ela utiliza dispositivos chamados ADI *(Abstract Device Interface)* [Gropp and Lusk 1994] para abstrair o transporte das mensagens. Essa abstração tem como objetivo facilitar a adaptação da biblioteca para permitir a execução de programas MPI sobre novos dispositivos.

Este trabalho apresenta a implementação de um dispositivo ADI, *ch_hyperbone*, que permite a execução de programas MPI utilizando a plataforma de hipercubos virtuais HyperBone. O objetivo é avaliar o custo do *ch_hyperbone* e assim verificar a viabilidade de executar aplicações paralelas em MPI, usando nodos da Internet executando o HyperBone. Apresenta também os resultados obtidos em três experimentos realizados. O primeiro demonstra a funcionalidade do dispositivo implementado através da execução de uma versão paralela do algoritmo de ordenação *mergesort*. O segundo apresenta a vazão atingida com o dispositivo implementado, comparando-a com a vazão atingida pelo dispositivo *ch_p4*, instalado por padrão com a biblioteca MPICH. O terceiro experimento apresenta o desempenho do dispositivo para execução do MPI-PovRay, uma aplicação de renderização de imagens, comparando com o desempenho atingido utilizando o dispositivo *ch_p4*.

O restante deste trabalho está organizado da seguinte maneira. A seção 2 apresenta o padrão MPI e a arquitetura da biblioteca MPICH; a seção 3 apresenta a plataforma de hipercubos virtuais HyperBone, a seção 4 detalha a implementação do dispositivo que permite a execução de programas MPI sobre o HyperBone e a seção 5 mostra os resultados experimentais obtidos. Na última seção é apresentada a conclusão do trabalho.

2. Programação Paralela e Distribuída com MPI

O padrão MPI (*Message Passing Interface*) provê um conjunto de estruturas de dados e funções que permitem o desenvolvimento de programas paralelos baseados no paradigma de troca de mensagens [Pacheco 1996]. Ele provê abstração de aspectos de baixo nível

da comunicação que permitem que o programador concentre seus esforços no problema a ser resolvido.

O padrão define modos de comunicação ponto-a-ponto (“Envie mensagem para um determinado nodo”, “Receba uma mensagem de algum nodo”), e coletivos (“Envie uma mensagem para todos os nodos”, “Receba mensagem de todos os nodos”, “Envie uma série de mensagens para um grupo de nodos”). Cada mensagem possui uma “etiqueta” (*tag*) que pode ser usada pelo usuário para identificação. O padrão especifica também formas de criar *comunidades*, que definem a que grupo de processos as operações coletivas se referem.

O MPI define também uma forma de sincronização de processos através da construção de barreiras, em que os nodos só avançam na computação depois que todos os nodos atingirem a barreira.

O padrão MPI define duas formas básicas de comunicação entre máquinas: comunicação bloqueante e comunicação não-bloqueante. A diferença básica entre as duas formas é que a comunicação bloqueante espera até que a operação seja completada para que o fluxo do programa prossiga, enquanto a comunicação não-bloqueante permite que a computação prossiga assim que os dados forem transferidos para um buffer a partir de onde possam ser transmitidos posteriormente. A flexibilidade de poder escolher o tipo de comunicação é importante para permitir o desenvolvimento de aplicações de alto desempenho.

Para que seja possível implementar a comunicação não-bloqueante é necessário um buffer. O padrão prevê que o programador pode manipular os objetos dentro do buffer diretamente, utilizando funções específicas para isso. Neste buffer os objetos são armazenados utilizando tipos próprios do MPI, independente da arquitetura ou da linguagem utilizada.

O uso de tipos próprios ao invés de tipos fornecidos pela linguagem de programação ajuda a aumentar a portabilidade já que, por exemplo, um número de ponto flutuante é armazenado no buffer em uma representação independente de plataforma. Este número pode então ser transportado para outro nodo da rede e quando for retirado do buffer será feita a conversão para a nova representação de ponto flutuante, se necessário. O padrão MPI define também funções para a criação de novos tipos baseados no agrupamento de tipos primitivos, permitindo a modelagem de estruturas de dados mais complexas.

Entre as diversas implementações do padrão MPI podemos destacar o MPICH [Gropp et al. 1996], descrito a seguir.

2.1. A Biblioteca MPICH

Uma das implementações do padrão MPI mais usadas é a biblioteca MPICH [Gropp et al. 1996]. Ela implementa as funções da versão 1.1 do MPI, está disponível como software livre para diversas plataformas e suporta diversas formas de transporte das mensagens.

O MPICH foi implementado utilizando uma estrutura em camadas. A camada mais próxima do dispositivo de hardware que fará o transporte das mensagens é o ADI *(Abstract Device Interface)* [Gropp and Lusk 1994]. O ADI é responsável por implementar as funções que vão efetivamente fazer a comunicação entre os nodos e o transporte das mensagens.

Os principais objetivos de utilizar-se uma camada separada para realizar essas operações são a facilidade obtida para portar a biblioteca para novos dispositivos e a pos-

sibilidade de ganho de desempenho, escrevendo-se uma implementação das funções do ADI específica para um determinado dispositivo. O ADI fornece um grande conjunto de funções que podem ser implementadas, permitindo a otimização de uma série de aspectos específicos. A implementação das funções do ADI é chamada de dispositivo. O pacote do MPICH já vem com alguns dispositivos, como o *ch_shmem* para memória compartilhada, o dispositivo *ch_p4* para redes TCP/IP (*Transmission Control Protocol/Internet Protocol*) e o dispositivo *ch_globus2* para grades computacionais construídas com o Globus.

As mensagens enviadas e recebidas pelas funções do ADI não são apenas as solicitadas explicitamente pelo usuário. Por exemplo, quando o programa do usuário executa uma operação coletiva como um *broadcast*, o ADI realizará uma chamada de comunicação ponto-a-ponto para cada nodo alvo do *broadcast*. Outro exemplo: se o programa do usuário atingir uma barreira são geradas mensagens de controle que serão trocadas entre os nodos para realizar a sincronização dos processos.

Porém, muitas vezes deseja-se implementar apenas um pequeno subconjunto dessas funções. O *Channel Interface* [Gropp et al. 1996] permite a criação de dispositivos que implementam apenas certas funções primitivas, usando macros para mapear as demais funções para as primitivas implementadas. Por exemplo, um dispositivo que não implemente comunicação não-bloqueante utilizará as funções de comunicação bloqueante para qualquer comunicação. Então as funções de comunicação não-bloqueante serão mapeadas para as de comunicação bloqueante.

3. HyperBone: Computação em Hipercubos Virtuais

O HyperBone é uma rede overlay que tem como topologia o hipercubo virtual [Bona et al. 2005]. A topologia de hipercubos apresenta algumas características que são interessantes no processamento paralelo distribuído como escalabilidade, simetria e diâmetro logarítmico, entre outras.

O hipercubo virtual é composto por um conjunto de nodos que formam uma rede virtual conectada por enlaces virtuais. Quando todos os nodos da rede virtual estão interligados ocorre a formação de um hipercubo virtual onde os nodos são conectados por conexões TCP (Transmission Control Protocol) persistentes. O HyperBone é responsável por fazer o roteamento das mensagens, detectar possíveis falhas em nodos e enlaces e, em caso de falha, reconfigurar o sistema de modo a manter o serviço disponível.

Por utilizar a topologia de hipercubo virtual, a principal característica do HyperBone é a escalabilidade, o hipercubo é uma estrutura escalável por definição, que apresenta características topológicas importantes como simetria, diâmetro logarítmico e boas propriedades de tolerância a falhas [Bona et al. 2005]. Ele oferece serviços de monitoração que permitem ao usuário determinar quais os nodoss mais estáveis e com melhor desempenho para a execução de determinado processamento. A plataforma também tunela o tráfego das aplicações através dos enlaces virtuais, o que permite a instalação de nodos atrás de firewalls restritivos. Estas características possibilitam a criação de hipercubos com grande quantidade de máquinas conectadas à Internet.

A figura 1 representa a arquitetura do sistema completo. A primeira camada representa os programas de aplicação do usuário que vão realizar a computação desejada utilizando o padrão MPI. Esses programas fazem chamadas às funções da biblioteca MPICH, representada pela segunda camada. Essa camada transforma as chamadas do usuário em mensagens que serão enviadas. Por exemplo, se o programa do usuário cria uma barreira esta camada cria mensagens ponto-a-ponto para sincronizar os nodos atendendo à barreira. O transporte das mensagens entre os nodos é responsabilidade da terceira camada,

ADI: *ch_hyperbone*. Ela encontra a melhor rota para a mensagem no hipercubo e transmite a mensagem. O transporte da mensagem é feito através das funções do HyperBone que utiliza conexões TCP persistentes, que formam a última camada.

