Um middleware para processamento bioestatístico em grades computacionais

Rodrigo Assirati Dias

Dissertação apresentada ao
Instituto de Matemática e
Estatística da
Universidade de São Paulo
para obtenção do título de
Mestre em Ciências

Programa: Ciência da Computação
Orientador: Prof. Dr. Roberto Hirata Jr.

— São Paulo, Agosto de 2009 —

Aos meus pais Ademir e Suely.
Aos meus irmãos Larissa e Vinicius.
A todos meus amigos ...
...e à Simone, com muito amor.

Agradecimentos

Agradeço aos meus pais pelo amor incondicional e por terem me dado a oportunidade de chegar até aqui, e aos meus irmãos que dividiram comigo quando mais precisei.

Às minhas tias e tios, primas e primos, sobrinhos e sobrinhas pelo apoio e presença em minha vida.

Ao *Roberto Hirata Jr.*, professor, amigo, agradeço pela dedicação, confiança, paciência e pelos ensinamentos (acadêmicos ou não).

Ao professor *Alfredo Goldman* pelas inestimáveis contribuições ao trabalho.

Agradeço aos professores, *Feofiloff*, *Coelho*, *Jordão*, *Ronaldo*, *Siang* pelos ensinamentos e em especial ao *Fabio Kon* e ao *Francisco Reverbel* pelas críticas e contribuições.

Ao *Lucas Augusto Meyer Perin* e *Eric Soares Costa* pela inspiração e apoio.

Ao *Alexandre Gomes Ferreira*, amigo e irmão, por toda ajuda e por me proporzionar a alegria de sua amizade.

A *Paula Akemi Nishimoto* pelas grandes contribuições no início deste trabalho.

A *Gisele Craveiro da Silva* pelas contribuições e aos demais professores de graduação por me darem subsídios para ter chegado até aqui.

Aos meus amigos *Alberto Lopes*, *Antonino Bossolan*, *Arthur Queiroz*, *César Honda*, *Cínthia Siqueira*, *Chico Takano*, *Fernanda Garzon*, *Fredericos*, *Fúlvio Andrade*, *Japa* ,*João Paulo Legat*, *Kimi Navai*, *Milton Goya*, *Raiji Takano*, *Rodolpho Koch*, *Samara Goya*, *Sérgio Queiroz*, *Silvério*, *Tatiana Beck* e a todos aqueles que não foram citados, mas que estão presentes em meu coração. Agradeço pela amizade, respeito e por me proporcionarem momentos de diversão quando eu mais precisava.

Ao Colégio Rainha da Paz pelo apoio e confiança constantes e às Irmãs Dominicanas pelo carinho, pela torcida.

Finalmente, um agradecimento mais que especial à minha esposa *Simone* pela paciência, amor e apoio constante.

Resumo

Neste trabalho é apresentado o Middle-R, um *middleware* de nível de usuário para computação estatística. O Middle-R foi desenvolvido para distribuir R (uma linguagem e ambiente para estatística) em estações de trabalho que rodam Windows. O Middle-R utiliza o Alchemi, um *middleware* de grade desenvolvida utilizando a tecnologia Microsoft .NET na Universidade de Melbourne . A solução implementada utiliza o Alchemi como um *middleware* de baixo nível faz com que seja possível executar scripts R em um ambiente de grade. A solução foi testada em dois ambientes distintos: os laboratórios didáticos de uma faculdade e de uma escola. Uma aplicação real de bioinformática foi utilizada como prova de conceito para testar a estabilidade e o desempenho da solução. A aplicação é uma tarefa combinatorial de encontrar marcadores moleculares de poucos genes para classificar corretamente amostras de leucemia linfoblástica aguda contra leucemia mielóide aguda.

Palavras-chave: computação em grade, middleware, Alchemi, R.

Abstract

In this work we present Middle-R, a user-level middleware for statistical computing. It has been developed to distribute R (a language and environment for statistics) in commodity computers running Microsoft Windows. It uses Alchemi, a grid middleware developed using Microsoft .NET technology in the University of Melbourne. The implemented solution uses Alchemi as a low-level middleware and makes it possible to execute R scripts in a grid environment. The solution has been successfully implemented and tested in two different environments: the didactic laboratories of a college and of a high school. A real bioinformatics applicarion was used as a proof of concept to test the stability and performance of the solution. The application is a combinatorial task for finding molecular markers of a few genes to classify correctly samples of acute lymphoblastic leukemia against acute myeloid leukemia.

Keywords: grid computing, middleware, Alchemi, R.

Sumário

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Introdução

A ciência frequentemente nos desafia com grandes problemas computacionais que exigem cada vez mais recursos para que possam ser explorados. Experimentos de Matemática, Física, Biologia e Química frequentemente exigem modelagens e simulações de muitas variáveis, mineração e análise de grande quantidade de dados, coleta e armazenagem de um grande volume de informações, entre outros exemplos de demanda por poder computacional. A Ciência da Computação tem tentado cada vez mais oferecer ferramentas melhores para lidar com esta grande demanda. Um destas ferramentas é o modelo de Computação em Grade [24], proposto formalmente em 1998 por Ian Foster e Carl Kasselman, mas utilizado de forma experimental desde 1995 [22]. Existem muitas definições do que é a computação em grade e sua abrangência. De maneira geral, neste trabalho trataremos a computação em grade como arquitetura e sistemas que permitem agregar, coordenar e compartilhar recursos de computação heterogêneos. Estes recursos podem ser locais, ou distribuídos ao longo de um ou mais domínios administrativos interligados por infraestrutura de rede.

Entre os benefícios de se adotar um modelo de computação que agregue recursos computacionais de forma colaborativa estão o aproveitamento dos recursos ociosos tanto de supercomputadores como de servidores e estações de trabalho, abordagem que motivou o surgimento das grades oportunistas. Além do melhor aproveitamento de recursos, pode-se citar como benefícios também a possibilidade de colaboração entre equipes de trabalho distantes geograficamente, que podem trabalhar não só compartilhar dados como também compartilhar recursos computacionais, manipulando-os remotamente. O custo também é um forte argumento a favor deste modelo computacional, uma vez que pode-se conseguir um bom poder computacional utilizando computadores mais baratos trabalhando de forma cooperativa.

1.1 *Middlewares* de Grade

Geralmente o modelo de computação em grade é implementado em termos de *software* por meio de um *middleware*. Um *middleware* é a camada intermediária de *software* que promove a interligação de aplicações, fornecendo serviços básicos para os componentes do sistema. Segundo Rajkumar Buyya [8], do ponto de vista das grades computacionais, um *middleware* de grade é responsável pela a agregação e coordenação dos recursos computacionais tornando-os virtualmente uma mesma unidade de computação. O papel principal dos *middlewares* de grade é operar sobre os diversos sistemas operacionais, arquiteturas e redes de comunicação, abstraindo seus detalhes e facilitando o desenvolvimento de aplicações para a grade. Buyya sugere uma arquitetura de grade em quatro camadas [8], sendo a camada dos recursos computacionais (*Grid Fabric*) a de mais baixo nível contemplando o *hardware*, sistema operacional, rede, etc., a camada de *middleware* de baixo nível (*Core Middleware*) responsável pela agregação e coordenação dos recursos e pelos serviços básicos como segurança, persistência etc., a camada de *middleware* de nível de usuário (*User-Level Middlware*) que se aproveita dos serviços e interfaces da camada de *middleware* de baixo nível para abstrair detalhes específicos da construção da grade, geralmente fornecendo recursos como bibliotecas, APIs e escalonadores de tarefas para a camada mais alta, chamada de camada das aplicações (*Applications and Portals*) que contém as aplicações que serão executadas na grade.

Existem hoje diversos exemplos de middleware de baixo nível, dentre principais temos o Globus [23], Legion [35], Condor [40] e o Integrade [33].

1.2 Motivação

Assim como em outras áreas da ciência, existem grandes problemas computacionais em bioinformática, por exemplo: análise de sequenciamento de DNA, busca por classificadores moleculares de poucos genes para prever o câncer ou para se verificar a confiabilidade de testes estatísticos (que pode envolver o uso de uma técnica de estatística conhecida como "*bootstrap*"). Normalmente o tempo de processamento dessas tarefas, em um computador de alta performance, é medido em semanas. O modelo de computação em grade se encaixa muito bem para facilitar este processamento, principalmente quando a análise pode ser quebrada em milhares de pequenos processos independentes com uma pequena variação dos parâmetros de entrada, um tipo de modelo que é conhecido por computação "embaraçosamente paralela" [26].

O Núcleo de Pesquisas em Bioinformática (BIOINFO) da USP possui vários grupos de pesquisa, muitos deles nas áreas de análise de dados de expressão gênica, redes de reg-

ulação gênica e bioinformática estrutural. Todas estas áreas possuem tarefas que exigem o uso intenso de poder computacional e que poderiam usufruir de uma infraestrutura de computação de alto desempenho. Além disto, estes grupos frequentemente colaboram com outros laboratórios nacionais e internacionais que são referência em pesquisas biológicas e que também poderiam usufruir de tal infraestrutura. Em um cenário ideal, os ciclos de computação excedentes de uma determinada instituição poderiam ser cedidos à estes grupos, ou então trocados com base em algum modelo econômico [12].

Aplicações científicas distribuídas não são novidade para a plataforma Unix, onde existem várias histórias de sucesso utilizando principalmente MPI [17, 54] e PVM [30]. Por outro lado aplicações científicas distribuídas sobre Windows são difíceis de achar e, uma vez que esta é uma das restrições de aplicação deste projeto, foi iniciado um projeto próprio de grade utilizando a estrutura tecnológica de um de nossos colaboradores, a Faculdade SENAC de Ciências Exatas e Tecnologia.

O SENAC nos disponibilizou inicialmente 25 estações de trabalho de um de seus laboratórios didáticos para a realização de um teste piloto em seu campus da Lapa, com a possibilidade de extensão para as outras cerca de 200 estações distribuídas por outros laboratórios. Havia ainda uma abertura para a utilização das mais de 900 estações de seu mais novo campus em Santo Amaro. Este último cenário, poderia colocar o projeto dentre os 50 primeiros colocados na classificação TOP500 Supercomputing Sites [16].

O projeto teve inicialmente quatro condições de contorno: (1) praticamente todo parque de máquinas do Senac utiliza Windows, assim a grade também deveria utilizar o mesmo sistema, (2) a solução deveria rodar sem comprometer as atividades diárias dos usuários, (3) o software de grade deve se comunicar facilmente com outros softwares utilizados em nossas análises e (4) a grade deveria ser capaz de se integrar com outras arquiteturas de grades computacionais. Para atender estas condições de contorno, inicialmente foi considerada a possibilidade de criar uma solução própria de grade, sob a plataforma .NET que seria capaz de fornecer o instrumental necessário para que a solução atendesse todas as condições acima, mas então descobrimos um middleware de grade também desenvolvido em .NET chamado Alchemi que nos ajudava nas condições (1) e (4), permitindo que nos concentrássemos nos problemas de bioinformática.

A possibilidade de ter uma implementação de grade com um grande número de computadores em uma plataforma fortemente baseada em seus produtos, fez com que a Microsoft se interessasse pelo projeto. Por intermédio do Dr. Carlos Hulot, foi destinada uma verba para a contratação de estagiários que trabalhariam neste projeto dentro do SENAC. Estes estagiários estariam vinculado ao Centro de Inovação [1], uma empresa júnior de capacitação e desenvolvimento de projetos mantida por uma parceria entre SENAC e Microsoft. No entanto, depois de cerca de um ano de negociações a oferta das estações para compor a grade foi retirada por conta de disputas políticas internas do SENAC, fazendo

com que a verba para contratação dos estagiários não pudesse ser aplicada no Centro de Inovação.

Com isto buscamos outras instituições que pudessem participar da parceria cedendo estações de trabalho para a realização do projeto. Conseguimos o apoio do Colégio Rainha da Paz, que nos permitiu utilizar seus dois laboratórios didáticos para testes bem mais modestos. Foram nos cedidas 40 estações com características semelhantes àquelas encontradas no SENAC.

1.3 Alchemi

O Alchemi é um *middleware* de grade de baixo nível totalmente desenvolvido sob a plataforma Microsoft .NET na Universidade de Melbourne como parte do projeto Gridbus [11]. Para a condição (2), o caminho natural seria fazer o Alchemi funcionar como um serviço do Windows, fazendo com que o agente Alchemi fosse fácil de instalar, iniciar e encerrar e sem a necessidade de interação com usuário. Como o Alchemi é um projeto de código aberto desenvolvido sob a licença LGPL da GNU, não tivemos problemas em adaptá-lo às nossas necessidades. Essa alteração, mais tarde, foi submetida à comunidade de desenvolvedores do Alchemi e foi incorporada à distribuição oficial.

1.4 O Ambiente R

Um importante ambiente para se fazer bioinformática nos dias de hoje é o *software* R, uma evolução de código aberto do ambiente e linguagem S. O R foi compilado para diversas plataformas, como: Linux, Windows, MacOS e Solaris e pode ser integrado com diversas linguagens (C, Fortran, etc.), embora não tenha suporte para a plataforma .NET. Entretanto pode se comunicar com outros processos utilizando tecnologia COM através de um pacote desenvolvido pela comunidade, chamado RCOM.

Para solucionar nosso problema de usar R em um ambiente de grade, nós propusemos o middle-R, um *middleware* em nível de usuário que usa o Alchemi como *middleware* de baixo nível tornando possível executar *scripts* R em ambiente de grade. Essa camada foi implementada e utilizada em nossas atividades de bioinformática.

Na época do desenvolvimento deste trabalho, existiam muitos pacotes para paralelizar processos do R [51, 55, 39]. Todos eles foram desenvolvidos e tem sido implementados em Linux e podem ser adaptados para Windows com algumas limitações. Contudo, funcionam melhor em pequenas redes dedicadas (um *cluster*, por exemplo) do que em grandes infraestruturas e requerem que o usuário gerencie os participantes da rede e a distribuição

da carga de trabalho de dentro do R. Não existe mecanismo de *logging*, resubmissão de tarefas e outras funcionalidades geralmente encontradas em *middlewares* de grade.

O uso do R especificamente em grades computacionais era incipiente quando este projeto se iniciou, porém houveram alguns avanços nesta direção.Durante o desenvolvimento deste trabalho, tivemos contato com trabalhos que utilizam o R sobre o Condor [18, 40], um *middleware* para grades oportunistas. Porém, até a data da escrita deste texto não havia ainda artigos ou relatórios técnicos sobre este cenário. Mais recentemente foi nos apresentada uma solução alternativa de computação voluntária para aplicações interpretadas utilizando a infraestrutura do BOINC [34]. Uma implementação do uso do R nesta última solução já está em desenvolvimento, porém a comparação com resultados obtidos com o Alchemi fará parte dos trabalhos futuros.

1.5 Contribuições do Trabalho

Este trabalho contribuiu para o desenvolvimento do Alchemi, conforme citado na seção anterior. Ao transformar o Alchemi em um serviço do Windows, foi criada uma nova versão do Executor (componente responsável por executar as tarefas nos nós da grade) onde sua interface gráfica foi extraída e suas funcionalidades colocadas em uma aplicação à parte que tem como função configurar e controlar o agora serviço do Executor. O mesmo foi feito com o Gerente (componente responsável por coordenar a distribuição e execução das cargas de trabalho para as estações) , onde foram criadas as aplicações `ManagerService` e `ManagerServiceController`. Estas aplicações passaram a fazer parte da distribuição oficial do Alchemi desde sua versão `1.04`. Uma outra contribuição menos significativa foi a implementação de uma versão *web* do aplicação `Console`. Através desta aplicação, pode-se monitorar a capacidade da grade e os estados de cada nó, além de submeter e monitorar a execução de aplicações da grade. A versão atual do `Console` pode ser executada remotamente, mas deve ser executada em ambiente Windows, através da instalação do programa. Havia a necessidade de monitorarmos a aplicação em horários onde normalmente já não se tinha mais acesso físico aos laboratórios onde as aplicações eram executadas. Foi feita então uma migração da aplicação para o ambiente *web*, na plataforma ASP.NET, em conjunto com o então aluno de graduação do BCC do SENAC, Leonardo Rombesso.

Do ponto de vista bibliográfico, esta dissertação poderá orientar futuras implementações da tecnologia prospota, em cenários onde o uso do R e a necessidade de poder computacional se fazem necessários. A mediação das tecnologias envolvidas não é trivial e este texto poderá servir para que algumas dificuldades já encontradas sejam mais facilmente ultrapassadas em projetos que utilizem pelo menos parte da solução proposta.

Ainda sob o ponto de vista bibliográfico, o seguinte artigo foi publicado condensando a solução proposta neste trabalho e descrevendo os resultados obtidos utilizando a aplicação de marcadores gênicos descrita no Capítulo 5:

- Middle-R - A User Level Middleware for Statistical Computing. Rodrigo Assirati Dias, Alfredo Goldman e Roberto Hirata Jr. Apresentado no VII WCGA (*Workshop on Grid Computing and Applications*) em Maio de 2009.

1.6 Organização da Dissertação

Esta dissertação se inicia com uma introdução onde é delineado o contexto no qual este trabalho está inserido. A introdução começa com uma pequena definição do que é a computação em grade e seus benefícios, seguida de uma breve discussão do que são *middlewares* de grade e o seu uso no contexto desta dissertação. Ainda na introdução, é exposta a motivação para a realização deste trabalho do ponto de vista do início do projeto, quais dificuldades que influenciaram suas alterações de escopo e a introdução de suas principais ferramentas.

No Capítulo 2, é descrito o Arcabouço Alchemi de forma mais detalhada. Na Seção 2.1, é descrita sua arquitetura interna e seus componentes tanto do ponto de vista de seus papéis quanto do ponto de vista de sua representação na API do Alchemi. a Seção 2.2, são descritas as configurações de implantação nos cenários típicos de uma grade Alchemi. Uma atenção maior é dada na Seção 2.3 aos modelos de programação de aplicações de grade, já que estes influenciaram de forma direta o desenvolvimento deste trabalho.

O Capítulo 3 é destinado à descrição das tecnologias nas quais a solução proposta está baseada. Esta descrição se faz necessárias para que possa se discutir as alternativas para se acessar o motor de computação do R a partir de uma aplicação .NET - plataforma sobre a qual o Alchemi foi construído. A Seção 3.1 introduz o ambiente R e sua relevância para a bioestatística, seguida da descrição do RCOM, pacote desenvolvido para facilitar a interface entre o R e aplicações COM. Na Seção 3.2, é descrito o modelo COM, base do RCOM. Na Seção 3.3 é detalhado o Arcabouço .NET e na Seção 3.4 os mecanismos previstos pelos criadores do arcabouço para sua interoperabilidade com o modelo COM.

A solução proposta neste trabalho está descrita no Capítulo 4, que se inicia com a retomada da discussão entre as formas de interoperabilidade entre o modelo COM e o Arcabouço .NET, desta vez aplicadas ao acesso ao R. Em seguida, na Seção 4.2 é descrita a solução proposta para utilização no modelo de *jobs* do Alchemi. A solução proposta para o modelo de *threads* está contemplada na Seção 4.3, seguida da discussão sobre as vantagens e desvantagens dos dois modelos. A Seção 4.5 apresenta uma alternativa para

o acesso ao R a partir do código .NET. O capítulo termina com a apresentação de uma aplicação exemplo, mas que foi utilizada para auxiliar o ambiente de gerenciamento das instalações do R nos laboratórios.

Foram realizados testes para comprovar a viabilidade e o desempenho da solução proposta. Foi escolhida uma aplicação real de bioinformática sobre a qual foram realizados testes em cenários diferentes. Os testes, seu cenário e seus resultados experimentais são discutidos no Capítulo 5. Nas seções 5.3 e 5.4 são apresentados os resultados da execução da aplicação com diferentes tamanhos de grade nos modelos de *jobs* e *threads* respectivamente. Foram realizados testes de estabilidade, descritos na Seção 5.4. Este capítulo termina com uma discussão sobre os resultados obtidos.

O Capítulo 6 contempla as conclusões deste trabalho, assim como a descrição dos trabalhos futuros e alguns assuntos em aberto que poderiam ser objetos de futura investigação.

CAPÍTULO 2

O arcabouço Alchemi

O Alchemi [4][42] é um arcabouço desenvolvido na Universidade de Melbourne por Akashay Luther, orientado pelo Dr. Rajkumar Buyya, que fornece componentes que permitem a criação de grades computacionais através de agregação de recursos heterogêneos de uma rede, e uma API para o desenvolvimento de aplicações habilitadas ao uso destas grades.

Seu desenvolvimento foi motivado por alguns fatores como: o crescente interesse de empresas comerciais na computação distribuída, em especial no modelo de computação em grade; a arquitetura comumente encontrada no ambiente empresarial e a falta de soluções de computação distribuídas para esta plataforma. Segundo seus criadores [41], a criação de uma solução de grade computacional com base na plataforma Windows seria determinante para o sucesso da adoção do modelo em ambientes empresariais, devido à grande adoção da plataforma neste tipo de ambiente e, na época de sua criação, faltavam soluções de grades computacionais orientadas a objeto para este cenário.

O Alchemi foi então desenvolvido utilizando a plataforma .NET para se beneficiar de sua integração com o sistema operacional e para explorar algumas facilidades desta plataforma como sua API para comunicação remota inter-processos chamada de *.NET Remoting* [43] e sua biblioteca integrada para serviços web [6]. O arcabouço foi organizado na forma de uma arquitetura em camadas totalmente orienta a objetos. Esta arquitetura engloba entre outras coisas, uma API que fornece um ambiente de programação orientado a objetos para aplicações Alchemi, um modelo de tarefas para execução de aplicações legadas, interfaces na forma de serviços web para integração com outras plataformas de grade, um ambiente de execução para aplicações da grade e bibliotecas que contém classes que representam os recursos e componentes de execução, fornecendo meios de controle e monitoramento do estado da grade e suas aplicações. A figura ?? mostra uma configuração desta arquitetura para um ambiente de grade na forma de um aglomerado de estações de

trabalho. Embora sua arquitetura seja abrangente, o Alchemi possui algumas limitações, que serão discutidas na seção 2.4.

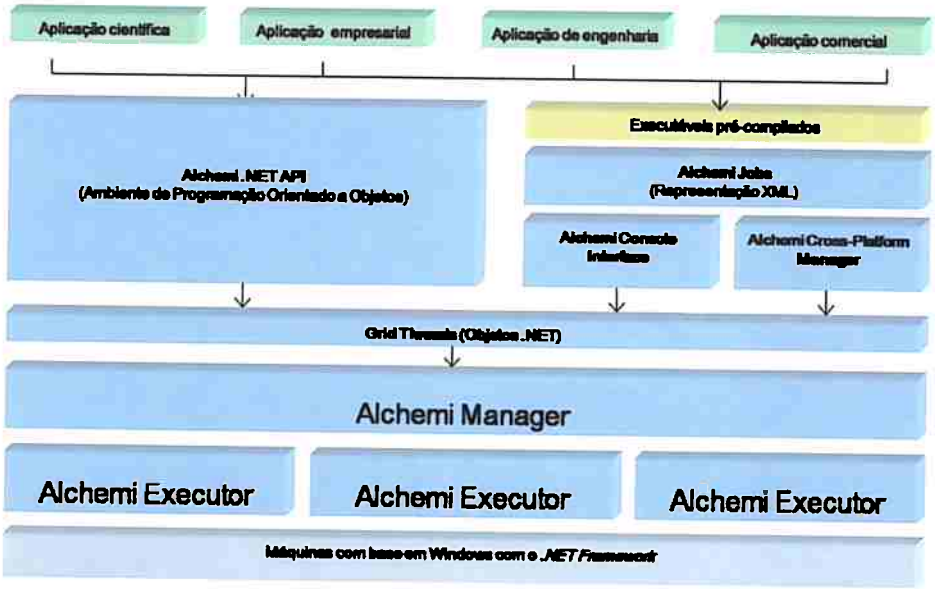

Figura 2.1: A arquitetura em camadas do Alchemi para o ambiente de computação em grade de aglomerado, adaptada de [42], pág. 5.

2.1 Componentes de uma grade Alchemi

Os componentes da arquitetura do Alchemi possuem papeis bem definidos e cada um deles é representado por uma classe em sua API. Em sua versão `1.0.6`, possui os seguintes componentes em sua arquitetura:

2.1.1 Manager (O gerente)

O *Manager* é o componente que engloba as mais importantes funcionalidades de uma grade Alchemi, como a criação, execução e gerenciamento de uma aplicação da grade, o agendamento de suas *threads* de grade (*grid threads*) para execução, controle e monitoramento de recursos e gerenciamento de credenciais de segurança. Um *Manager* é uma instância da classe `GManager` e expõe a interface pública `IManager` através da qual outros componentes da arquitetura têm acesso a serviços de gerenciamento. A criação de uma aplicação na grade, por exemplo, é feita pelo proprietário da aplicação, representado por

uma instância de `GOwner`, através da execução do método `CreateApplication(...)` em um `GManager` e do registro das *threads* de grade que compõe esta aplicação, através do método `SetThreads(...)`. A partir daí o controle da aplicação é feito através de métodos como `VerifyApplication(...)`, `GetApplicationState(...)` e `StopApplication(...)`. Cada gerente é responsável por controlar o estado dos executores conectados a ele através de uma coleção de referências para estes executores, assim como todas as tarefas que foram submetidas, fazendo o agendamento desas entre os executores disponíveis.
Além das funcionalidade de gerenciamento da grade descritas, um nó gerente também pode ser um nó executor conectado a um nó gerente de mais alto nível. Para este gerente de mais alto nível, este seria um executor com maior capacidade de processamento de *threads* da grade. Para suportar esta funcionalidade de grade hierárquica, a interface `IManager` herda a interface `IExecutor`. Este cenário de um ambiente hierárquico está descrito em maior detalhe na seção 2.2.

2.1.2 Executor

Um nó *Executor* de uma grade Alchemi é o responsável pela execução das threads de grade enviadas pelo *Manager*, caso possua credenciais para tal execução. Na arquitetura do Alchemi, a classe que representa o *Executor* é a `GExecutor`, que por sua vez implementa a interface `IExecutor`, que define métodos para controle e execução de *threads* de grade. A classe `GExecutor` implementa as funcionalidades da interface `IExecutor` através da instanciação de algumas classes de trabalho, divididas por funcionalidade e que são executadas em *threads* separadas no nó. Assim, enquanto uma *thread* de grade está sendo executada dentro de uma *thread* da aplicação, outras *threads* podem ser disparadas para realizar tarefas administrativas e de monitoramento. As classes de trabalho contidas no executor são a `HeartBeatWorker`, `ExecutorWorker` e `NonDedicatedExecutorWorker`. A `HeartBeatWorker` é a classe responsável por notificar o *Manager* ao qual este *Executor* está conectado de seu estado a cada período de tempo estipulado, enquanto as classes `ExecutorWorker` e `NonDedicatedExecutorWorker` são utilizadas para executar as *threads* de grade nos modelos de conexão dedicado e não dedicado. Enquanto apenas uma instância da classe `HeartBeatWorker` pode existir em uma aplicação *Executor*, várias instâncias das classes `ExecutorWorker` ou `NonDedicatedExecutorWorker` podem existir simultaneamente, dependendo das configurações da aplicação e da estação que é executada.
A arquitetura do Alchemi suporta ainda dois modelos de execução, o modelo de execução dedicado e o não dedicado. Em ambos o registro do *Executor* em um *Manager* é feito a partir do *Executor*, através do método `Executor_ConnectDedicatedExecutor(...)` da `IManager` para o modelo dedicado, e através do método `Executor_ConnectNonDedicatedExecutor(...` da mesma interface para o modelo não dedicado. No modelo não dedicado, o *Executor* se conecta ao seu *Manager* em um regime de voluntariado, solicitando e retornando threads

de grade à medida que um estado de ociosidade é detectado e notificando constantemente seu estado para o *Manager*. A detecção do estado de ociosidade é feita através de um componente instalado como protetor de tela do nó executor, nos mesmo moldes de outras grades desktop [7, 38] e toda a comunicação se dá através de chamadas a métodos da interface `IManager` pelo *Executor*. No modelo dedicado, a comunicação se dá em duas vias, e o *Manager* tem acesso as funcionalidades de um *Executor* através de sua interface pública `IExecutor`, através da qual pode monitorar e controlar o estado do *Executor* e a execução de suas *threads* de grade.