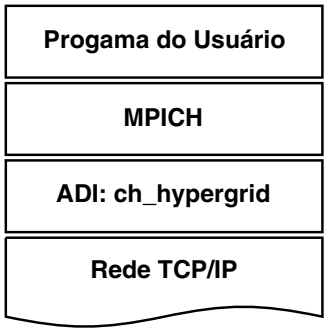

Figura 1: Arquitetura do sistema baseado em MPICH.

O monitoramento e a manutenção do hipercubo virtual são feitos pelo algoritmo DiVHA (*Distributed Virtual Hipercube Algorithm*). Quando um nodo falha, o algoritmo pode alocar outro nodo para substituir o nodo falho. Isto é possível porque no HyperBone os nodos também são virtuais, até mesmo no caso de indisponibilidade de um outro nodo físico, um nodo do hipercubo pode assumir dois ou mais nodos virtuais na estrutura.

Para que seja possível executar programas MPI utilizando a estrutura de hipercubo virtual do HyperBone, foi implementado um dispositivo ADI. Ele recebe as mensagens geradas pelo programa MPI e utiliza o HyperBone para fazer o transporte até o destinatário.

4. *ch_hyperbone*: Um Dispositivo MPICH para Hipercubos Virtuais

A implementação do dispositivo utilizou a versão 1.2.5.2 do MPICH em ambiente GNU/Linux e permite executar programas MPI sobre a estrutura do HyperBone.

Durante a execução de um programa MPI usando a plataforma de hipercubos virtuais é necessária a execução de tarefas internas como o monitoramento dos nodos e o roteamento de informações através do hipercubo. Essas tarefas são executadas por uma thread separada que utiliza o algoritmo DiVHA [Bona et al. 2005] para manter informações sobre os nodos e enlaces e fornecer informações de roteamento no hipercubo.

A lista de nodos participantes da computação é lida de um arquivo de configuração durante a inicialização. Esta lista é usada para iniciar o monitoramento dos nodos e estabelecer os enlaces virtuais. Cada nodo recebe seu número identificador na computação como parâmetro do programa. Esse parâmetro é acrescentado pelo *script* que faz o disparo dos processos e é removido pela função de inicialização de forma que o programa de aplicação do usuário recebe os mesmos parâmetros fornecidos pelo usuário.

Desta forma, o HyperBone cria a estrutura necessária para a troca de mensagens através do hipercubo e inicia o monitoramento dos nodos. O dispositivo desenvolvido utiliza essas funcionalidades fornecidas pelo HyperBone para implementar as funções definidas pelo ADI, possibilitando a execução de programas MPI. A implementação foi baseada em *Channel Interface* que permite a criação do dispositivo implementar um conjunto de funções obtendo as demais funcionalidades do MPI utilizando as funções implementadas. As funções implementadas foram:

- `MPID_Init` responsável pela inicialização do sistema. Quando essa função retornar, todos os nodos deverão estar inicializados e conhecer os demais.

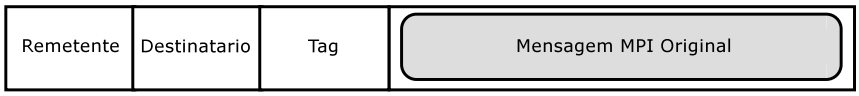

Figura 2: Estrutura da mensagem transmitida.

- `MPID_Send` envia uma mensagem de modo bloqueante. É invocada quando mensagens precisam ser enviadas para um determinado nodo. Essas mensagens são geradas direta ou indiretamente pelo programa do usuário.
- `MPID_Recv` entrega para o programa do usuário uma mensagem que tenha sido recebida da rede. Caso a mensagem esperada ainda não tenha chegado da rede a função fica bloqueada até que esta chegue.
- `MPID_Probe` retorna se existem ou não mensagens recebidas da rede não entregues para o usuário.
- `MPID_from` retorna o remetente da ultima mensagem recebida.
- `MPID_count` retorna o número total de nodos na rede.

A função `MPID_Init` é responsável pela inicialização dos nodos. Ela lê a lista de nodos do arquivo de configuração, inicia a thread de monitoramento do HyperBone e prepara o ambiente para a troca de mensagens.

As funções `MPID_Send` e `MPID_Recv` são responsáveis por enviar e receber mensagens. O transporte das mensagens é feito utilizando funções de envio e recebimento fornecidas pelo HyperBone. No envio da mensagem é adicionado um cabeçalho formado por três campos de quatro bytes cada: o remetente, o destinatário e um *tag* recebido do MPI junto com a mensagem. A figura 2 mostra a estrutura da mensagem transmitida. É necessário transportar estas informações junto com o conteúdo da mensagem porque as mensagens serão roteadas através do hipercubo, isto é, o nodo de onde se recebe a mensagem não é necessariamente o remetente nem o nodo que recebe a mensagem é sempre o destinatário.

A função de envio de mensagens adiciona esse cabeçalho, determina o próximo nodo para se chegar ao destinatário, utilizando a função de roteamento fornecida pelo HyperBone e envia a mensagem. Quando o outro nodo recebe, ele verifica se é o destinatário da mensagem ou se deverá enviar a mensagem para o próximo nodo no caminho até o destinatário. Esse próximo nodo é determinado pela função de roteamento. Caso ele seja o destinatário, a mensagem é colocada em um buffer de recebimento.

O buffer de recebimento é necessário para armazenar as mensagens entre o momento em que são recebidas da rede e o momento em que o programa de aplicação do usuário solicita o recebimento através do ADI. Quando a mensagem é recebida da rede pelo HyperBone ela é armazenada neste buffer, onde aguardará até que o ADI solicite essa mensagem através da função de recebimento de mensagem, que a retira desse buffer. Quando o ADI solicita uma mensagem ele informa o *tag* da mensagem esperada. Assim, a ordem em que as mensagens sairão do buffer pode ser diferente da ordem em que entraram. Por causa disso, foi utilizada uma lista encadeada para armazenar o buffer. Para evitar a alocação de memória para cada mensagem que entra no buffer e a liberação desta memória quando a mensagem é retirada, um conjunto de posições de memórias é alocado durante a inicialização. Quando uma mensagem é recebida, ela é colocada em uma posição livre desse conjunto e após ser retirada do buffer a posição usada é devolvida ao conjunto. O uso de posições de memória pré-alocadas ao invés de chamadas ao sistema operacional para alocar e desalocar memória resultou em ganho de desempenho.

As mensagens são escritas no buffer por uma thread criada para o atendimento da

conexão, enquanto a leitura das mensagens é feita pela thread principal do programa. Por isso, foi necessário garantir a exclusão mútua nas funções que manipulam o buffer. Como o recebimento de mensagens é bloqueante, foi utilizado um semáforo para que a função somente procure pela mensagem no buffer quando alguma mensagem nova for recebida.

5. Validação Experimental

Esta sessão apresenta os resultados dos testes preliminares realizados com o dispositivo implementado em um único cluster. O objetivo destes experimentos foi validar o *ch_hyperbone*, demonstrando a viabilidade da execução de programas paralelos e distribuídos utilizando padrão MPI sobre uma rede overlay baseada em hipercubos virtuais. Os testes foram realizados no cluster da Universidade Federal do Paraná, composto por 20 máquinas com processadores AMD Athlon executando sistema operacional Debian GNU/Linux ligadas com rede Gigabit Ethernet.

Foram realizados três experimentos. O primeiro experimento demonstra a funcionalidade do dispositivo implementado através da execução de uma versão paralela para hipercubos do algoritmo de ordenação mergesort. O segundo experimento realizado teve como objetivo medir a vazão atingida com o dispositivo implementado, compará-la com a vazão atingida com o *ch_p4*, o dispositivo instalado por padrão com o MPICH, e medir o impacto causado na vazão pelo roteamento das mensagens dentro do hipercubo virtual. O terceiro experimento teve como objetivo comparar o desempenho do dispositivo implementado com o *ch_p4* para a execução do MPI-PovRay, uma aplicação típica de renderização de imagens.

5.1. Mergesort Paralelo para Hipercubos

O primeiro experimento teve como objetivo demonstrar a funcionalidade do dispositivo através da execução do algoritmo de ordenação mergesort distribuído para hipercubos [Foster 1995]. Ao iniciar, cada nodo sorteia um número aleatório e no final da execução o nodo de identificador 0 conterá um vetor com os números sorteados por todos os nodos de forma ordenada. A cada iteração, cada nodo envia ou recebe os seus números para um vizinho.