2.1.3 *Owner* (Proprietário)

Aplicações de uma grade Alchemi são criadas e executadas em um nó *Owner*, que tem como responsabilidades principais a submissão das *threads* de grade desta aplicação para execução em um nó *Manager* e o monitoramento do estado da execução destas *threads*. Ao criar uma aplicação, através do método `CreateApplication(...)`, o usuário recebe um identificador desta aplicação no nó *Manager* em que foi criada e através deste identificador, diversas propriedades desta aplicação podem ser configuradas através de métodos específicos . Dentre as propriedades de uma aplicação que são determinadas e gerenciadas pelo seu proprietário através destes métodos, estão a coleção de *threads* e o inventário de dependências desta aplicação .

2.1.4 *Cross-Plataform Manager* (Gerente Multi-Plataforma)

O Gerente Multi-plataforma é responsável pela integração de grades Alchemi com aplicações legadas e grades construídas em outras plataformas, desde que estas suportem a comunicação através de serviços web [6]. A integração é feita através de uma interface pública disponível na forma de serviços web e dela, é possível receber tarefas independentes de grade que são então transformadas em *threads* do Alchemi e gerenciadas pelo nó *Manager* ao qual o gerente multi-plataforma está acoplado.
Devido às restrições dos binários que serão executados neste tipo de cenário (geralmente aplicações legadas completas e não objetos próprios da plataforma .Net) apenas o modelo de *jobs* (descrito na seção 2.3) é suportado pelo gerente multi-plataforma. Outra limitação óbvia é que os binários que serão executados deverão ser compatíveis com a plataforma de execução da grade Alchemi para qual se está submetendo as tarefas.
A interface de serviços web é bem simplificada e possui poucos métodos, apenas o suficiente para a criação e consulta de jobs e tarefas. A criação de um *job* por uma aplicação externa, por exemplo, é feita em poucos passos: A aplicação solicita a criação de uma tarefa através do método `SubmitTask(...)` passando como parâmetro um arquivo XML

contendo a tarefa e a serialização dos *jobs* que a contém, recebendo de volta o identificador desta tarefa no gerente para qual esta foi submetida. Alternativamente, a criação de tarefas e *jobs* pode ser feita utilizando-se a interface: É feita uma chamada ao método `CreateTask(...)` que devolve o identificador da tarefa recém criada; Em seguida, a aplicação pode utilizar o método `AddJob(...)` para criar um novo *job*, passando o identificador da tarefa ao qual este job pertence e um arquivo XML contendo o *job* que será executado; Em ambos os casos, pode-se utilizar o método `GetFinishedJobs(...)` que devolve um arquivo XML contendo os *jobs* que terminaram de ser executados ou o método `GetJobState(...)` que devolve o status de execução de um determinado *job*. Caso o status de um *job* esteja marcado como "*Failed*", pode-se obter detalhes da falha com o método `GetFailedJobException(...)`. Por fim, tarefas e *jobs* podem ser cancelados através dos métodos `AbortTask(...)` e `AbortJob(...)` respectivamente.
Embora a própria tecnologia de serviços web permita a interconexão com qualquer aplicação ou *middleware* que suporte a tecnologia, a interface do gerente multi-plataforma bem como o formato do arquivo de tarefa não seguem um padrão específico (como o OGSI [21] ou o WSRF [10] propostos na OGSA [25]), requerendo que o usuário de outras plataformas que queira utilizar uma grade Alchemi como um nó de uma grade global escreva código específico para este fim.

2.2 Configurações de implantação

Os componentes do Alchemi podem ser combinados de formas diferentes permitindo a construção de configurações diferentes de grade, como grades na forma de aglomerados de estações locais, grades de multi-aglomerados, grades globais ou grades multi-plataforma.

- **Aglomerado**

  A configuração mais simples que se pode obter para uma grade Alchemi é a de um aglomerado de computadores que, nesta definição, é qualquer configuração onde temos apenas um nó gerente, controlando um ou mais nós executores, independente do modo de execução. Para submeter aplicações para execução nesta configuração, basta que o proprietário da aplicação conecte-se ao nó gerente e submeta a aplicação para execução.
  Embora o tipo mais comum de aglomerado geralmente envolva computadores dispostos em uma rede local dentro de um mesmo domínio administrativo utilizando o modo de execução dedicado, ilustrado na Figura 2.2, podemos obter algumas variações desta configuração.
  Por exemplo, pode-se utilizar o modo de execução voluntário em parte dos computadores que estiverem conectados através da Internet , desde que a porta de gerenci-

amento esteja aberta para o nó gerente em todos os dispositivos de roteamento e filtragem intermediários.

- **Multi-Aglomerado**

  Esta configuração se caracteriza por um ou mais nós gerentes de aglomerados conectados a um nó gerente de mais alto nível, assumindo o papel de executores para este gerente, conforme o exemplo da Figura 2.3. Em sua configuração mais usual, montam-se aglomerados independentes em cada rede local (em um mesmo domínio administrativo ou não) e quando existe a necessidade conectam-se os gerentes destes aglomerados a um gerente de alto nível que vai coordenar as execuções em todo o ambiente.

  Conforme descrito na seção 2.1.1, o *Manager* implementa a interface `IExecutor` e, por isto, contém métodos para conexão com outros gerentes além de métodos para notificação e recepção de threads de grade, fazendo com que um gerente que estiver em alguma camada intermediária da hierarquia possa tanto receber a submissão direta de uma aplicação por parte de um proprietário quanto receber threads de grade submetidas por outro gerente de mais alto nível. Pelo gerente assumir o papel de executor com todas as suas funcionalidades, os dois modos de conexão estão disponíveis, tanto o modo dedicado, onde o gerente monitora o estado do executor (que neste caso é outro gerente) e quando detecta um estado de ociosidade submete *threads* de grade disponíveis para execução, quanto o modo voluntário onde o executor requisita *threads* voluntariamente sempre que se encontrar em estado de ociosidade.

  Nos dois casos, tanto ao solicitar *threads* de grade, quanto ao receber *threads* para agendamento em seus executores, a prioridade de execução dessas threads é decrescida em uma unidade no momento do agendamento. Enquanto as threads submetidas por um proprietário diretamente ao gerente sempre são agendadas com prioridade máxima, fazendo com que *threads* de grade submetidas por um gerente de mais alto nível, tenham sempre sua execução agendada com prioridade menor frente às *threads* de uma aplicação submetidas diretamente a este gerente, fazendo com que e a medida que as *threads* são passadas a gerentes de menor nível da hierarquia, vão tendo sua prioridade decrescida.

  Obviamente, as aplicações podem ser submetidas pelos proprietários a qualquer gerente do ambiente, porém a execução será agendada somente nos executores que este gerente controla, estando hierarquicamente abaixo dele.

- **Grades Multi-plataforma**

  Este cenário é formado por uma grade que siga qualquer uma das duas configurações vistas anteriormente tendo como diferencial um Gerente Multi-Plataforma associado a um dos gerentes desta grade, fazendo com que esta se torne um nó de uma grade

global gerenciada por algum outro *middleware* capaz de se comunicar através de uma interface de serviços web.

Conforme discutido na seção 2.1.4, a interface de serviços web fornecida pelo gerente multi-plataforma não segue nenhum padrão específico de interface para serviços de grade, fazendo com que grades que necessitem se comunicar com um nó Alchemi escrevam chamadas específicas para sua interface ou que utilizem um *broker* compatível com o Alchemi. Um exemplo de *broker* compatível com o Alchemi é o Gridbus [11], projeto que também é desenvolvido na Universidade de Melbourne e também sob coordenação do Dr. Rajkumar Buyya.

De forma semelhante com a que acontece na arquitetura multi-aglomerado, o gerente ao qual o gerente multi-plataforma está conectado pode receber *threads* de grades de um gerente de nível superior, agendando estas *threads* dentre os seus executores com prioridade reduzida, assim como pode também receber aplicações submetidas diretamente a ele através do proprietário da aplicação, agendando as *threads* que constituem esta aplicação com prioridade máxima em seus nós executores, e também receber *threads* de outras aplicações através do gerente multi-plataforma, que neste caso são agendadas apenas nos executores que este gerente controla e com a mesma prioridade recebida.

As Figura 2.4 e 2.5 mostram exemplos destas arquiteturas.

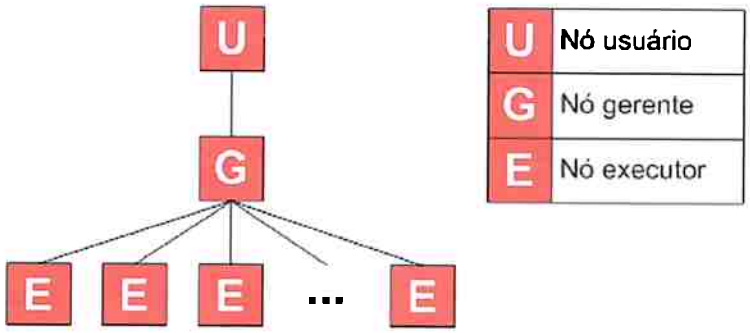

Figura 2.2: Representação de uma grade Alchemi configurada como um aglomerado local.

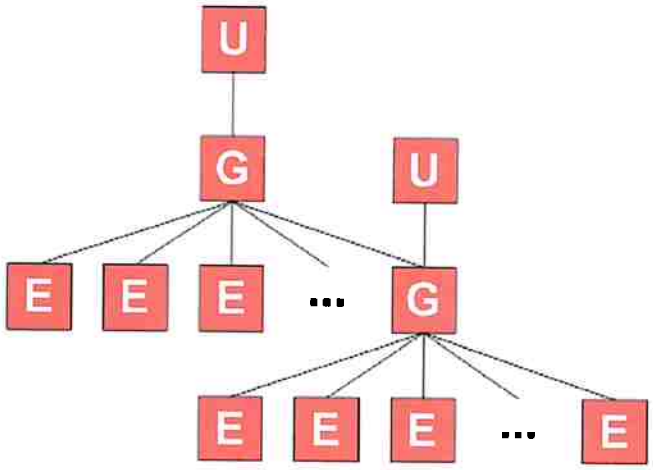

Figura 2.3: Representação de uma grade Alchemi configurada como um multi-aglomerado.

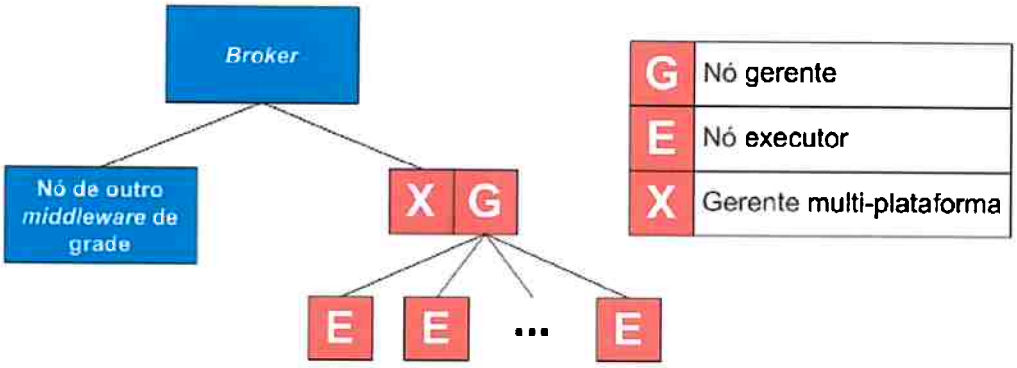

Figura 2.4: Representação de uma grade Alchemi configurada como um nó de uma grade global. Neste caso, tarefas são recebidas através de um *broker* que coordena outras grades.

2.3 Modelos de aplicação

2.3.1 O modelo de *threads* de grade

Através de sua API, o Alchemi fornece um ambiente de desenvolvimento de aplicações de grade orientado a objetos que tem como principal vantagem a abstração dos detalhes da arquitetura da grade criando um ambiente semelhante à programação *multi-threading*. Assim como na programação *multi-threading*, uma aplicação Alchemi que utilize o modelo de *threads* de grade possui uma *thread* principal que contém o código de coordenação da aplicação e das *threads* que a compõe. Este código, assim como o código da aplicação é executado localmente em um nó usuário pelo proprietário da aplicação e não em um nó executor ou gerente e por isto é referido algumas vezes na documentação do Alchemi

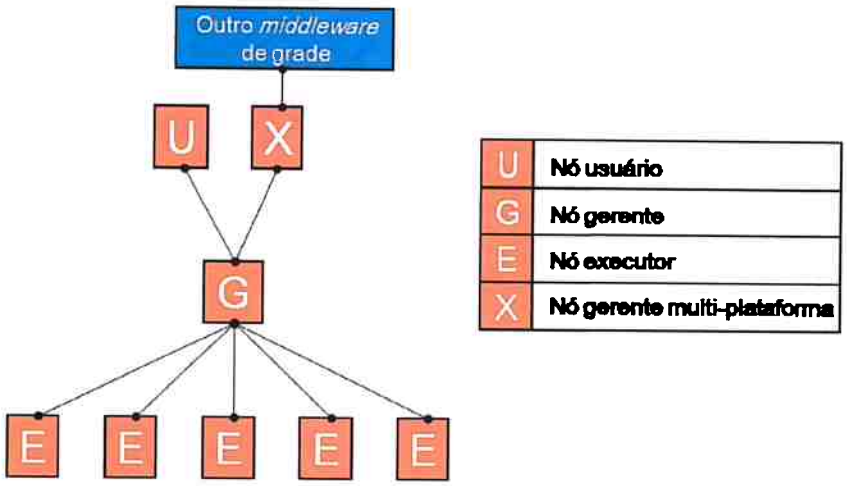

Figura 2.5: Representação de uma grade Alchemi configurada como um nó de uma grade global. Neste caso, tarefas são recebidas diretamente através de outras grades pela interface de serviços web.