Para tornar o experimento menos suscetível a interferências de fatores como a ordem de escalonamento das threads pelo sistema operacional o programa foi alterado de forma a fazer uso intenso da rede. Assim, em cada rodada cada nodo envia ou recebe seus números diversas vezes. Foi também criada uma barreira no inicio do programa, antes da tomada do tempo inicial, com o objetivo de desconsiderar a diferença de tempo de inicialização entre os nodos.

O gráfico da figura 3 mostra os tempos de execução real, de usuário e de sistema para o algoritmo com 2, 4, 8 e 16 nodos. A linha pontilhada na parte de baixo do gráfico representa o tempo de processamento gasto pelo aplicativo, a linha do meio o tempo de processamento gasto com chamadas ao sistema operacional e a linha superior representa o tempo real de execução. Como a quantidade de números ordenados é igual ao número de nodos integrantes da computação, a quantidade de trabalho realizado é diferente em cada caso. Por isso esses dados não devem ser interpretados como comparação de desempenho.

5.2. Medição da Vazão

O uso de uma rede overlay baseada em hipercubos virtuais gera um *overhead* devido à necessidade de roteamento das mensagens através dos enlaces virtuais e pela camada de software que implementa o HyperBone. O segundo experimento mede a vazão atingida

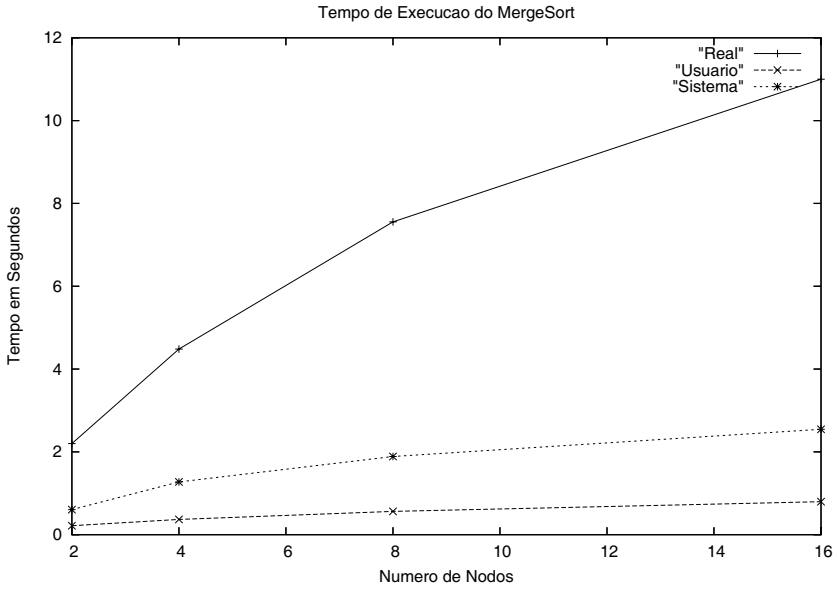

Figura 3: Tempo de execução do algoritmo MergeSort.

com o dispositivo implementado e o impacto causado na vazão por este *overhead* de comunicação.

A vazão foi medida utilizando o tempo de execução de um programa que usa as funções `MPID_Send` e `MPID_Recv` para transferir 100.000 mensagens de 8KB cada. As mensagens são transmitidas sempre entre os nodos mais distantes no hipercubo, isto é, aqueles em que as mensagens precisam ser roteadas através do maior número de nodos. A vazão atingida com o *ch_p4* foi medida apenas entre dois nodos, já que este dispositivo não usa a topologia de hipercubo. Para cada caso, o experimento foi executado cinco vezes e a média dos tempos é apresentada a seguir.

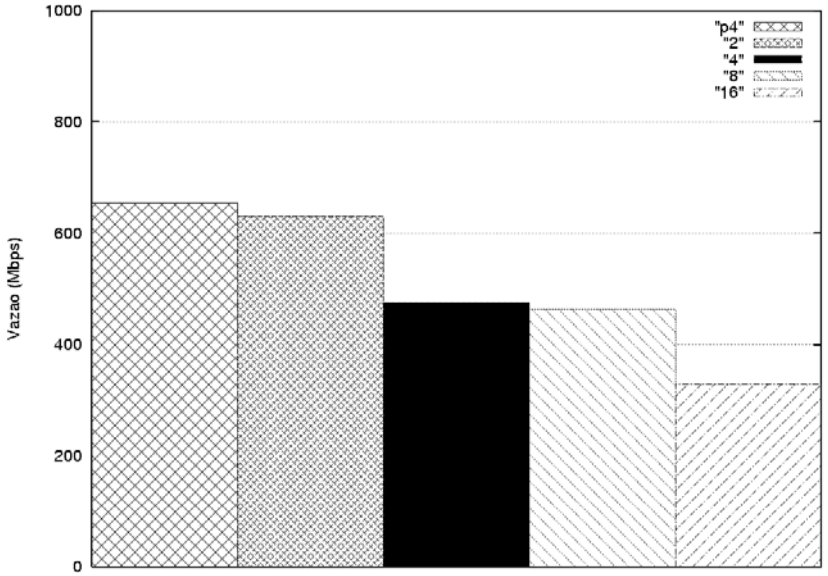

Figura 4: Impacto do roteamento no hipercubo e comparação com dispositivo *ch_p4*.

O gráfico da figura 4 apresenta o resultado. A primeira barra representa a vazão atingida com o dispositivo *ch_p4* sendo de 653,97 Mbps. As demais barras representam

a vazão atingida no pior caso usando o dispositivo implementado, isto é, medida entre os dois nodos mais distantes no hipercubo. A segunda barra representa a vazão atingida na execução com 2 nodos, na qual não houve necessidade de roteamento e atingiu-se a vazão de 629,17 Mbps. A terceira barra representa a vazão atingida em um hipercubo com 4 nodos, no qual é necessário o roteamento através de um nodo e atingiu-se a vazão de 474.72 Mbps. A quarta barra representa a vazão no hipercubo com 8 nodos que possui dois nodos no caminho mais longo e atingiu-se uma vazão de 463.67 Mbps. A ultima barra representa a vazão no hipercubo com 16 nodos, com caminho máximo de 3 nodos e vazão de 328,07 Mbps.

A vazão atingida nos experimentos realizados utilizando o hipercubo varia de 96,32% a 50,22% da vazão atingida pelo dispositivo *ch_p4*, dependendo da quantidade de nodos pelos quais as mensagens precisam ser roteadas. Entretanto, o impacto deste fato sobre o desempenho total das aplicações não é necessariamente desta ordem porque a vazão representada na figura 4 é a vazão no pior caso. Na maioria dos casos as mensagens não precisam ser roteadas através de todo o hipercubo para chegar ao destinatário, o que permite uma vazão maior. Algoritmos clássicos para hipercubos como inversão de matrizes [Foster 1995] e o mergesort distribuído utilizado no experimento anterior foram desenvolvidos de forma que toda a comunicação acontece entre nodos diretamente conectados no hipercubo, não sendo necessário o roteamento de mensagens e obtendo-se sempre o melhor caso da vazão.

Outro motivo pelo qual impacto do roteamento das mensagens sobre o desempenho global das aplicações é menor do que sobre a vazão atingida é que muitas aplicações utilizam a maior parte do tempo total de execução para realizar processamento e apenas uma pequena parte para comunicação em rede. Ou seja, o volume de processamento necessário é grande o suficiente para tornar vantajoso o uso de mais nodos, mesmo que isso implique em um *overhead* causado pelo aumento do diâmetro do hipercubo. Isto pode ser constatado com a execução do programa de renderização de imagens MPI-PovRay mostrada a seguir.

5.3. Renderização de Imagens com MPI-PovRay

O MPI-PovRay [MPI-Povray 2003] é um software de renderização de imagens que utiliza a estrutura de mestre-escravos para realizar a renderização. Foram realizados experimentos com o dispositivo implementado e com o dispositivo *ch_p4*. Uma mesma imagem de 1000x500 pixels foi renderizada utilizando 2, 4, 8 e 16 nodos. Em cada caso a aplicação foi executada três vezes e o tempo médio é apresentado.

Os resultados obtidos utilizando o HyperBone e o *ch_p4* foram bastante próximos e são apresentados no gráfico da figura 5. A linha contínua representa o tempo gasto na execução utilizando o dispositivo implementado e a linha pontilhada representa o tempo gasto na execução usando o dispositivo *ch_p4*.

Utilizando 2 nodos foram necessários 111,66 segundos com o *ch_hyperbone* e 109,33 segundos com o *ch_p4*. Usando 4 nodos o tempo gasto pelo *ch_hyperbone* foi de 37,66 segundos enquanto o *ch_p4* gastou 36 segundos. Com 8 nodos foram necessários com o *ch_hyperbone* 17 segundos e 20,33 segundos com o *ch_p4*. Finalmente, com 16 nodos a mesma imagem foi renderizada em 10,33 segundos usando o *ch_hyperbone* e em 11 segundos usando o *ch_p4*.

6. Conclusão

Este trabalho descreveu a implementação de um dispositivo MPICH que permite executar programas MPI sobre a estrutura de hipercubos virtuais provida pelo HyperBone.

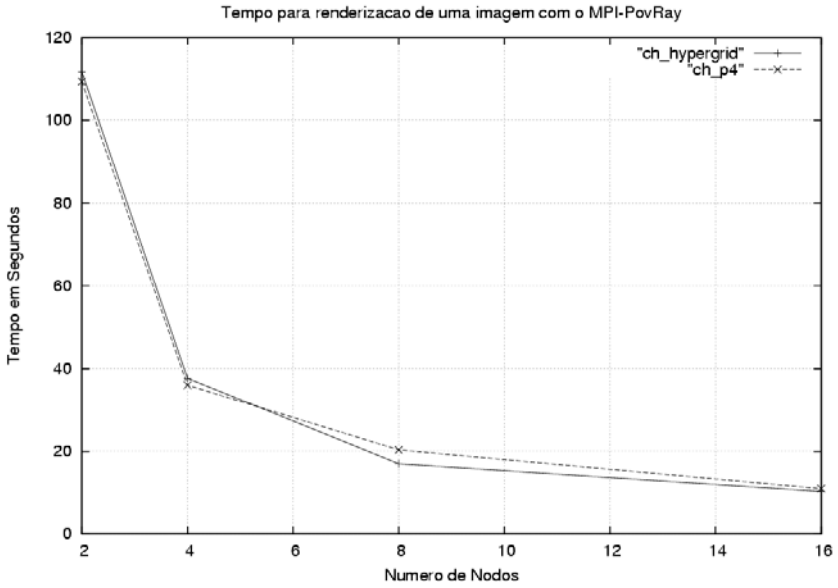

Figura 5: Tempo de renderização de uma imagem utilizando o programa MPI-PovRay.

A possibilidade de executar programas paralelos e distribuídos baseados no padrão MPI sobre esta plataforma provê uma ferramenta poderosa para resolução de problemas que demandem de grandes quantidades de recursos computacionais.

A plataforma HyperBone provê roteamento de mensagens no hipercubo, monitoramento de falhas e determinar quais os nodos mais estáveis ou com melhores características para realizar uma tarefa. A topologia de hipercubos virtuais apresenta uma série de características interessantes para o processamento paralelo como escalabilidade, simetria e diâmetro logarítmico. A implementação do dispositivo foi feita através da codificação das diversas funções necessárias para o ADI. Devido à escassez de documentação a respeito do processo de desenvolvimento de dispositivos ADI a análise das mensagens enviadas e recebidas e do código fonte de outros dispositivos foi fundamental na implementação.

Foram relatados três experimentos realizados com o dispositivo implementado. O primeiro utilizou o algoritmo de ordenação *mergesort* distribuído para demonstrar a funcionalidade do dispositivo. O segundo mediu o impacto do roteamento das mensagens através do hipercubo na vazão atingida, comparando com a vazão atingida com o dispositivo *ch_p4*. O terceiro mostrou o desempenho do dispositivo para execução do MPI-PovRay, uma aplicação de renderização de imagens.

Com os experimentos foi possível constatar que o uso da plataforma de hipercubos reduz a vazão atingida devido à necessidade de roteamento das mensagens. Entretanto, os tempos obtidos para a renderização de imagens com o MPI-PovRay foram bastante próximos dos tempos obtidos com o dispositivo *ch_p4*. Embora o uso da plataforma de hipercubos tenha introduzido um *overhead* devido ao roteamento das mensagens, em aplicações que necessitem grande quantidade de processamento como o MPI-PovRay esse *overhead* torna-se relativamente pequeno. Assim, é vantajoso o uso de grandes quantidades de máquinas, já que o *overhead* no pior caso é proporcional ao diâmetro do hipercubo, que é logarítmico. Isto permite a utilização de quantidades bastante grandes de nodos com *overhead* relativamente baixo.

Para trabalhos futuros podemos destacar a implementação de suporte a

comunicação não-bloqueante e otimizações no envio e recebimento de mensagens. Podemos destacar também a implementação de mecanismos para recuperação do estado do programa MPI em nodos falhos, permitindo a continuidade da computação mesmo na presença destas falhas.

Referências

Amir, Y. and Danilov, C. (2003). Reliable communication in overlay networks. *IEEE International Conference on Dependable Systems and Networks (DSN 2003)*, pages 511–520.

Bona, L. C. E., Jr., E. P. D., and Fonseca, K. V. O. (2005). Hypergrid: Uma arquitetura de grade baseada em hipercubo virtual. *3o Workshop de Grade Computacional e Aplicações*.

Forum, M. P. I. (1994). Mpi: A message-passing interface standard. *International Journal of Supercomputer Applications*, 8(3/4):165–414.

Foster, I. (1995). *Designing and Building Parallel Programs : Concepts and Tools for Parallel Software Engineering*. Addison-Wesley.

Foster, I. and Kesselman, C. (2003). *The Grid*. Morgan Kaufmann, 2 edition.

Gropp, W. and Lusk, E. (1994). An abstract device definition to support the implementation of a high-level point-to-point message passing interface. Technical Report MCS-P392-1193, Argonne National Laboratory.

Gropp, W., Lusk, E., Doss, N., and Skjellum, A. (1996). A high-performance, portable implementation of the mpi message passing interface standard. *Parallel Computing*, 22:789–828.

MPI-Povray (2003). Distributed povray using mpi message passing. http://www.verrall.demon.co.uk/mpipov/. Acesso em março de 2003.

Pacheco, P. (1996). *Parallel Programming with MPI*. Morgan Kaufmann.

Posters

ContextGrid - Uma infra-estrutura de suporte a contexto para integração de dispositivos móveis a grades computacionais

Alaor José da Silva Junior[1], Márcio Nunes de Miranda[1], Fábio Moreira Costa[1]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
Caixa Postal 131 – 74001-970 – Goiânia, GO – Brazil

{alaor,miranda,fmc}@inf.ufg.br

Resumo. *Este trabalho descreve uma arquitetura de computação sensível ao contexto, que provê a base para a integração de dispositivos móveis em ambientes de computação em grade, permitindo a adaptação dinâmica do middleware de acordo com as variações de contexto a que estes dispositivos estão sujeitos.*

1. Introdução

As grades computacionais [Foster *et al* 2001] podem ser formadas por diversos tipos de computadores. Mais recentemente, dispositivos móveis têm sido interligados às grades computacionais fixas [Phan 2002], representando um novo cenário para o uso desses dispositivos. A exploração deste cenário depara-se com a natureza altamente dinâmica dos dispositivos e suas diversas restrições. Em ambientes extremamente dinâmicos e diversificados como este, uma arquitetura sensível ao contexto [Dey 2000], capaz de reagir e adaptar-se, possibilita um melhor aproveitamento dos recursos envolvidos, melhor usabilidade e agregação de variados tipos dispositivos e recursos.

Este trabalho descreve uma infra-estrutura de suporte a contexto e à adaptação dinâmica, integrada ao *middleware* de grade computacional. Os objetivos são a construção de uma infra-estrutura sensível ao contexto e adaptável para *middleware* de grades computacionais e a integração de dispositivos móveis a grades computacionais.

2. ContextGrid

O ContextGrid é uma infra-estrutura de suporte à computação sensível ao contexto e à adaptação dinâmica, desenvolvida como uma extensão do MAG [Lopes 2006], que acrescenta novos componentes e introduz novos conceitos, protocolos e serviços, para que o MAG possa considerar o contexto dos nós envolvidos diretamente com uma tarefa, bem como o contexto geral da grade. A introdução de contexto e adaptação dinâmica visa melhorar a integração com dispositivos móveis. O ContextGrid não oferece serviços diretamente às aplicações, como ilustra a Figura 1. Ele está situado sob a camada MAG, trabalhando como uma extensão, preservando as interfaces do MAG e inter-operando com seus módulos.