como código local. Na API do Alchemi, a própria aplicação é responsável por serializar e transmitir as *threads* de grade que compõe esta aplicação para um nó gerente para agendamento e execução nos nós executores. Assim,como o código que forma cada *thread* é executado remotamente pela grade, é referido como código remoto.
O código local é representado na API do Alchemi pela classe `GApplication`, enquanto o código remoto é representado pela classe `GThread`, que representam a aplicação de grade e a *thread* de grade respectivamente. A comunicação entre o código de coordenação da aplicação e o código remoto é feita através da assinatura de eventos específicos da aplicação, utilizando a estrutura de eventos do arcabouço .NET. de acordo com o padrão comportamental *Observer* [29].
Para criar uma aplicação típica do Alchemi, um programador deve: Criar no código local uma aplicação de grade, instanciando a classe `GApplication`; Informar à aplicação de quais classes ela depende (tipicamente as classes de *threads* e classes referenciadas por elas) através do atributo `Manifest`; Instanciar as *threads* e adicioná-las à coleção de *threads* da aplicação, representada pelo atributo `Threads`; Criar métodos para manipular os eventos de término de aplicação e término de *thread*, se necessário, e registrá-los nos eventos respectivos `ApplicationFinish` e `ThreadFinish`; Criar uma conexão com um nó gerente, representada por uma instância da classe `GConnection`, e associar ao atributo `Connection` da aplicação; Iniciar a execução da aplicação através do método `Start()`.
De forma similar a programação *multi-threading*, uma *thread* de grade pode ser qualquer classe marcada com a anotação `[Serializable]`, que herde a classe `GThread` e sobrescreva o método `Start()`. Ao ser serializada para o nó gerente e posteriormente para o

nó executor, seu método `Start()` é invocado ocasionando a execução desta *thread*. Após cada execução de *thread*, a aplicação notifica o usuário através dos métodos registrados nos eventos `ThreadFinish`, em caso de sucesso, e `ThreadFailed`, em caso de falha. Ainda em caso de falha, o usuário poderá consultar a exceção gerada, cuja referência é armazenada no atributo `RemoteExecutionException` da *thread*. Somente quando todas as *threads* forem finalizadas com sucesso, a aplicação então notifica o usuário através do evento `ApplicationFinish`.

```
[Serializable]
class VerificadorDePrimos : GThread
{
    public readonly int Candidato;
    public int Fatores = 0;

    public VerificadorDePrimos(int candidato)
    {
        Candidato = candidato;
    }

    public override void Start()
    {
        // conta o número de fatores de n de 1 até n
        for (int d=1; d<=Candidato; d++)
        {
            if (Candidato[d] == 0) Fatores++;
        }
    }
}
```

Figura 2.6: Um exemplo de uma *thread* de grade na API do Alchemi. Adaptado de exemplos do SDK do Alchemi.

2.3.2 O modelo de *jobs*

O modelo de *jobs* da arquitetura do Alchemi foi desenvolvido com o propósito de dar suporte a aplicações legadas, aplicações escritas em outras linguagens que não suportam a plataforma .NET e aplicações submetidas por outros *middlewares* de grade através da interface de serviços web do gerente multi-plataforma.

Apesar da terminologia diferente em sua documentação, o modelo de *jobs* segue o conceito de aplicações *Bag-of-Tasks* [53] ou aplicações completamente paralelas (embarrassingly parallel) [26], ou seja, aplicações formadas por tarefas independentes que não se comunicam e que podem ser executadas paralelamente em qualquer ordem, geralmente representando a execução de um mesmo comando porém com parâmetros diferentes. O modelo de *jobs* do Alchemi apresenta termos diferentes mas a semântica deste conceito se conserva. Uma aplicação da grade é representada por uma tarefa que por sua vez é formada por diversos *jobs*. A tarefa possui dependências, que geralmente são binários (aplicações e bibliotecas compiladas) necessários para sua execução nos nós executores e uma coleção de *jobs*. Cada *job*, possui um ou mais arquivos de entrada e um ou mais arquivos de saída utilizados para se passar parâmetros e receber resultados deste *job*, assim como o comando que deve ser executado quando este *job* for executado pelo nós executor. Ao enviar uma tarefa para um gerente, todas as dependências e arquivos de entrada são transmitidos ao gerente e repassados aos executores através de cópia binária sobre TCP/IP e quando a tarefa for terminada os arquivos de saída são devolvidos ao gerente da mesma forma.
O Alchemi prevê três tipos de entradas para a cargas de trabalho no modelo de jobs:

- A interface de console do gerente
  É a maneira mais simples de se submeter uma aplicação para uma grade Alchemi. A interface de console do gerente é uma aplicação que pode ser utilizada para monitorar aplicações e *threads* de grade sob responsabilidade de um dado gerente e através dela, pode-se também enviar tarefas sem que haja a necessidade de se criar um programa específico para este fim e sem que seja necessário o conhecimento de sua biblioteca. A tarefa que será carregada é especificada em um arquivo XML que contém todas as informações necessárias para descrever a tarefa e seus *jobs*. Conforme ilustrado na Figura 2.3.2, a tarefa é delimitada pela etiqueta `<task>` e deve conter a indicação de suas dependências dentro da etiqueta `<manifest>`, seguida pela descrição de todos os seus *jobs*. Estes por sua vez, são delimitados pela etiqueta `<job>` que contém um atributo determinando seu identificador (que deve ser único dentro de determinada tarefa), a descrição de seus arquivos de entrada e saída (informados nas etiquetas `<input>` e `<output>` respectivamente) e o comando que será executado, identificado pela etiqueta `<work>`.

- API de *jobs*
  O modelo de *jobs* do Alchemi também é suportado por sua API, que contém classes que permitem a escrita de aplicações capazes de criar, enviar e gerenciar tarefas e *jobs* em um nó gerente. Uma tarefa na API de *jobs* do Alchemi continua sendo representada por uma aplicação, pois além de serem conceitos equivalentes, compartilham da mesma estrutura. Assim como uma tarefa, uma aplicação também

tem uma coleção de dependências representada pelo atributo `Manifest`, conforme discutido na seção anterior. A diferença, porém, é que ao invés de adicionarmos instâncias da classe `GThread` à coleção `Threads` da aplicação, adicionamos objetos da classe `GJob` à mesma coleção. Isto é possível por polimorfismo, já que a classe `GJob` na verdade uma especialização da classe `GThread`. Além dos atributos e métodos da classe `GThread`, a classe `GJob` ainda possui uma coleção de dependências de entrada e uma de saída (`InputFiles` e `OutputFiles`), além de do atributo `RunCommand` que contém o comando que será executado quando este job for processado.

- Gerente multi-plataforma
  Conforme discutido na seção 2.1.4, um gerente também pode receber tarefas de outros *middlewares* de grade, através da interface de serviços web do gerente multi-plataforma. Pelo fato do SOAP [46], protocolo padrão de comunicação de serviços web, ter como base do formato de mensagens um documento XML, a especificação de uma tarefa através desta interface segue o mesmo modelo do arquivo utilizado para carga na interface de console do gerente. A diferença apenas está na forma como são transmitidos os arquivos indicados nas dependências da aplicação e nas entradas e saídas de cada job. Como o serviço web do Alchemi utiliza SOAP em cima da camada HTTP [20] para transporte, os arquivos que forem destinados ao gerente e posteriormente aos executores, têm de ser serializados e codificados como texto dentro da própria mensagem SOAP de forma similar como ocorre a codificação MIME [28] para anexos de mensagens de correio eletrônico. Mais uma vez o Alchemi tira proveito da integração do arcabouço .NET com serviços web, que faz todo o trabalho de codificação e decodificação e transmissão destas mensagens.

Apesar de ambos os modelos de desenvolvimento de aplicações de grade suportarem apenas o modelo de computação completamente paralela, ou seja, não suportarem comunicação entre *threads* de grade, o modelo de *threads* oferece maior versatilidade comparando-se com o modelo de jobs, pois oferece para a aplicação um controle maior sobre a execução das *threads* de grade, tornando possível implantar estruturas como pontos de sincronização, de checagem e controle transacional.

2.4 Limitações

Embora suas funcionalidades sejam adequadas para as aplicações que desenvolvemos ao longo deste trabalho, o Alchemi possui algumas limitações. Algumas delas foram superadas durante o desenvolvimento deste trabalho e envolveram modificações feitas no código do Alchemi outras porém fugiam do escopo ou requeriam grande esforço para que

fossem realizadas, sendo deixadas de lado por não impactarem nos resultados esperados. Uma limitação que deve ser ultrapassada para que o Alchemi possa ser utilizado em uma gama maior de aplicações paralelas é o suporte apenas ao modelo de aplicações completamente paralelas. Seria desejável que sua arquitetura suportasse a comunicação entre threads de grades permitindo a adoção de outros modelos de aplicações paralelas como BSP [52] e a escrita de bibliotecas que suportem aplicações de protocolos bem estabelecidos como MPI [54][17] e PVM [30].

A interface de serviços web, por sua vez, fornece ao Alchemi uma grande flexibilidade de interação com outras plataformas de grade, porém, para um melhor aproveitamento de suas funcionalidades poderia ser melhorada adotando-se um padrão aberto para serviços web de grade, como o proposto pela especificação Open Grid Services Architecture (OGSA) [25], que propõe um padrão para um conjunto de funcionalidades de uma grade disponíveis na forma de serviços de grade. Estes serviços de grade são disponibilizados através de serviços web estendidos através de outros padrões como o Web Services Resource Framework (WSRF) [10] ou o antigo Open Grid Services Infrastructure (OGSI) [21] para adicionar funcionalidades como o gerenciamento de estado e de transações aos serviços web. Estes padrões poderiam ser aplicados à interface de serviços web do Alchemi com certa facilidade através do Web Services Enhancements (WSE) [13], um complemento da própria Microsoft para o arcabouço .NET que fornece aderência à uma série de especificações de serviços web.

A adoção de padrões como estes faria não só com que grades Alchemi pudessem receber cargas de trabalho de grades de outras plataformas como também enviar cargas de trabalho para estas grades.

Existe uma preocupação sempre presente [36] quando se trabalha com grades computacionais no que diz respeito à segurança. O modelo ou os requisitos de segurança de uma grade computacional podem algumas vezes impor limitações à sua utilização dependendo do tipo de recurso utilizado na grade ou da sensibilidade das informações da aplicação executada. Apesar de o Alchemi, assim como o Globus e o Condor, possuir um modelo de segurança baseado em papéis e para muitas ações na grade seu protocolo de comunicação exigir que credenciais sejam fornecidas, seu modelo de segurança não é tão robusto se comparado a outros arcabouços. Falta ao Alchemi, por exemplo, um mecanismo de encriptação na camada de transporte, como o protocolo TLS [14], fazendo com que toda comunicação entre os nós da grade fique vulnerável a ataques de interceptação, modificação ou fabricação de seus pacotes. Uma outra característica ausente no Alchemi é um mecanismo mais robusto de autenticação, como o Kerberos [37], utilizado em outros arcabouços como o Condor e o Legion, e suporte a credenciais e certificados X.509 [5], suportados principalmente pela especificação GSI [27], que teve sua origem no serviço de segurança do Globus e até hoje é sua base. Ainda no que tange a segurança , seria desejável também um mecanismo de *sandbox* mais robusto. Hoje, cada tarefa é executada em um contexto específico e com fronteiras bem definidas nos nós executores, porém, as

tarefas têm permissões irrestritas neste contexto, podendo ter acesso a quaisquer recursos do nó executor.
A falta de um modelo de segurança mais robusto pode limitar o uso do Alchemi a aplicações e redes que trafegam informações que não exigem um nível alto de segurança ou estão em um ambiente confiável.

```
class GeradorDePrimos
{
    public static GApplication App = new GApplication();

    [STAThread]
    static void Main(string[] args)
    {
        // cria threads de grade para verificar se algum número
        // grande gerado aleatoriamente é primo.
        Random rnd = new Random();
        for (int i=0; i<1000; i++)
        {
            App.Threads.Add(new VerificadorDePrimos(rnd.Next(1000)));
        }

        // especifica as propriedades de conexão
        App.Connection = new GConnection("localhost", 9000, "user", "user");

        // dependências da thread de grade
        App.Manifest.Add(new ModuleDependency(typeof(VerificadorDePrimos).Module));

        // registro no evento ThreadFinish
        App.ThreadFinish += new GThreadFinish(ManipuladorDeFimDeThread);

        // inicio a aplicação
        App.Start();
    }

    private static void ManipuladorDeFimDeThread(GThread thread)
    {
        // coerção de uma GThread para um VerificadorDePrimos
        VerificadorDePrimos verificador = (VerificadorDePrimos) thread;

        // verifica se dado candidato é primo ou não
        bool primo = false;
        if (verificador.Fatores == 2) primo = true;

        // mostra os resultados
        Console.WriteLine("{0} é primo? {1} ({2} fatores)",
                          verificador.Candidato, primo, verificador.Fatores);
    }
}
```

Figura 2.7: Um exemplo de uma aplicação de grade na API do Alchemi. Adaptado de exemplos do SDK do Alchemi.

```
<task>
  <manifest>
    <embedded_file name="Reverse.exe" location="Reverse.exe" />
  </manifest>
  <job id="0">
    <input>
      <embedded_file name="entrada1.txt" location="entrada1.txt" />
    </input>
    <work run_command="Reverse.exe entrada1.txt > resultado1.txt" />
    <output>
      <embedded_file name="resultado1.txt"/>
    </output>
  </job>
  <job id="1">
    <input>
      <embedded_file name="entrada2.txt" location="entrada2.txt" />
    </input>
    <work run_command="Reverse.exe entrada2.txt > resultado2.txt" />
    <output>
      <embedded_file name="resultado2.txt"/>
    </output>
  </job>
</task>
```

Figura 2.8: Um exemplo de um arquivo XML contendo a especificação de uma tarefa no Alchemi. Adaptado de exemplos do SDK do Alchemi.

CAPÍTULO 3

Bases Tecnológicas

Neste capítulo são descritas as principais tecnologias envolvidas na construção do Middle-R e que viabilizaram a realização dos testes preliminares.

3.1 O ambiente R

Atualmente o mais popular projeto de software aberto para análise e computação estatística é o ambiente e linguagem R [48]. O projeto foi originalmente desenvolvido por Robert Gentleman e Ross Ihaka do Departamento de Estatística da Universidade de Auckland e mais tarde foi disponibilizado gratuitamente sob a licença GPL da GNU. Seu código fonte e seus binários estão disponíveis gratuitamente no site do projeto: `www.r-project.org` para diversas plataformas (Linux, Unix, Windows e MacOS). Como ambiente, o R oferece ao usuário métodos para manuseio e armazenamento de dados, operadores para operações com matrizes, ferramentas para análise de dados e geração de gráficos, funções de análises estatísticas avançadas tais como modelagem linear e não linear, testes estatísticos, análise de séries temporais, classificação de dados, etc. Enquanto linguagem, o R é completo e robusto, oferecendo recursos como manuseio de entrada/saída, desvios condicionais, laços de repetição e suporte para criação de funções definidas pelo usuário.