Para compor a infra-estrutura do ContextGrid são introduzidos dois módulos principais: o LCM (*Local Context Manager*) - responsável por coletar as informações de contexto das mais diversas fontes, de forma transparente, padronizá-las e enviá-las ao GCM (*Global Context Manager*). O GCM - responsável por armazenar, processar e converter as informações contextuais em contexto, transformando-as em contextos mais elaborados, ou pela junção de várias informações ou pela inferência de novos dados a

partir daqueles obtidos dos dispositivos. É parte desse processamento a classificação dos contextos em categorias, adotando critérios como similaridade, proximidade e/ou domínio administrativo. O GCM também é responsável por realizar as notificações de mudanças de contexto a todos os interessados através de canais de eventos.

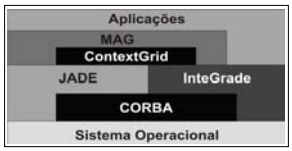

Figura 1. Arquitetura em camadas do contextGrid

Na arquitetura proposta a adaptação não é o foco principal do trabalho. São providos meios pelos quais os módulos do *middleware* possam tomar conhecimento do contexto e de suas mudanças, ficando a cargo do desenvolvedor decidir qual(ais) contexto(s) é(são) interessante(s) e qual(ais) ação(ões) deve(m) ser tomada(s) diante de um determinado contexto ou de sua mudança. Isso torna a arquitetura mais flexível no que se refere à adaptação.

3. Considerações finais

O ContextGrid provê um conjunto de recursos para suporte à computação sensível ao contexto e com adaptação dinâmica do *middleware*. Esta abordagem visa simplificar a integração dos mais diversos dispositivos móveis às grades, resolvendo ou amenizando os desafios introduzidos, como a diversidade entre os dispositivos, a onipresença do ambiente computacional e o comportamento altamente dinâmico.

Dentre os principais trabalhos relacionados, pode-se destacar**:** ISAM [Yamin *et al* 2003] e o *middleware* MoCA [Viterbo *et al* 2006]. Como trabalhos futuros, pretende-se usar redes de sensores para fornecer contexto, explorar recursos adaptáveis e usar reflexão computacional consciente de contexto no *middleware* de grade.

Referências

Dey, A. K. (2000) “Providing Architectural Spport for Building Context-Aware Applications”, Ph.D. Thesis, Georgia Institute of Technology.

Foster, I., Kesselman, C. and Tuecke, S. (2001) “The Anatomy of the Grid: Enabling Scalabe Virtual Organizations”, The International Journal of Supercomputer Application, 15(3).

Lopes, R. F. (2006) "MAG: uma grade computacional baseada em agentes móveis", Dissertação de Mestrado, Universidade Federal do Maranhão, São Luiz, MA, Brasil.

Phan, T., Huang, L. and Dulan, C. (2002) “Challenge: Integrating Mobile Wireless Devices Into the Computational Grid”, In: MobiCom '02.

Viterbo, J., Sacramento, V., Rocha, R.C.A. da, Endler, M. (2006) "MoCA: Uma Arquitetura para o Desenvolvimento de Aplicações Sensíveis ao Contexto para Dispositivos Móveis", Proc. of the SBRC 2006, Tool Session, Curitiba.

Yamin, A., Barbosa, J., Augustin, I., Silva, L., Geyer, C., Cavaleiro, G. (2003) “Towards Merging Context-aware, Mobile and Grid Computing”, International Journal Of High Performance Applications, London: Sage Publications, v.17, n.2.

Um Ambiente Hierárquico para Definição e Negociação de Recursos Ociosos em uma Grade Institucional

Lourival Aparecido de Góis[1], Walter da Cunha Borelli[2]

[1]Coordenação de Informática – Universidade Tecnológica Federal do Paraná, Campus Ponta Grossa (UTFPR)
Av. Monteiro Lobato, s/n, Km 04 – Ponta Grossa - Pr

[2]Departamento de Telemática, Faculdade de Engenharia Elétrica e de Computação – Universidade de Campinas (UNICAMP)

gois@pg.cefetpr.br, borelli@dt.fee.unicamp.br

Resumo. *Este artigo apresenta um modelo hierárquico de busca e disponibilização de recursos ociosos, localizados em uma Instituição de ensino com vários campi, distribuídos e interconectados através de um backbone confiável e gerenciado por regras de QoS.*

1. Introdução

Este trabalho propõe um modelo que possibilite o levantamento e o gerenciamento de recursos segundo uma estrutura hierárquica [Kacsuk et al. 2002], regulamentada por políticas de negociações, ou seja, um ambiente onde o elemento fundamental é o proprietário dos recursos, pertencente a um domínio onde é instituído um aglomerado que representa um conjunto limitado e restrito de nós interconectados através de uma estrutura de rede local de alta velocidade [Parashar et al. 2005]. Como é comum a estruturação de várias redes locais em uma instituição, normalmente interligadas por enlaces rápidos e confiáveis, será possível então a criação de vários aglomerados, interligados através de seus elementos gerenciadores que negociam o uso apropriado destes recursos em uma grade institucional e proprietária.

2. Definição e Implementação do Modelo

O modelo está baseado em dois componentes, que são: CRAM (*Cluster Resource Allocation Manager*), é definido como sendo o gerente do aglomerado e tem como responsabilidades a centralização de todas as informações sobre os recursos existentes no seu domínio, a interconexão com outros gerentes e a recepção e alocação das tarefas submetidas pelos usuários; RAA (*Resource Allocation Agent*), pertencentes a usuários da rede que disponibilizam seus recursos ociosos segundo uma política pré-definida. Este componente tem como requisitos a determinação dos recursos existentes no equipamento no qual ele está instalado, bem como identificar quais destes recursos estão disponíveis segundo critérios pré-estabelecidos.

Visando a implantação da estrutura de uma grade computacional que possa ser utilizada em uma instituição de ensino, a partir dos microcomputadores instalados em seus laboratórios de informática, distribuídos em vários *campi,* e interconectados por um *backbone* gerenciado por regras de QoS, foram desenvolvidos os protótipos dos componentes apresentados no modelo a partir da linguagem Java. O projeto encontra-

se atualmente em fase de implementação dos módulos responsáveis pelo escalonamento e submissão de tarefas no aglomerado, considerando para isto, a execução de aplicações do tipo *bag-of-task* [Cirne et al. 2003], compostas por tarefas independentes, o que facilita a sua especificação em *threads* e consequentemente sua transferência em um ambiente distribuído através dos mecanismos de serialização de objetos da linguagem de programação adotada.

3. Análise do Consumo de Recursos pelos Componentes do Modelo

Como premissa de modelagem para o ambiente proposto, um dos fatores mais relevantes é o consumo de recursos como processador e memória pelos componentes RAA e CRAM nos equipamentos em que estiverem instalados, bem como o aumento de tráfego provocado na rede pela comunicação entre eles. Em função disto, foram realizados alguns testes visando o levantamento e análise destes pontos críticos.

A partir resultados obtidos, ficou constatado que o consumo dos recursos monitorados não foi expressivo, apresentando resultados que se mantiveram próximos a 5% da capacidade total do processador e memória nos instantes em que os índices de disponibilidades eram recalculados e enviados ao gerente do aglomerado. Com relação a estrutura de rede, verificou-se que o tráfego gerado pelos componentes manteve-se em média de 5% com relação ao tráfego total.

Os próximos esforços estão voltados para a definição de novos mecanismos de interconexão, o que exigirá novas simulações visando a verificação do comportamento do consumo do processador, da memória e do *overhead* causado na estrutura de rede, utilizando outras tecnologias como RMI (*Remote Method Invocation*) e CORBA (*Common Object Request Broker Architecture*) ao invés de *sockets*.

4. Conclusões

O modelo apresentado visa a construção de um sistema computacional em grade a partir de uma base de máquinas comuns existentes na instituição onde o projeto está sendo executado, tendo como critérios a utilização de tecnologias emergentes, principalmente no que se refere aos recursos disponíveis na linguagem Java, como sua portabilidade, facilidade de desenvolvimento de aplicações distribuídas e fundamentações no paradigma da orientação a objetos.

5. Referências Bibliográficas

Kacsuk P., Kranzlmuller D., Volkert J. and Nemeth Z., (2002). "Distributed and Parallel Systems: Cluster and Grid Computing", In Kluwer International Series in Engineering and Computer Science.

Cirne, W., D. Paranhos and F. Brasileiro et al. (2003) "Running Bag-of-Tasks Applications on Computational Grids: The MyGrid Approach", No Proceedings of the International Conference on Parallel Processing (ICPP-03).

Parashar, M. and Browne, J.C. (2005) "Conceptual and implementation models for the grid", Proceedings of the IEEE. Volume 93, Issue 3, Page(s):653 – 668. Digital Object Identifier 10.1109/JPROC.2004.842780

Mobilidade em Ambientes de Grades Computacionais

Hans Alberto Franke, Carlos Becker Westphall, Carlos Oberdan Rolim, Fabio Navarro, Fernando Koch

Universidade Federal de Santa Catarina (UFSC)
Departamento de Informática e Estatística (INE), Laboratório de Redes e Gerência (LRG), Caixa Postal 476, 88040-970, Florianópolis - SC.

{koiote@inf.ufsc.br, westphal,oberdan,fabio,koch@lrg.ufsc.br}

Resumo. *Este artigo visa o estudo de um ambiente hospitalar que utiliza grades computacionais, para coleta e disponibilização de dados vitais dos pacientes em tempo real. Propõe-se a criação de um middleware que permita a integração dos dados coletados através dos sensores com a grade computacional, permitindo assim uma homogeneização dos nodos da grade e uma melhoria quanto à facilidade oferecida ao usuário. Este middleware deve possuir os mínimos recursos necessários existentes em outros middlewares, suportando também dispositivos móveis.*

1. Introdução

Para estudo da proposta utiliza-se a telemedicina, em específico o monitoramento remoto de pacientes em um leito hospitalar, o qual é feito utilizando sensores. Pelo fato de existirem diversos equipamentos espalhados pelo hospital, a homogeneização desses equipamentos se torna uma questão importante. Ou seja, os dados dos diversos equipamentos têm que ser fornecidos ao médico de forma rápida, segura e proveniente de todos os equipamentos.
Neste artigo propõe-se a utilização de grids computacionais para a solução deste problema, pois permite a visão de cada equipamento distinto como sendo um nodo pertencente à grade. Isso facilita muito as funções de gerenciamento, comunicação e integridade dos dados.

Procura-se a criação de uma arquitetura que pudesse garantir duas questões fundamentais:

(1) Garantir que os diversos coletores, atuadores e dispositivos móveis possam ser gerenciados de maneira homogênea, sem a necessidade da criação de uma interface para cada dispositivo;
(2) Como divulgar esses dados em tempo real.

As principais contribuições desse trabalho são: o estudo dessa arquitetura de grade computacional, e a proposta da criação de um grid middleware que facilite a integração entre a parte de coleta de dados, integração com a grade e disponibilidade para o usuário, e que possua os mínimos recursos necessários existentes em outros middlewares e além disso possa suportar o ingresso de dispositivos móveis.

2. Metodologia

A arquitetura proposta [Navarro 2005] foi construída estudando o caso de uso do hospital e da telemedicina. Ela foi proposta após a constatação que em um hospital existem diversos tipos de sensores diferentes, o que acarretaria em uma interface diferente para cada dispositivo na construção do middleware. Com isso a sua implementação se tornaria bastante difícil. Então usou-se a idéia de virtualização de recursos [Joseph 2004], uma das vantagens de se trabalhar com grid.

Realizou-se um estudo entre os diversos middlewares existentes no mercado para descobrir quais deles ofereciam suporte a dispositivos móveis. E apenas o GridBus oferece esse suporte, de maneira muito simples.

A partir dessas duas pesquisas foi proposta a criação de um middleware que suportasse a esta arquitetura e, além disso, oferecesse suporte para execução de dispositivos móveis.
Ele foi desenvolvido em linguagem Java Micro Edition, J2ME, desenvolvida pela SUN. Utilizando a IDE Eclipse com plugin para o Wireless ToolKit. Foi utilizado o MIDP 2.0, pois fornece recurso de requisições https, parte importante quanto á segurança. Os dados coletados pelos sensores são convertidos para XML, e disponibizados para a grade e para o httpServer. O httpServer foi desenvolvido utilizando o software Vertrigo, para a simulação de uma requisição de um usuário fora da grade.

3. Resultados

A arquitetura foi divida em 4 camadas: Sensor, Inference, Grid Middleware e Grid Service. A Grid Middleware também foi dividia em 4 camadas: Node Managemente Interface, Monitoring Interface, Profile Interface, Data Comunication Interface. Isto foi feito para garantir um reaproveitamento de tecnologia e garantir uma maneira fácil de implementação.

As requisições entre o dispositivo móvel e httpServer foram feitas no formato http. O dispositivo móvel para fazer essas requisições utiliza-se de http.Connection, inputStream e OutpuStream.

Tudo isso acontece de forma muito simples, pela divisão de camadas da grid middleware, pois alem de cada interface ser construída de forma modular, ou seja, independente das demais, a alteração em um modulo não interfere nas funções da outra interface. Basta definir as funções de entrada e saída em cada interface.

Para estudo de caso foi utilizado o exemplo da telemedicina. Simulando a necessidade de um médico se conectar na grid utilizando um dispositivo móvel. Para isso ele deve acessar o url do httpServer, logar-se, e procurar por pacientes e tarefas, que estão salvas no httpServer através de arquivos XML.

A grade computacional proporcionou a homogeneidade e virtualização de recursos e serviços no gerenciamento dos equipamentos conforme havíamos propostos inicialmente. Pelo fato de estarmos usando apenas um pequeno conjunto de funcionalidades proporcionadas pela grade computacional, à camada grid middleware abstraiu a complexidade de implementação das funções que fazem uso da grade. Dessa forma conseguimos desenvolver o protótipo do ambiente para rodar em diferentes tipos de equipamentos sem deixar o código atrelado a um tipo específico.

4. Conclusão

Neste artigo procurou-se resolver o problema de gerenciamento de diferentes tipos de equipamentos e a sua disponibilização em tempo real. Adotou-se uma arquitetura em grades computacionais, permitindo assim uma homogeneização dos mesmos, garantindo uma facilidade na implementação e gerenciamento dos recursos provenientes de diferentes equipamentos, pois cada equipamento passa a ser visto como um nodo na grade, igual aos demais.

A utilização da grade permitiu também critérios como: segurança, comunicação e compartilhamento de recursos, características naturais de uma grade, garantindo assim um reaproveitamento de tecnologia, pois não foi necessária a criação de uma interface para cada equipamento especifico.

A principal contribuição do artigo foi a implementação de uma grid middleware dentro da arquitetura proposta. O que prova que o uso de uma grade computacional e a grid middleware, para o problema de coleta de recursos e disponibilização em tempo real é uma boa solução, uma vez que a grade fornece reaproveitamento de tecnologia e grande poder computacional.

Notou-se também que é possível construir uma grid middleware com um conjunto mínimo de funções necessárias que são providos pelos demais middlewares, (Ex: registerNode, deleteNode, sendTask, checkTaskStatus, fetchTaskResults, getNodes, getNodeStatus, etc...), e com suporte a participação de dispositivos móveis, que não eram oferecidos pelos demais com exceção do GridBus.

5. Bibliografia

Foster, I.; Gannon, D. & Kishimono, H. (2003) “The Open Grid Service Architecture”, http://forge.gridforum.org/projects/ogsa-wg, Novembro.

Foster, I., & Kesselman, C. (2004) “The Grid: Blueprint for a New Computing Intrastructure”, 2nd Etidion, Willey 2004.

Joseph, J., Ernest, M., Fellenstein, C.(2004), “Evolution of grid computing architecture and grid adoption models”, IBM Systems Journal, Volume 43, Numero 4, Junho 2004.

Navarro, Fabio P. (2006) “Um middleware para Grades de Dispositivos Móveis”. Dissertaçao de Mestrado. PPGCC-UFSC, Florianópolis, 2006.

Gerência e Fornecimento de Certificados de Confiança para Nós Móveis Usando Certificados de Curta Duração.

Fabio L. Licht[12], Bruno R. Schulze[1], Edison Ishikawa[2]
[1]LNCC – Laboratório Nacional de Computação Científica
Av. Getulio Vargas, 333, 25651-070 - Petrópolis, RJ, Brasil
[2]IME – Instituto Militar de Engenharia – Departamento de Engenharia de Sistemas
Praça General Tibúrcio, 80/ DE-9 – Praia Vermelha, 22290-270 - Rio de Janeiro, RJ – Brasil
{licht, schulze}@lncc.br
{fllicht, ishikawa}@de9.ime.eb.br

Abstract - This work presents a proposal to extend an existing service for short lived certificates (SLCs), with purpose of allowing mobile nodes to enter computational grids. Amongst the parts of this work, it is cited, the use public, private keys, supply of trustworthy certificates and dynamic SLCs. MyProxy is the tool responsible for supplying dynamic certificates and greater agility in the inclusion of new users and also the extension to nodes that want to enter a computational grid.