Uma outra característica importante do ambiente R, é o seu suporte para extensibilidade, através da possibilidade de integração de pacotes de funcionalidades desenvolvidos pelos usuários. Enquanto redigimos esta qualificação, existem quase mil pacotes do R disponíveis, que podem estender bastante as funcionalidades do ambiente, adicionando, por exemplo, suporte à conexão com bancos de dados, comunicação através de rede, análise de sinais e de imagens.

O ambiente R também é muito popular entre bioestatísticos porque suporta análise de dados biológicos. Na realidade, muitas técnicas estatísticas e computacionais utilizadas em bioinformática estão presentes no R através de pacotes desenvolvidos pela comunidade, como o projeto independente chamado Bioconductor [32], fazendo com que o R seja bem popular entre estes pesquisadores.

Existem vários pacotes para paralelizar processos no R [51, 55, 39]. Todos estes pacotes foram implementados e estão disponíveis para Linux mas podem ser adaptados para Windows com algumas limitações. Entretanto não existem pacotes com suporte para computação em grade com Windows.

3.1.1 RCOM

A tecnologia Component Object Model (COM) da Microsoft, permite comunicação entre componentes de software de diferentes fabricantes. Um pacote chave utilizado em nossa solução é o pacote RCOM [9], desenvolvido por Erich Neuwirth e Thomas Baier como um pacote para instalações do R baseadas em Windows. Através do RCOM nós podemos expor as características do R através de interfaces COM para outras aplicações Windows escritas em qualquer linguagem de programação compatível com o modelo COM.

Quando o RCOM é instalado em uma máquina, é criado um servidor COM que habilita clientes COM a se conectarem e criarem objetos representando uma instância do R. Através desta instância, pode-se transferir dados (com a devida conversão de dados) entre o R e uma aplicação cliente, usar o motor de computação do R, acessar os controles graficos e facilidades de entrada/saída.

3.2 *Component Object Model*

O modelo COM (*Component Object Model*, ou Modelo de Objeto Componente) foi desenvolvido pela Microsoft em 1993 com o objetivo de fornecer um padrão para a interconexão entre componentes , que fosse independente de linguagem de programação e de ambiente de execução[1]. De forma similar com a que acontece com outras arquiteturas semelhantes como CORBA e *Java Beans*, componentes criados a partir da arquitetura COM, através de uma linguagem de programação compatível, podem comunicar-se com outros componentes que residem fora do contexto de sua aplicação. Este contexto pode ser desde o processo de outra aplicação que esteja sendo executada no mesmo computador, ou uma aplicação sendo executada em computadores diferentes, que podem ou não

[1]Tanto o ambiente de execução quanto a linguagem de programação devem ser compatíveis com o modelo COM.

estar na mesma rede. No modelo COM, dá-se o nome de servidor ao componente que é acessado externamente, e de forma análoga chamamos de cliente o código que acessa um componente COM. O componente, também pode ser chamado de objeto componente ou de objeto COM.

De acordo com padrão COM, para que as implementações de um método possam ser acessadas externamente por um cliente, uma tabela virtual de funções (chamada de `vtable`) é criada em memória, contendo ponteiros para cada um dos métodos que se deseja expor. Um componente é então criado, contendo um ponteiro para a tabela virtual de funções, e é publicado em um servidor. A partir daí, um código cliente pode obter uma referência para o componente servidor e assim ganhar acesso ao ponteiro para a tabela virtual que contém o método que se deseja acessar. Este acesso pode ser feito a partir de qualquer linguagem que possa evocar funções através de ponteiros e suporte o padrão COM.

3.2.1 Interfaces COM

Os componentes COM contêm classes que expõem grupos de métodos, chamadas de interfaces, através das quais os clientes podem interagir com os objetos. A interface é uma coleção dos métodos e dados de um componente, que estão disponíveis para acesso externo e é construída com base na sua `vtable`. O nível de intermediação imposto pelas interfaces fornece ao padrão COM o encapsulamento da implementação dos objetos, uma vez que toda chamada a método ou acesso a dados é feita através de ponteiros para interfaces e nunca diretamente a implementação dos métodos.

As interfaces implementadas são imutáveis, ou seja, uma vez definidas não podem ser alteradas. Pode-se adicionar novas funcionalidades a um componentes através da adição de novas interfaces a uma classe, porém o componente ainda será compatível com interfaces antigas. Contudo pode-se mudar a implementação de uma interface existente sem afetar os clientes que já utilizam a classe. A funcionalidade não é perdida quando componentes COM são atualizados, funcionalidades são melhoradas ou adicionadas. Apesar das interfaces não possuírem número de versão, cada componente COM é identificado através de um identificador de classe (CLSID) e cada interface é identificada através de um identificador de interface (IID) próprio. Tanto os CLSIDs, quanto os IIDs, assim como os demais identificadores do modelo COM são identificadores globalmente únicos[2] (Globally Unique Identifier, ou GUID[3]).

[2]Por ser um inteiro de 128 bits utilizado em todo o contexto de execução, os identificadores são considerados pseudo-únicos, já que existe a pequena probabilidade de um identificador ser gerado para mais do que um componente.

[3]O GUID é a implementação da Microsoft para o padrão UUID (Universally Unique Identifier) pro-

O suporte à herança existe no modelo COM, porém, somente as interfaces podem ser herdadas, não sua implementação. Cada interface COM precisa herdar da interface `IUnknown`, que possui apenas os métodos `QueryInterface(...)`, `AddRef(...)` e `Release(...)`. O método `QueryInterface(...)` é utilizado para obter a referência para uma determinada interface, cujo IID é passado como parâmetro, retornando o ponteiro para a mesma. Os métodos `AddRef(...)` e `Release(...)` servem para controlar o ciclo de vida do componente. No modelo COM, o próprio componente controla seu próprio ciclo de vida, para que o modelo possa prover controle de ciclo de vida em ambientes distribuídos. Sempre que um cliente obtiver um ponteiro para uma interface, deve evocar o método `AddRef(...)` informando a interface da qual obteve o ponteiro, fazendo com que o contador de referências deste componente seja incrementado. De forma análoga, quando um cliente liberar uma interface, deve antes evocar o método `Release(...)` para que o contador da interface informada seja decrementado.

As interfaces são o único modo a partir do qual um componente pode mandar uma mensagem para um método de outro componente. Um ponteiro para um componente, nada mais é do que um ponteiro para uma `vtable`, que por sua vez representa uma das interfaces que este componente implementa e contém um conjunto de ponteiros para seus métodos. Comumente um componente implementa mais do que uma interface.

Pode-se construir componentes COM utilizando qualquer linguagem de programação que suporte a criação de interfaces COM. Entre as linguagens suportadas estão C, C++, Small Talk e Ada.

3.2.2 DCOM

A tecnologia DCOM (*Distributed Component Object Model*, ou Modelo de Objeto Componente Distribuído) é uma extensão do COM que adiciona ao modelo a capacidade de interconexão entre componentes através de computadores diferentes que podem ou não estar na mesma rede ou em redes adjacentes.

Ao modelo COM, a tecnologia DCOM substitui a chamada local inter-processos por uma chamada remota através de um protocolo de rede.Esta chamada remota acontece de forma transparente tanto para o cliente quando para o servidor, e segue o padrão RPC proposto no DCE [50]. A comunicação entre o cliente e servidor é feita através de qualquer protocolo de rede suportado pelo sistema operacional em que o modelo DCOM for utilizado (ex. TCP/IP, UDP, IPX/SPX , NetBIOS, etc., no caso da plataforma Windows 98 ou posterior).

Para ativar um objeto, o cliente solicita a criação de uma instância de uma determi-

posto pela Open Software Foundation como parte do DCE (Distributed Computing Environment).

nada classe utilizando funções da biblioteca COM. Ao verificar que a classe solicitada é uma classe remota, a biblioteca COM redireciona a chamada ao *Service Control Manager* (SCM), componente DCOM responsável pelo mapeamento dos endereços de servidores e de seus objetos. O SCM solicita então a criação do objeto no respectivo servidor, fazendo com que o servidor crie o objeto se necessário e crie um objeto *stub*, responsável por serializar a referência da interface desta instância específica de volta para o cliente, criar e registrar no servidor um objeto *stub* para serializar e de-serializar as mensagens enviadas e recebidas pela interface solicitada. Ao receber a referência serializada a biblioteca DCOM do lado do cliente cria um objeto *proxy* para aquela instância específica. O objeto *proxy* então de-serializa a interface e cria um outro objeto *proxy* que será responsável por serializar e de-serializar as mensagens enviadas e recebidas por este cliente. A criação dos *proxies* e *stubs* é feita de forma transparente pela biblioteca DCOM e tanto o cliente, quanto o servidor se comunicam apenas com estes objetos locais, através de chamadas COM locais.

Por ser uma extensão do modelo COM, o DCOM se utiliza do mesmo mecanismo para a definição de interfaces se diferenciando apenas nos métodos de criação dos *proxies* e *stubs*.

O modelo DCOM oferece também um mecanismo de controle distribuído do cliclo de vida dos objetos, que, assim como no COM funciona com base em contadores de referências. A contagem de referência distribuída também funciona de forma transparente tanto para o cliente quanto para o servidor, uma vez que os *proxies* possuem métodos para incrementar e decrementar o contador, com chamadas aos métodos `AddRef(...)` e `Release(...)` da interface `IUnknown` que são redirecionadas para o SCM. Para que chamadas remotas freqüentes aos métodos responsáveis pelo controle de referências não impactem no desempenho da aplicação, cada *proxy* é responsável por redirecionar as chamadas a estes métodos para o exportador de objetos do DCOM, responsável por controlar localmente o ciclo de vida dos objetos, através dos métodos *RemAddRef(...)* e *RemRelease(...)* de sua interface *IRemUnknown*. O exportador de objetos então modifica o seu contador de referências local e chama os métodos remotos respectivos de acordo com a política adotada. Na política padrão, somente quando o contador local de referências chega a zero, o método *Release(...)* do objeto remoto é evocado diminuindo assim a quantidade de chamadas remotas.

Para lidar com os problemas de disponibilidade dos objetos remotos, tanto devido a falhas de acesso quanto devido ao ciclo de vida normal dos objetos no servidor, o modelo DCOM utiliza um mecanismo de *heartbeat* baseado em *pings* que são enviados aos servidores a cada dois minutos. Caso um servidor não responda três *pings* consecutivos para um determinado objeto, todas as referências para este objeto são marcadas como expiradas e o objeto fica disponível para ser removido pelo coletor de lixo. Cabe ao

exportador de objetos avaliar quais objetos existem em quais servidores e gerar pacotes únicos de *ping* para cada servidor, contendo as verificações para cada objeto daquele servidor, evitando que se gerem chamadas desnecessárias para cada objeto.

De forma semelhante funciona o sistema de balanceamento de carga de cada chamada ao servidor, que é feito de forma dinâmica e automática de acordo com as características de cada requisição de cada cliente e das políticas de balanceamento de carga adotadas por cada aplicação.

3.3 O Arcabouço .Net

O Arcabouço .NET [47, 2] é a tecnologia da Microsoft para desenvolvimento de software, que faz parte do conjunto de tecnologias .NET. O Arcabouço .NET tem como componentes principais um modelo de programação, um conjunto de linguagens de programação, um conjunto de bibliotecas e um ambiente de compilação, interpretação e execução de código.

Seu modelo de programação, chamado de modelo de programação gerenciado, gira em torno de um componente chamado de *Common Language Runtime* (CLR) [44], responsável por executar e gerenciar em tempo de execução códigos escritos através de uma linguagem .NET. No modelo gerenciado, o código fonte escrito em qualquer linguagem aderente ao .NET é compilado, gerando um mesmo código intermediário escrito em uma linguagem chamada de Linguagem Intermediária Comum (*Common Intermediate Language*, ou CIL), chamado de código gerenciado. Este código é então compilado novamente pelo CLR em tempo de execução, através da técnica de compilação *Just In Time* (JIT), gerando o código de máquina nativo do sistema operacional alvo, cuja execução também é gerenciada pelo CLR.

Por gerenciamento entende-se o controle em tempo de execução de aspectos como: uso de memória, uso de *threads*, manipulação de falhas e exceções, verificação de uso tipos de dados e conversões, segurança, comunicação entre componentes locais ou remotos, etc.

De acordo com o modelo, o código executado pelo CLR é chamado de código gerenciado, enquanto o código executado diretamente pelo sistema operacional é chamado de código não gerenciado. Ainda de acordo com o modelo, aplicações escritas em código gerenciado e não gerenciado são chamadas de aplicações gerenciadas e não gerenciadas respectivamente.

O conjunto de linguagens aderentes ao Arcabouço .NET é extenso [3], talvez devido ao fato de que as especificações para a criação destas linguagens são abertas e estão disponíveis para desenvolvedores. Para que uma linguagem possa ser aderente ao Ar-

cabouço .NET, é necessário que se siga a Especificação Comum de Tipos (*Common Type Specification*, ou CTS) e a Especificação Comum de Linguagem (*Common Language Specification*, ou CLS), que garantem que o compilador desta linguagem seja capaz de gerar o mesmo código intermediário na linguagem CIL que os compiladores das outras linguagens compatíveis.

Os compiladores de todas as linguagens aderentes ao Arcabouço .NET geram somente código gerenciado, com exceção do Visual C++, capaz de gerar também código nativo do sistema operacional que, assim como componentes legados escritos em tecnologias anteriores ao arcabouço, são considerados códigos não gerenciados.

O Arcabouço .NET possui ainda um conjunto de bibliotecas orientadas à objeto que fornece tipos para a realização de muitas tarefas básicas como acesso a dados, segurança, manipulação de entrada e saída, depuração de código, serviços de thread, interfaces com o usuário, comunicação remota, entre outras. A biblioteca de classes é única e está disponível para todas as linguagens .NET. Pelo fato do código gerenciado ser o mesmo, não importando a linguagem fonte, tipos escritos em qualquer linguagem .NET podem inter-operar sem necessidade de outro componente intermediário. A Figura 3.1 ilustra a relação entre o CLR, a biblioteca de classes e a relação entre aplicações gerenciadas e não gerenciadas.

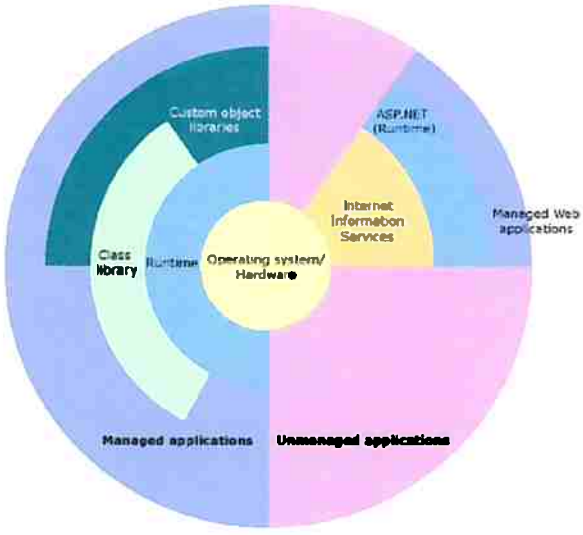

Figura 3.1: Relação entre o CLR, a biblioteca de classes e a relação entre aplicações gerenciadas e não gerenciadas. Fonte: .NET Framework Conceptual Overview (http://msdn.microsoft.com/en-us/library/zw4w595w.aspx).

3.3.1 *.Net Remoting*

O *.NET Remoting* [49] é uma API integrante da biblioteca de classes do Arcabouço .NET e assim como sua tecnologia predecessora, o DCOM, oferece serviços de interconexão entre componentes remotos para a construção de aplicações distribuídas. O princípio básico de funcionamento da arquitetura *.NET Remoting* é similar ao modelo DCOM. O cliente se comunica com o objeto remoto através de um objeto proxy local que emula o objeto remoto e que, ao receber chamadas locais a seus métodos as passa a um formatador que as serializa e as transforma em mensagens que são passadas para uma camada de transporte que as envia ao servidor. No servidor, a camada de transporte recebe as mensagens enviadas pelo cliente e as envia ao formatador para que sejam de-serializadas e repassadas a um objeto *dispatcher* que realizará chamadas locais ao objeto alvo.

A arquitetura é composta por cinco componentes principais: *proxies*, mensagens, sorvedouros de mensagem (*message sinks*), formatadores (*Formatters*) e canais de transporte (*Transport channels*). Assim como no modelo DCOM, os *proxies* são responsáveis por emular o objeto remoto fazendo com que o cliente tenha a percepção de que está lidando com um objeto local. No caso do modelo *.NET Remoting*, dois *proxies* são criados quando se obtêm uma referência para um objeto remoto ou quando se instancia um novo objeto remoto. O `TransparentProxy` é o *proxy* responsável por receber as requisições do cliente e transformá-las em mensagens representadas por objetos `Message`, que então são repassados ao `RemoteProxy`. Cabe a este último, a adição de parâmetros aos objetos `Message` de acordo com o destino de cada mensagem, para então repassá-las a uma seqüência de sorvedouros de mensagens. Os sorvedouros de mensagens acrescentam certo grau de flexibilidade ao modelo *.NET Remoting*, pois pode-se escrever sorvedouros personalizados para realizar operações nas mensagens e conectá-los à seqüência existente. Um sorvedouro padrão fornecido pelo modelo é o formatador de mensagens, responsável por serializar as mensagens de acordo com o padrão escolhido. Pode-se serializar a mensagem no padrão binário, em SOAP [46] ou pode-se escrever um formatador personalizado para outros formatos. Os canais de transporte, responsáveis por encaminhar as mensagens serializadas ao servidor, são outros exemplos de sorvedouros padrão da arquitetura *.NET Remoting*. Como canais de transporte, pode-se escolher por padrão um canal HTTP para transportar mensagens serializadas em SOAP, um canal TCP para o transporte de mensagens serializadas no formato binário, ou então estender qualquer um destes canais acrescentando outras funcionalidades.

O *.NET Remoting* especifica dois tipos de objetos remotos que se diferem na maneira como são acessados através de contextos diferentes, os objetos que são passados por valor e os que são passados por referência. Os objetos passados por referência, são objetos de classes descendentes da classe `MarshalByRefObject` se encontram somente no contexto do servidor. Quando passados como argumentos, estes objetos não ultrapassam o contexto

em que foram criados. Somente sua referência é passada através de um método similar ao do COM, onde a referência para o objeto é obtida em termos de uma interface ou classe e o objeto é representado por um *proxy* no lado cliente. Sendo assim, os métodos e propriedades do objeto são acessados externamente pelo cliente e são executados sempre no contexto do servidor.

Analogamente ao que acontece com muitas linguagens de programação orientadas a objetos, objetos que são passados por valor são copiados quando passados como argumento ou como retorno de métodos. Para ultrapassar o contexto da aplicação, estes objetos são serializados e transmitidos para o cliente, onde são de-serializados e restaurados como uma cópia do objeto original do servidor. A partir daí são acessados e executados localmente pelo cliente, de forma independente do servidor.

A serialização de objetos é fornecida nativamente pelo mecanismo de formatação do Arcabouço .NET . Cabe a este mecanismo a tarefa de converter os tipos básicos de dados bem como interfaces, objetos e todas as referências que estiverem contidas neste objeto para o formato binário ou SOAP para que possam ser transmitidos através de um canal de comunicação. Para isto, basta que as classes envolvidas na serialização sejam marcadas com o atributo de classe `[Serializable]` ou que implementem a interface `ISerializable`. A serialização então passa a valer para todos os membros desta classe.

A arquitetura .NET Remoting oferece um mecanismo de controle do ciclo de vida dos objetos com base em licenças (*leases*) e patrocinadores (*sponsors*). O controle do ciclo de vida dos objetos remotos é feito por cada aplicação, que possui um gerente de licenças. O gerente de licenças é o componente que, além de controlar o tempo de expiração dos objetos, os marca para serem coletados pelo coletor de lixo. Quando um novo objeto remoto de uma aplicação é criado, este é registrado no gerente de licenças quando são informados o tempo de licença e seus objetos patrocinadores. Quando a licença de um objeto expira, seus objetos patrocinadores são contatados para que possam renovar o tempo de licença. Dependendo da política adotada, o tempo de licença volta ao valor iniciar cada vez que o objeto for acessado por um cliente. O tempo de licença também depende da política adotada para o controle do ciclo de vida na aplicação. Caso um objeto patrocinador não esteja disponível no momento em que a licença de um objeto expirar, o gerente de licenças ainda espera a resposta por um período de tempo configurável que, depois de ultrapassado, faz com que o objeto seja marcado para remoção pelo coletor de lixo.

3.4 Interoperabilidade com códigos legados

Pelo fato do modelo de desenvolvimento e execução do Arcabouço .NET ser baseado no CLR, que interpreta, recompila e gerencia a execução do código .NET, o código gerenciado não consegue ter acesso à objetos e componentes não gerenciados. O código gerenciado e o código não gerenciado não compartilham dos mesmos tipos e nem da mesma região de alocação de memória, entre outras diferenças. Entretanto, manter a interoperabilidade entre os dois modelos é importante para aumentar as chances de aceitação do Arcabouço .NET, principalmente por empresas e desenvolvedores que investiram recursos no desenvolvimento de componentes não gerenciados.

Para prover serviços de interoperabilidade com código legado, três mecanismos [45] fazem parte do Arcabouço .NET: o *COM Interoperability* (chamado de *COM Interop*) para promover o acesso a componentes COM a partir de código gerenciado; o *.NET Interoperability* (chamado de *.NET Interop*) para promover acesso a componentes gerenciados à partir de código COM e os *Platform Invokation Services* (chamados de *PInvoke*) que provêm serviços para o acesso de bibliotecas legadas a partir de código .NET.

Durante o desenvolvimento deste trabalho, foram estudados os mecanismos *COM Interop* e *PInvoke*. Os dois modelos são compostos por serviços presentes no CLR, APIs e ferramentas disponíveis no SDK do Arcabouço .NET.

3.4.1 *COM Interoperability*

O modelo COM e o Arcabouço .NET possuem grandes diferenças em termos de modelo de programação, tipos de dados, ciclo de vida de objetos, etc. No Arcabouço .NET, a forma prevista para a interação entre estes dois modelos é a utilização de uma camada de interoperabilidade criada através das bibliotecas e ferramentas presentes no SDK do arcabouço. A camada de interoperabilidade é desempenhada por um *wrapper* chamado de *Runtime Callable Wrapper* (RCW). Um RCW é um *proxy* que traduz chamadas do código gerenciado .NET para o código não gerenciado de um componente COM. Essa intermediação é transparente para o código .NET, que acessa o componente COM da mesma maneira com que acessa um objeto gerenciado .NET.

O gerenciamento de memória no lado não gerenciado é feito pelo próprio RCW, que gerencia o acesso dos códigos gerenciados aos ponteiros das interfaces do objeto COM, controlando suas referências e seu ciclo de vida. Por sua vez, como o RCW é um componente gerenciado, seu ciclo de vida é controlado pelo CLR. Cada vez que uma classe RCW criada para um determinado componente COM é instanciada, uma nova instancia deste componente é criada. Deste modo, para cada instância de uma classe COM é atribuído um único RCW.

Além do controle do ciclo de vida do objeto COM, uma das tarefas essenciais do RCW é converter os tipos de dados entre os dois modelos. A conversão -chamada de *parameter marshaling*- ocorre tanto nos argumentos passados nas mensagens para os objetos COM, quanto nas mensagens de retornos destes objetos. O modelo fornece serviços de conversão de dados através de classes e métodos presentes no espaço de nomes `System.RunTime.InteropServices`.

A ferramenta `TLBIMP.EXE`, que integra o SDK do Arcabouço .NET, é a responsável por criar uma classe RCW a partir de um componente COM (geralmente representado na forma de uma biblioteca DLL). Esta ferramenta utiliza os tipos de dados descritos na biblioteca de tipos de um objeto COM (a forma com que o modelo COM descreve os tipos utilizados em suas operações) criando os métodos necessários para sua conversão em tipos do Arcabouço .NET, descritos como metadados (forma com que o modelo .NET descreve os tipos de dados suportados por suas classes). Estes métodos são adicionados aos métodos que manipularão as interfaces do objeto COM e seu ciclo de vida e integrados a uma classe .NET. Esta classe é então compilada, gerando uma DLL representando o RCW daquele componente COM.

3.4.2 *Platform Invocation Services (PInvoke)*

O mecanismo *PInvoke* foi desenvolvido pela Microsoft para dotar a plataforma .NET da capacidade de acessar, a partir de código gerenciado, códigos nativos (código não gerenciado) que não possuam interfaces públicas publicadas através de um objeto COM. O mecanismo *PInvoke* é parte integrante da especificação *Common Language Infrastructure* (CLI) também desenvolvida pela Microsoft e sob a qual a plataforma .NET foi baseada.

O acesso se dá através da lista das funções exportadas por cada biblioteca. Estas listas são criadas pelo *linker* no momento da compilação e contêm os pontos de entrada estáticos através dos quais funções da bilioteca podem ser chamadas a partir de aplicações externas. Apenas as funções marcadas para exportação são expostas e farão parte desta lista. As funções são exportadas utilizando a convenção de chamada `CDecl` [19].

Para acessar as funções das bibliotecas não gerenciadas utilizando o *PInvoke* é necessário declará-las no código não gerenciado através de sua assinatura e para cada uma utilizar o atributo `DLLImports` para informar o compilador em qual biblioteca a função se encontra e para qual função está mapeada. O nome da função mapeada precisa coincidir com o nome exportado pela biblioteca e sua declaração precisa seguir a mesma convenção de chamada.

Utilizando os serviços do PInvoke, é possível criar um *proxy* .NET para receber uma requisição para uma determinada função presente em uma DLL e repassá-la para o en-

dereço de memória onde o ponto de entrada desta função reside, fazendo as conversões necessárias de argumentos e dados e dados de retorno. Assim como acontece no *COM Interoperability*, a conversão de tipos (*parameter marshaling*) é feita através das classes e métodos presentes no espaço de nomes `System.RunTime.InteropServices`.

Quando é feita uma chamada pela primeira vez a uma função atribuída a uma biblioteca não gerenciada, o CLR localiza esta biblioteca e a carrega em memória. A localização do endereço de memória do ponto de entrada da função é feita também apenas na primeira chamada, através da busca na lista de pontos de entrada estáticos da biblioteca. A partir daí, sempre que uma chamada for feita a esta função, o CLR converterá seus argumentos e os empilhará na pilha de execução (já no espaço de memória não gerenciado) e transferirá o controle para a função chamada. Após a execução, os valores devolvidos são reconvertidos em metadados e repassados para o responsável pela chamada. Caso haja uma exceção, esta também será repassada para o responsável pela chamada.

Apesar dos modelos *COM Interoperability* e *PInvoke* compartilharem as mesmas classes de conversão de tipos, existem diferenças importantes entre os modelos. No modelo *PInvoke*, a fronteira entre os espaços de memória gerenciados os não gerenciados é atravessada diversas vezes para escrita e leitura de valores, longe do alcance do gerenciamento de tipos do CLR, estando assim mais suscetível a problemas de compatibilidade de dados e violação de memória. Outra diferença importante é que por não possuírem nenhum método de controle do ciclo de vida (como aqueles utilizados no modelo COM) ou de coleta de lixo (como aquele fornecido pelo CLR), todo o gerenciamento de memória e controle transacional deve ser tratado pelo usuário.

CAPÍTULO 4

Middle-R

Este capítulo se inicia com a discussão sobre a aplicação dos mecanismos de interoperabilidade entre o Arcabouço .NET e o modelo COM aplicados na interligação entre o uma aplicação .NET Alchemi e o motor de computação do R. Em seguida é feita uma discussão sobre os modelos de programação do Alchemi e as diferentes abordagens para aproveitá-los, seguida do detalhamento das soluções propostas.