Resumo *- Este trabalho apresenta a proposta de extensão de um serviço existente de fornecimento de certificados confiáveis de curta duração, com o intuito de autorizar nodos móveis a ingressarem temporariamente em grades computacionais. Dentre as atividades deste trabalho, inclui-se a utilização de chaves públicas, privadas, fornecimento de certificados confiáveis e dinâmicos além de criptografia e utilização de redes móveis. O uso destes padrões em conjunto com a ferramenta MyProxy, responsável por fornecimento de certificados dinâmicos de curta duração, irá propiciar maior agilidade na inclusão de novos usuários e a sua extensão para inclusão de nós móveis que quiserem associar-se à grade computacional.*

1. Introdução

Problemas relacionados à segurança em grades computacionais têm dado motivos para pesquisa nesta área. Em uma grade ou cluster a idéia de segurança resume-se em fornecer a uma determinada máquina e ao usuário desta a permissão para o uso como define Lucas em [R05] e esta "permissão" é fornecida, normalmente, pelo gerente de autenticação de chaves públicas e privadas. Lorch [M04] cita a utilização de kerberos para criação destas chaves.

O funcionamento é simples como definem Santim [A03] e Wangham [M03], o usuário gera a sua chave privada e uma chave pública, a chave pública é enviada à unidade certificadora da grade para ser assinada pelo gerente de certificados. O problema nisso é que os certificados ficam em poder da pessoa que fez o pedido e dependendo dos critérios de segurança desta entidade, ou seja, ficam as chaves sujeitas à proteção do sistema de arquivos local, podendo assim cair em mãos erradas.

Uma solução é a utilização de certificados de curta duração e quando da necessidade de uso por uma entidade, este certificado seria repassado ao solicitante através um sistema confiável para distribuição destes. Outras soluções para a segurança também envolvem a utilização de criptografia como descrito por Menezes [A01] e exigência de senhas seguras.

Dispositivos móveis como PDAs, Celulares, PCs ou Laptops com rede Wireless tendem a gerar preocupação maior por serem sistemas pervasivos como define Mattes em [L04] e necessitam de mecanismos de segurança mais flexíveis e seguros que os disponíveis em sistemas baseados em redes locais e como não é possível em uma grade computacional se limitar à restrição de acesso dos usuários a delegação de direitos torna-se um desafio.

2. Segurança na Grade

Quando se pensa em segurança em uma rede local é relativamente simples criar uma política que sirva de base, em uma grade isso não é tão simples, tem-se que pensar que a grade pode estar interligada a diversas entidades de sub-redes diferentes e com comunicação por meios físicos diferentes

Quando se pensa em redes móveis ou dispositivos visitantes, laptops e PDAs, por exemplo, tratar de segurança torna-se uma tarefa ainda mais difícil. Deve-se pensar em um sistema de autenticação e fornecimento de chaves de um modo descentralizado e uma proposta a este é a criação de uma ferramenta que possa não só fornecer chaves públicas a entidades que desejam participar da grade, mas também fornecer uma garantia de segurança validando as chaves e verificando a autenticidade das entidades de forma descentralizada por dispositivos de confiança, análogo ao que ocorre com um serviço chamado MyProxy [G05].

MyProxy é um sistema de repositório de certificados para uso em grades computacionais como descrito por Basney em [J03] e foi inicialmente desenvolvido para delegar credenciais a entidades de confiança, ou seja, diversas entidades poderiam fornecer certificados coletados a partir de um repositório aos usuários que as solicitassem. Como MyProxy fornece certificados a usuários temporários, a mesma idéia pode ser usada para fornecer certificados de máquina, sendo esta uma das propostas deste trabalho.

Em uma grade padrão os certificados que forem gerados tanto para máquinas quanto para usuários tem um prazo de validade relativamente grande. Durante este prazo pode ocorrer de uma máquina ou usuário deixar de fazer parte da grade e seus certificados continuarem válidos, deixando uma falha na segurança. No caso de dispositivos móveis ou visitantes o problema se agrava e deve-se garantir de uma maneira confiável que certificados estejam disponíveis somente a usuários e máquinas autênticas obrigando a um usuário que faça requisição de um certificado de máquina a ter um certificado de usuário antes.

3. Conclusão

Em uma grade computacional, na utilização de dispositivos móveis ou visitantes é bem clara a necessidade de segurança. Torna-se um padrão usar os critérios de redes LAN para definição das políticas de segurança, o problema nisso é que a implementação destas políticas pode não ter um resultado como esperado. A utilização de certificados com prazos curtos usando MyProxy se mostrou eficiente em termos de segurança e sua implementação mais abrangente se faz necessária.

Num ambiente de grade o fornecimento de permissões de uso torna-se uma tarefa de difícil administração, centralizar a autenticação usando certificados com chaves públicas e privadas pode causar demora quando é necessária a assinatura de uma entidade certificadora. Como a tarefa de autenticação não é automatizada, se a rede estiver com problemas ou se o responsável pela assinatura das chaves não estiver disponível um certificado pode demorar, o que pode comprometer o uso da grade por usuários de redes móveis.

Assim sendo a idéia central deste trabalho é automatizar a geração de certificados de uma forma descentralizada e ao invés de fornecer certificados com prazo de validade longo, utilizar certificados de curta duração para que os usuários e máquinas possam fazer suas submissões de tarefas bem como interagir com a grade.

4. Referências Bibliográficas

[M04] M. Lorch, J. Basney, D. Kafura, “A Hardware-secured Credential Repository for Grid PKIs” http://www.ncsa.uiuc.edu/~jbasney/myproxy-ccgrid2004.pdf

[G05] Globus Alliance, http://www.globus.org/toolkit/docs/4.0/security/

[A01] A. Menezes, P. Oorschot, S. Vanstone, “Handbook of Applied Cryptography – Chapter 8” http://www.cacr.math.uwaterloo.ca/hac/

[R05] R. M. de Lucas, “Seguridad y Uso de Certificados Digitales en GRID”.

[L04] L. Mattes, L. G. Kiatake, E. A. R. Marques , J. A. Zuffo, “Linguagem Lógica Formal para Expressar Segurança em Ambientes Pervasivos”

[J03] J. Basney, W Yurcik, R Bonilla, A Slagell, “Credential Wallets: A Classification of Credential Repositories Highlighting MyProxy” National Center for Supercomputing Applications (NCSA)

[A03] A. O. Santin, J. da S. Fraga, E. R. de Mello, F. Siqueira, “Teias de federações como extensões ao modelo de autenticação e autorização SDSI / SPKI” - XXI Simpósio Brasileiro de Redes de Computadores.

[M03] M. S. Wangham, J. da S. Fraga, “Mecanismos de Segurança para Plataformas de Agentes Móveis, baseados em Redes de Confiança SPKI/SDSI” - XXI Simpósio Brasileiro de Redes de Computadores.

Análise Paralela e Distribuída de Dados Micrometeorológicos Utilizando a Plataforma JXTA*

Marcelo Veiga Neves[1], Tiago Scheid[2], Andrea Schwertner Charão[2], Guilherme Sausen Welter[3], Osvaldo Luiz Leal de Moraes[3]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)

[2]Laboratório de Sistemas de Computação (LSC)
Curso de Ciência da Computação – Universidade Federal de Santa Maria (UFSM)

[3]Laboratório de Micrometeorologia (LμMet)
Centro de Ciências Naturais e Exatas – Universidade Federal de Santa Maria (UFSM)

mvneves@inf.ufrgs.br, {scheid,andrea}@inf.ufsm.br

gswelter@yahoo.com.br, ollmoraes@smail.ufsm.br

1. Introdução

Grades computacionais e computação *peer-to-peer* (P2P) vêm se afirmando como soluções para o processamento de aplicações que demandam grande quantidade de recursos. De fato, ambas abordagens permitem agregar recursos em ampla escala, acomodando sistemas heterogêneos cuja disponibilidade pode variar dinamicamente. Neste sentido, uma abordagem que vem sendo explorada é o uso de infra-estruturas P2P, por exemplo JXTA [JXTA 2001], como base para serviços e aplicações típicos de grades computacionais [Cirne et al. 2003].

O presente artigo relata uma experiência de uso da plataforma JXTA para a execução paralela e distribuída de uma aplicação que analisa grandes conjuntos de dados coletados por sensores micrometeorológicos. O objetivo do trabalho é, de um lado, obter ganho de desempenho com a adaptação da aplicação existente e, de outro, flexibilizar o uso dos recursos computacionais disponíveis, permitindo aproveitar a ociosidade dos mesmos. A abordagem adotada para adaptação da aplicação permite que a mesma seja executada tanto em grades computacionais como em redes P2P, o que reforça a aproximação entre estes dois paradigmas.