4.1 Interoperabilidade entre .NET e COM

O método de acesso ao motor computacional do R a partir de código externo na plataforma Windows, previsto pelos desenvolvedores do R é a utilização do pacote RCOM. A utilização do pacote RCOM como forma de conectar o motor de computação do R com bibliotecas e códigos externos surgiu naturalmente devido ao fato do pacote estar disponível aos usuários do R desde suas primeiras versões para Windows e de sua biblioteca fazer parte da distribuição principal do *software*. Outro fator decisivo na escolha do pacote RCOM como método de acesso ao R foi o fato de que através do pacote RCOM, os objetos e métodos do R podem ser acessados sem que haja a necessidade de conhecimento prévio de seus tipos e de conversões específicas, podendo obtê-los dinamicamente através da interface publicada no servidor COM para uma instância de sua classe *proxy*. Com a utilização de um método padrão de acesso, houve economia de tempo para o desenvolvimento do protótipo e prova de conceito.

Conforme exposto na Seção 3.2 , o modelo COM foi desenvolvido alguns anos antes do Arcabouço .NET e apesar da Microsoft as considerar tecnologias complementares, na verdade são incompatíveis uma vez que seus modelos de programação são significantemente diferentes. Um dos desafios a ser ultrapassado para que o R pudesse ser utilizado por uma aplicação de grade Alchemi era conseguir utilizar o motor de computação do R a partir

de uma aplicação .NET construída utilizando as bibliotecas da API do Alchemi, estando assim pronta a utilizar as facilidades de uma grade Alchemi. O problema principal é que código COM não tem sua execução gerenciada pelo CLR, sendo considerado código não-gerenciado. Sendo assim, uma aplicação que utilize as bibliotecas .NET do Alchemi não tem acesso ao código COM, uma vez que código gerenciado só tem acesso aos códigos cuja execução é gerenciada pelo CLR, conforme descrito na Seção 3.3.

4.1.1 Acessando o R através do servidor RCOM

O pacote RCOM fornece um servidor COM que expõe as funcionalidades do motor de computação do R para que possam ser acessadas através de um cliente COM. Estas funcionalidades são acessadas através da interface COM `IStatConnector` .Através desta interface, um cliente COM é capaz de iniciar e encerrar uma sessão R, ler e escrever dados nesta sessão, avaliar expressões R, entre outras funcionalidades. O servidor COM propriamente dito é implementado através da classe COM `StatConnector`, que implementa a interface `IStatConnector`. Para ter acesso ao motor de computação do R, o servidor com necessita da classe *proxy* `SC_Proxy_Object` que contém os ponteiros para funções internas da API do R (contidas, em sua maioria, na biblioteca `R.dll` no caso das distribuições R para Windows). A classe proxy também define métodos para conversão de dados entre o formato de dados interno do R e um formato intermediário desenvolvido pelos criadores do pacote para troca de dados entre os contextos do COM e do R, chamado de *Binary Data eXchange format* (BDX). Estes ponteiros e métodos são mapeados pela interface `IStatConnector` do servidor COM e expostos para o clientes COM. Apesar do servidor COM ser fornecido através da instalação do pacote RCOM, a classe *proxy* se encontra na biblioteca `Rproxy.dll` fornecida junto com as distribuições Windows do R. Isto se dá pelo fato da classe *proxy* ser fortemente acoplada à API do R, para evitar que uma nova versão do servidor COM tenha que ser lançada a cada versão do R .

4.1.2 Acessando e utilizando o servidor COM

Para realizar o acesso ao servidor COM, foi criado um componente que faz o papel de um *Runtime Callable Wrapper* (RCW). Conforme exposto na Seção 3.4, um RCW funciona como um *proxy* que fica na fronteira entre os contextos do código gerenciado e do código não gerenciado. O componente desenvolvido realiza as conversões entre as chamadas COM e o código C# do Alchemi e fazendo com que o objeto COM `StatConnector` seja visto como um componente nativo .NET para o código de uma aplicação Alchemi, conforme ilustrado na Figura 4.1.

Esta camada de interoperabilidade foi construída usando a ferramenta `tlbimp.exe`

fornecida pelo próprio arcabouço .NET para conversão de interfaces públicas COM em objetos .NET utilizando como base as informações armazenadas na biblioteca de tipos do componente COM. A saída da ferramenta é a classe `StatConnectorClass`, compilada na biblioteca `StatConnector.Interop.dll` , que pode ser utilizada por qualquer aplicação .NET para realizar chamar ao objeto COM `StatConnector` do RCOM para acessar as funcionalidades do motor de computação do R.

Ao instanciar a classe `StatConnectorClass` no código .NET, o componente COM é localizado pelo construtor da classe *proxy* através do método `QueryInterface(...)`. Neste momento, uma nova instância da interface `IStatConnector` é criada e adicionada ao contador de referências do objeto, através do método `AddRef(...)`. A partir deste momento, o objeto COM está pronto para ser utilizado na aplicação .NET cliente.

Para utilização do ambiente R na aplicação .NET, o seguinte ciclo de chamadas aos métodos da `StatConnecotrClass` é obedecido: O ambiente R é iniciado através do método `Init(...)`; dados são transferidos para o ambiente através do método `SetSymbol(...)`; expressões R são avaliadas através dos métodos `Evaluate(...)` e `EvaluateNoReturn(...)`; dados são lidos através do método `GetSymbol(...)`; o ambiente R é finalizado através do método `Close(...)`. A Figura 4.2 descreve a seqüência de mensagens entre os componentes envolvidos em um fluxo típico de chamadas ao R.

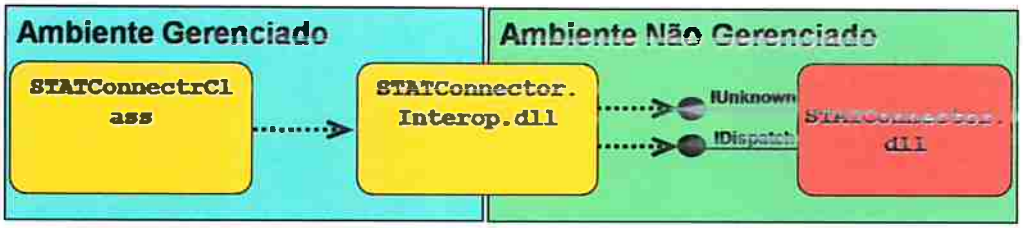

Figura 4.1: Representação do funcionamento do componente `StatConnector.Interop`, funcionando como um *Runtime Callable Wrapper*.

4.2 O MiddleR no modelo de *jobs*

Uma vez que o acesso ao motor de computação do R a partir de código .NET foi resolvido, foi desenvolvido uma primeira versão do MiddleR para funcionar no modelos de *jobs* do Alchemi.

Conforme descrito na Seção 2.3, o modelo de *jobs* do Alchemi suporta aplicações simples do tipo *bag-of-tasks*, e foi desenvolvido com o propósito de dar suporte a aplicações

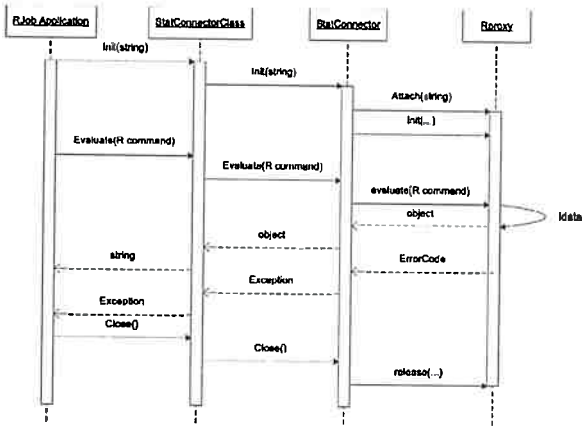

Figura 4.2: Sequência de mensagens entre os componentes encolvidos em uma chamada ao R.

legadas, aplicações que não foram desenvolvidas utilizando as APIs do Alchemi e aplicações submetidas por outros *middlewares* de grade através do gerente multi-plataforma.

No modelo de *jobs*, uma aplicação da grade é representada por uma tarefa que por sua vez é formada por diversos *jobs*. Ao criar uma tarefa, deve-se especificar os arquivos (executáveis, bibliotecas, etc.) dos quais os *jobs* desta tarefa dependem para serem executados nos nós executores. Para cada *job*, deve ser especificada uma coleção de arquivos de entrada e uma de arquivos de saída utilizados para se passar parâmetros e receber resultados deste *job* e o comando que deve ser executado pelo *job* nos nós executores.

Para o MiddleR funcionar no modelo de *jobs*, foi desenvolvido um pequeno programa chamado RJob que tem como papel ler um *script* R, avaliá-lo no motor de computação do R e gerar um arquivo de saída com o resultado desta avaliação. Para utilizar o RJob em uma grade Alchemi, basta colocá-lo como dependência de uma tarefa da grade, definir os *scripts* R que serão utilizados em cada *job* e definir os argumentos de execução deste *script*. No momento da execução, o RJob é copiado do nó gerente para os nós executores, junto com um arquivo de entrada contendo o *script* R que se deseja executar em cada nó da grade.

Na API do Alchemi, uma aplicação é uma instância da classe `GApplication`. O inventário de arquivos dos quais a aplicação depende, é representado por uma coleção de objetos do tipo `FileDependency`, e deve conter todos os arquivos que o RJob necessita para ser executado localmente, inclusive o próprio `Rjob.exe`. O objeto `GApplication` contém uma coleção de *jobs* que serão executados. Cada *job* é um objeto herdado de `Gjob` e tem uma coleção de arquivos de entrada, uma coleção de arquivos de saída e um comando que será executado quando chegar a um nó Executor.

O RJob é uma aplicação Windows escrita em C# que utiliza o componente `StatConnectorClass` para se comunicar com o R através do modelo COM. Por este motivo, a biblioteca `StatConnector.Interop.dll` também deve ser adicionada como dependência da tarefa.

Em cada nó executor, o RJob analisa o respectivo arquivo de entrada avaliando cada linha do *script* e gerando um arquivo de saída com os resultados da execução deste *script*. O resultado será copiado de volta ao nó Gerenciador da grade. A Figura 4.3 mostra um esquema da solução Rjob.

Ao enviar uma tarefa para um gerente, todas as dependências e arquivos de entrada são transmitidos ao gerente e repassados aos executores através de cópia binária sobre TCP/IP e quando a tarefa for terminada os arquivos de saída são devolvidos ao gerente da mesma forma.

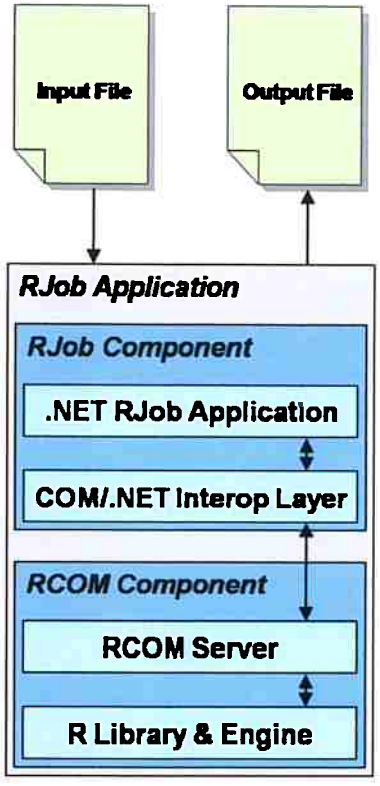

Figura 4.3: Esquema geral de funcionamento do RJob.

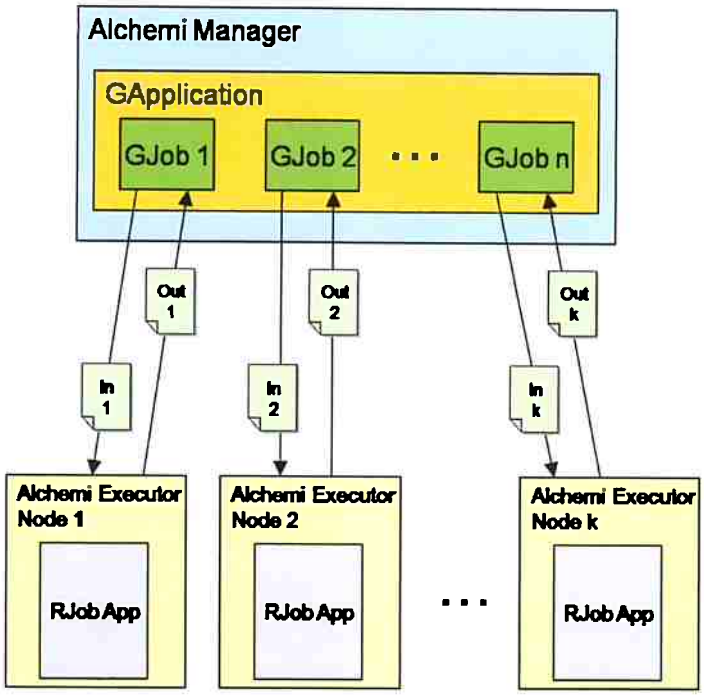

Figura 4.4: Esquema geral de funcionamento do MiddleR no modelo de *jobs* do Alchemi.

4.3 O MiddleR no modelo de *threads*

O modelo de *threads* do Alchemi é o modelo de criação e execução de aplicações de grade de grade orientado a objetos que abstrai detalhes da arquitetura da grade criando um ambiente semelhante à programação *multi-threading*. O modelo de *threads* oferece ao desenvolvedor de uma aplicação de grade acesso à API do Alchemi e suas facilidades como o monitoramento da execução das threads, mecanismos de comunicação *inter-threads*, eventos, etc.

Uma aplicação Alchemi que utilize o modelo de *threads* de grade possui uma *thread* principal que contém o código de coordenação da aplicação e da execução das *threads* que a compõe. Este código é executado localmente no nó gerente, chamado de código local e representado pela classe `GApplication` na API do Alchemi. O código que forma cada *thread* é o código que será executado remotamente pelos nós da grade, sendo referido como código remoto e representado pela classe `GThread`. No modelo de *threads*, a própria aplicação é responsável por serializar e transmitir as *threads* de grade que compõe esta aplicação para um nó gerente para agendamento e execução nos nós executores.