2. Arquitetura do Sistema Desenvolvido

A fim de aproveitar os computadores ociosos do laboratório LμMet e também aumentar o desempenho da aplicação, desenvolveu-se um sistema capaz de prover uma plataforma dinâmica de execução. Este sistema segue um modelo P2P puro [Schollmeier 2001], onde os *peers* são os computadores que integram a rede virtual constituída através dos protocolos JXTA.

Os *peers* podem interagir entre si, através da troca de mensagens, ou receber requisições de clientes. Clientes permitem que os usuários submetam trabalhos para a plataforma de execução. Cada *peer* do sistema é constituído por controlador, um monitor de ociosidade e um ou mais trabalhadores. Esses elementos são encapsulados em um

*Trabalho parcialmente financiado por FAPERGS e CNPq.

programa que é executado em cada *peer* como um processo *daemon*. O controlador é responsável por manter um lista de trabalhadores livres e pela distribuição e coordenação de tarefas. Os trabalhadores recebem tarefas, processam-nas e retornam o resultado, além de interagir com o monitor de ociosidade para descobrir se o *peer* local está em estado ocioso.

De posse desse sistema, foi possível implementar e avaliar duas abordagens diferentes de adaptação da aplicação alvo, chamadas de versão distribuída e versão paralela. Na versão distribuída, a aplicação foi adaptada para permitir que trabalhadores livres processem diferentes conjuntos de dados. Já na versão paralela, divid-se o problema em várias tarefas que são distribuídas entre os trabalhadores.

3. Avaliação

A fim de avaliar o ganho de desempenho utilizando o sistema implementado foram realizados testes com as versões distribuída e paralela da aplicação alvo. Para realizar os testes foi constituída uma plataforma de execução formada por 5 *peers*. Os resultados obtidos foram comparados com a versão seqüencial da aplicação, que executa durante 24,04 minutos.

No caso da versão distribuída, foi realizado o processamento simultâneo de arquivos de dados micrometeorológicos. Duplicando-se a quantidade de dados a ser processados o tempo passou a 31,18, o que representa um aumento de 1,36 % no tempo de processamento. Em comparação com a versão seqüencial, que processava um arquivo em 24,04 minutos, a versão distribuída obteve um ganho de 35 % ao processar 2 arquivos simultaneamente. Ao processar 3 arquivos simultaneamente, esse ganho passou para 55 %. A versão paralela também apresentou ganho de desempenho. Mesmo no caso em que foram usados somente 2 trabalhadores, o *speedup* foi de 1.555, o que corresponde a 77.8 % de eficiência.

4. Conclusão

Este trabalho apresentou o uso de JXTA no suporte à distribuição e paralelização de uma aplicação que analisa grandes conjuntos de dados coletados por sensores micrometeorológicos. Com as adaptações realizadas na aplicação e sua execução no sistema desenvolvido, obteve-se ganho de desempenho com o processamento de maiores conjuntos de dados em um tempo mais curto. Como trabalhos futuros, pretende-se principalmente aprofundar a avaliação do sistema desenvolvido, comparando-o com outras soluções baseadas em JXTA e analisando o impacto do monitor de ociosidade.

Referências

Cirne, W. et al. (2003). The OurGrid Project: Running Bag-of-Tasks Applications on Computational Grids. In *SC'2003 Conference CD*. IEEE/ACM SIGARCH.

JXTA (2001). Project JXTA Web site. http://www.jxta.org/. Acesso em abril de 2006.

Schollmeier, R. (2001). A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. In *Peer-to-Peer Computing*, pages 101–102.

Specification of a MyProxy Plugin for Mozilla

Luiz M. R. Gadelha Jr.[1]

[1]Coordenação de Sistemas e Redes
Laboratório Nacional de Computação Científica
Av. Getúlio Vargas, 333 – Quitandinha
25.651-075 – Petrópolis – RJ – Brasil

lgadelha@lncc.br

Abstract. *This poster specifies a MyProxy plugin for the Mozilla web browser. It should allow grid users to generate and store their credentials in the cryptographic repository of the browser and to delegate proxy credentials to MyProxy repositories.*

Resumo. *Este poster especifica um plugin MyProxy para o navegador web Mozilla. Ele deve permitir que um usuário de grade gere e armazene a sua credencial no repositório criptográfico do browser e que delegue credenciais proxy para um repositório MyProxy.*

1. Introduction

The National System for High-Performance Computing (SINAPAD) [SINAPAD 2006] is composed of seven HPC centers thoughout Brazil. A project for setting up a production-level computational grid to serve the Brazilian scientific community was concluded recently. The grid was implemented using standard tools such as the Globus toolkit [Ferreira et al. 2003] and Gridport [Thomas et al. 2001]. The *Grid Security Infrastructure* (GSI) [Butler et al. 2000] provides the basic security services like data confidentiality and integrity and entity authentication. GSI relies heavily on public-key infrastructure (PKI) [Housley and Polk 2001] techniques. A PKI was set up for the grid using OpenCA [OpenCA Labs 2006], a software toolkit which implements all the functionality needed to run a certification authority (CA). The certificates are issued for users and hosts of the grid and are used in SSL client-server mutual authentication. GSI extends standard PKI by defining *proxy certificates* which are short-term credentials issued by the user long-term credential. This allows one to reduce the exposure of a credential private key and to perform privilege delegation within the grid.

2. Plugin Specification

Currently a user is admitted to the grid after performing a number validation steps in the grid administrative system. After being validated the user must generate a certificate request using standard GSI tools such as *grid-cert-request* or a Java application developed by SINAPAD. The files generated in this process are stored in the local filesystem and are managed by the user. The request is sent to the grid CA where the certificate is issued and sent back to the user. To effectively use the grid the user must delegate a proxy certificate to the MyProxy repository using the `myproxy-init` tool or the Java CoG Kit [Java CoG Kit 2006]. Users frequently find this procedure to be difficult since it involves

managing files containing their credentials and running applications to upload their proxy certificates.

The majority of the browsers today are fully compliant with PKI technology. Usually they come with software-based cryptographic tokens which allows a user to transparently generate certificate requests and store certificates issued by a CA. This makes the process of requesting certificates to a CA easier since the users would simply fill a form with their data. Upon submission of the form the browser's cryptographic token is automatically activated to generate a key pair and a certificate signing request (CSR). The CSR is sent together with the form to the CA. Installation of the certificate is also very simple since the CA usually sends an e-mail containing a link to the certificate repository. After accessing the link the certificate is automatically installed in the browser's cryptographic token. A functionality not available yet is the ability to generate and upload proxy certificates to a MyProxy repository.

A Java-based plugin for the Mozilla web browser is being developed within the context of SINAPAD which will be able to interact with MyProxy repositories. It makes use of the Java CoG Kit and Mozilla Network Security Services (NSS) which is a library that allows one to develop PKI routines using the browser's cryptographic token.

3. Conclusion

It is expected that using browser native PKI tools will make the process of accessing the SINAPAD computational grid considerably simpler than now. The use of the software-based cryptogrphic tokens contained in browsers will also improve security since the user will no longer have to manage files containing their credentials. It will also be easier to use hardware-based cryptogrphic tokens since these interface well with the current browsers.

References

Butler, R., Engert, D., Foster, I., Kesselman, C., Tuecke, S., Volmer, J., , and Welch, V. (2000). A National-Scale Authentication Infrastructure. *IEEE Computer*, 33:60–66.

Ferreira, L., Berstis, V., Armstrong, J., Kendzierski, M., Neukoetter, A., Takagi, M., Bing-Wo, R., Amir, A., Murakawa, R., Hernandez, O., Magowan, J., and Bieberstein, N. (2003). *Introduction to Grid Computing with Globus*. IBM Redbooks.

Housley, R. and Polk, T. (2001). *Planning for PKI: Best Practices Guide for Deploying Public Key Infrastructure*. Wiley.

Java CoG Kit (2006). `http://www-unix.globus.org/cog/java/`.

OpenCA Labs (2006). `http://www.openca.org`.

SINAPAD (2006). `http://www.lncc.br/sinapad`.

Thomas, M. P., Mock, S., Dahan, M., Mueller, K., and Sutton, D. (2001). The gridport toolkit: a system for building grid portals. In *Proceedings of the Tenth IEEE International Symposium on High Performance Distributed Computing*.

Índice por Autor