Para utilizar o R no modelo de *threads*, foi desenvolvido uma classe chamada `RThread`, que estende a classe `GThread`. A classe `RThread` possui uma referência para um objeto

`StatConnectorClass`, através do qual faz a conexão com o servidor COM do R. Para abstrair o ciclo de utilização de uma sessão R e manter a existência de apenas uma sessão para cada *thread* executada, ao a iniciação de uma sessão R é feita de forma automática pelo construtor da classe `RThread`. Da mesma forma, a sessão R é finalizada no momento da destruição do objeto. Para manter a funcionalidade da avaliação de scripts R presente no `RJob.exe`, foi criado um método `EvaluateScript(...)` que recebe como argumento o caminho de um *script* R e devolve o resultado textual da avaliação deste *script*. O resto dos métodos são apenas mapeamentos dos métodos presentes na `StatConnectorClass` para avaliação de declarações, leitura e gravação de valores na sessão do R. Foram mapeados os métodos `Evaluate(...)`, `EvaluateNoReturn(...)`, `SetSymbol(...)` e `GetSymbol(...)`.

Como a classe `RThread` funcionará como uma *thread* da grade, será serializada e transmitida para os nós executores. Um cuidado teve de ser tomado uma vez que a classe `StatConnectorClass` não pode ser serializada devido a sua complexidade e a suas dependências de código não-gerenciado. Porém quando uma classe é marcada com o atributo `[Serialzable]` mantêm uma referência a uma outra classe, o CLR tentará serializar esta outra classe também. Caso não seja possível, uma exceção do tipo `SerializationException` será lançada. Para contornar esta condição, a referência para o objeto `StatConnectorClass` teve de ser macarda com o atributo `[NonSerializable]`. Como o componente `StatConnectorClass` não é copiado automaticamente para as estações no momento da serialização, deve-se certificar de que o binário `StatConnector.Interop.dll` se encontra ao alcance do responsável pela execução da *thread* (neste caso, o componente Executor). Como a estrutura de diretórios criada pelo Alchemi é dinâmica e feita com base na identificação de cada aplicação no nó gerente, foi tomada a decisão de projeto de copiar o binário como dependência de cada *thread* quando esta é transmitida aos nós executores.

4.4 Vantagens e desvantagens dos dois modelos

Para a utilização do MiddleR com o modelo de jobs, nenhum conhecimento de programação é necessário, uma vez que pode-se montar uma tarefa utilizando um arquivo XML contendo a descrição de seus *jobs*, dependências e comandos de execução. Este arquivo pode ser submetido utilizando uma ferramenta para submissão de *jobs* fornecida pelo próprio SDK do Alchemi. A utlização do MiddleR no modelo de *threads* requer um certo conhecimento de programação C# e da API do Alchemi por parte do desenvolvedor da aplicação. No modelo de *jobs*, pode-se melhorar o desempenho da aplicação eliminando-se a dependência do `RJob.exe` e do `StatConnector.Interop.dll` e apenas mandando a linha de comando para execução da aplicação e copiando os arquivos necessários em um passo de *staging* anterior a execução da tarefa. No modelo de *threads*, para garantir

que a biblioteca `StatConnector.Interop.dll` esteja ao alcance da execução da thread, deve-se copiá-lo junto com cada thread transmitida. Porém, através do modelo de thread, o desenvolvedor da aplicação pode ter um maior controle da execução das threads, uma vez que tem acesso a API do Alchemi e de seus mecanismos de monitoramento, comunicação, eventos, além de poder contar com as bibliotecas .NET e recursos do C#, podendo combinar a aplicação R com processamento em C#. Ainda, no modelo de jobs apenas respostas textuais são devolvidas como resultado do processamento de um *script* R, enquanto através do modelo de *threads*, podemos ter acesso aos métodos para leitura e gravação de dados na sessão do R.

4.5 Melhorando o desempenho

Na tentativa de melhorar o desempenho da aplicação e diminuir o número de camadas intermediárias nas chamadas à biblioteca do R, foi utilizado um segundo método para acessar a `R.dll` utilizando os serviços do PInvoke. O método de acesso ao motor computacional do R utilizado até então envolve o código .NET de uma *thread* ou *job* do Alchemi, passando pela classe de interoperabilidade `StatConnectorClass`, pelo objeto COM `StatConnector`, pela classe *proxy* `SC_Proxy_Class` até finalmente acessar as funcionalidades do motor computacional do R presentes em sua biblioteca `R.dll`. A idéia era melhorar o desempenho no acesso eliminando as camadas do servidor COM e da classe proxy `SC_Proxy_Class` e acessar diretamente as funções da biblioteca `R.dll`.

4.5.1 Acessando a Rproxy.dll através da PInvoke

Para acessar a `R.dll` através do *PInvoke*, foi desenvolvida uma classe `RWrapper` baseada em uma classe de mesmo nome desenvolvida por Laurent Dupuis e publicada na lista de desenvolvedores do R [?]. Por conveniência, a classe `RWrapper` possui os mesmos métodos de acesso ao R presentes no objeto `StatConnector` sendo os métodos `Evaluate(...)` e `EvaluateNoReturn(...)` para avaliar declarações R e `GetSymbol(...)` e `SetSymbol(...)` para ler e gravar dados no ambiente.

Na classe `RWrapper`, as funções das bibliotecas gerenciadas são declaradas com o atributo `DLLImports`. Os tipos de dados utilizados como argumentos e retornos de função são declarados na forma de *structs* e precisam seguir o mesmo alinhamento dos tipos utilizados no R. Os ponteiros para as funções associadas a interface gráfica do R são declarados como *delegates*, mas em alguns casos, é utilizado um atributo `MarshalAs` para forçar uma conversão de tipo para um tipo não gerenciado.

Como o acesso é feito de forma direta a `R.dll`, toda a troca de dados e chamadas

entre os contextos gerenciado e não gerenciado tem de ser feita manualmente. Para as conversões de dados algumas funções da biblioteca `RProxy.dll` tiveram de ser declaradas, como as funções que fazem conversões entre tipos. Os tipos envolvidos são BDX (estrutura para troca de dados entre contexto interno e externo do R), *strings*, tipos variantes (geralmente representados por uma referência a classe genérica `object`), expressões do R (representado por uma estrutura SExp), etc. Como o gerenciamento de memória fica sob responsabilidade do código cliente, algumas funções de gerenciamento de memória também foram declaradas.

Apesar de sua implementação ser um tanto quanto complicada, o uso da `RWrapper` é bem simples. Basta instanciar um objeto e a partir deles utilizar as funções de avaliação e transferência de dados.

A inicialização da sessão R é feita pelo construtor. No momento da instanciação, o construtor carrega as bibliotecas do R com suas variáveis principais, inicia uma nova sessão e armazena seu ponteiro para as futuras referências e associa seus retornos de chamada.

As conversões de dados relevantes acontecem nas funções de avaliação e de transferência de dados.

As funções `Evaluate(...)` e `EvaluateNoReturn(...)`, assim como suas equivalentes no objeto COM, recebem uma *string* contendo a expressão R a ser avaliada. A partir desta *string* é criada uma versão da expressão R no formato da estrutura `SExp` (utilizando as funções da *RProxy.dll*) que é avaliada no contexto da sessão R. Caso a função avaliação possua retorno, a função de avaliação no contexto da sessão R retorna um ponteiro para uma estrutura `SExp` de retorno. Esta expressão precisa ser convertida para o formato BDX para depois ser convertida para o tipo `object` , este já um tipo gerenciado.

Ao usar ou `GetSymbol(...)` ou `SetSymbol(...)`, as mesmas conversões são utilizadas. No caso do `GetSymbol(...)` o nome do símbolo (estrutura, objeto ou variável) existente na sessão R é passado como uma *string*, que precisa ser convertida para `SExp` para ser procurada na sessão R. O resultado da busca retorna como uma `SExp` que precisa ser convertida para BDX e em seguida para o `object` que é retornado. No caso do `SetSymbol(...)`, além do nome do símbolo, é necessário passar um `object` contendo o valor que se deseja atribuir a ele. Este objeto passa pela conversão inversa do seu argumento.

O acesso direto às bibliotecas do R possui algumas desvantagens:

- Gerenciamento de Memória
  O gerenciamento de memória deve ser feito manualmente, devendo-se preocupar em liberar memória não mais utilizada. O código não gerenciado está fora do alcance do coletor de lixo.

- Leitura e escrita diretamente da memória

Como existe escrita e leitura diretamente da memória não gerenciada a partir do código gerenciado. A tarefa de depuração se torna muito complicada porque pode ser muito difícil obter mensagens de erro. Em caso de acesso a alguma região de memória incorreta, pode levara a aplicação a se comportar de maneira não normal ao invés de obter mensagens de erro. O acesso direto também pode ser potencialmente inseguro.

- Conversões Manuais
  Os dados devem ser convertidos manualmente.

- Ciclo de vida da sessão R
  Quando o acesso é feito através do objeto COM, por conta de seu desenho, somente uma instância de uma sessão R pode existir por objeto. Quando o acesso é feito de forma direta, pode-se iniciar muitas sessões simultâneas podendo levar a enganos e esgotamento de recursos.

Algumas destas desvantagens foram endereçadas no desenho da `RWrapper` pelo próprio autor da classe, porém um comando frequentemente utilizado no R não é suportado. O comando print, que serve para impressão de mensagens no console, causa um erro que leva ao encerramento da aplicação. Este problema não foi resolvido também por este trabalho, persistindo como uma falha a ser corrigida.

4.6 Uma aplicação de exemplo

Em um ambiente como R, uma tarefa importante de se cumprir é ser capaz de gerenciar os pacotes instalados, ou seja, manter o controle da versão dos pacotes, atualizá-los, sincronização de dados e assim por diante. Este tipo de controle é importante porque aplicações científicas precisam de uma garantia de reprodutibilidade [31], e o que normalmente é feito ao realizar uma análise é registrar as versões do ambiente e de todos os pacotes usados na análise. Entretanto, esta tarefa que normalmente consome muito tempo quando na manutenção de uma estação, pode se tornar um problema sério quando evoluímos a escala para centenas de estações.

Existem dois métodos para instalação de pacotes no R. Uma instalção semi-automática onde, através de uma interface gráfica (uma característica única do Windows) ou através da linha de comando do ambiente R, pode-se selecionar um repositório, obter uma lista de todos os pacotes disponíveis e automáticamente baixar do servidor e instalar a versão mais recente do pacote escolhido. E uma instalação manual onde pode-se instalar um pacote previamente transferido manualmente de um dos repositórios (na forma de um arquivo compactado) através de um comando no console. Em um ambiente distribuído,

a primeira opção tem várias desvantagens: a necessidade de parametrizar manualmente as preferências de transferência através de uma interface gráfica ou da linha de comando do ambiente, e o tráfego gerado na rede por todas as estações que estiverem transferindo arquivos de um servidor na internet, por exemplo. A segunda opção é melhor porque não tem as desvantagens da primeira e, mais importante, consegue-se manter controle das versões de pacotes instalados nas estações.

4.6.1 Uma solução Middle-R

A solução faz uso do middle-R para atualizar e instalar os pacotes do R criando um processo que copia todas as dependencias de arquivos e executa todos os comandos necessários no R para a instalação e sincronização dos pacotes.

Esse exemplo foi construído utilizando o modelo de jobs do alchemi através de sua API. Uma vez iniciado o `GApplication`, um executável Rjob é adicionado ao inventário de dependências para que a aplicação, junto com o arquivo `e1071_1.5-1.zip` que contém o pacote e1071 [15]. Os arquivos serão copiados do nó Gerenciador a cada nó Executor no processamento do job, portanto, nós definimos o número de jobs a serem criados e executados. Uma vez que um job é configurado, ele é adicionado à coleção de threads da aplicação, que esta pronta para ser iniciada. Figura 4.6.1 mostra um pedaço do código de uma aplicação que utiliza o modelo de job do Alchemi.

Como esclarecemos, um arquivo de entrada para este processo é um arquivo script que contém comandos do tipo:

```
install.packages("e1071_1.5-1.zip",.libPaths()[1])
```

O comando `install.packages` instala os pacotes passados no primeiro argumento no ambiente R local.

Quando todos os processos tiverem terminado, haverá uma quantidade de arquivos de saída em texto igual ao número de jobs submetidos, com o resultado de cada operação. Nesse caso, esses arquivos podem ser auditados mais tarde para se averiguar o sucesso das operações. Para se atualizar vários pacotes no mesmo processo, basta adicionar todos os binários dos pacotes inventário de dependências da aplicação e incluir o respectivo comando `install.packages` ao arquivo de entrada de cada job.

```
...

gridApp = new GApplication(
            GConnection.FromConsole("localhost",
              "9000", "user", "user"));
...

gridApp.Manifest.Add(
  new EmbeddedFileDependency("RJob.exe",
    @".\RJob\bin\RJob.exe"));

gridApp.Manifest.Add(
  new EmbeddedFileDependency(
    "e1071_1.5-1.zip",
       @".\Packages\e1071\e1071_1.5-1.zip"));

numJobs  = n;

for (int jobNum = 0; jobNum < numJobs; jobNum++) {
    GJob job = new GJob();
    string outputFileName =
      string.Format("output{0}.txt", jobNum);

    job.InputFiles.Add(
      new EmbeddedFileDependency("input.txt",
        @".\Packages\e1071\input.txt"));

    job.OutputFiles.Add(
        new EmbeddedFileDependency(
          outputFileName));
    job.RunCommand = string.Format(
      "RJob input.txt > {0}", outputFileName);

    gridApp.Threads.Add(job);
}

gridApp.Start();

...
```

Figura 4.5: Exemplo de código do modelo de job do Alchemi

CAPÍTULO 5

Resultados Experimentais

Neste capítulo, são descritos os testes realizados com a plataforma proposta, utilizando como aplicação o problema combinatorial de bioinformática de encontrar classificadores moleculares utilizando técnicas de reconhecimento de padrões.

5.1 Descrição do problema

Uma das mais importantes áreas em medicina, e conseqüentemente em bioinformática, é o prognóstico e diagnóstico correto da expressão de certos genes em uma amostra de tecido. A expressão de um gene é a quantidade deste gene que está se expressando em uma célula a um certo momento de seu ciclo. A idéia é encontrar um conjunto de genes que podem ter sua expressão mensurada (de tecidos de biópsias, por exemplo) e analisada para auxiliar o diagnóstico, ou até melhor, o prognóstico de uma certa condição deste tecido.

Existem várias técnicas para quantificar a expressão gênica de células de um dado tecido. Microarrays de cDNA e arrays de oligonucleotídeos de alta densidade são as técnicas de microarray mais populares. O ponto positivo destas técnicas é que cada experimento de microarray pode medir a expressão de milhares de genes ao mesmo tempo.

Por razões práticas, a saída de um experimento de microarray é uma matriz $n \times p$, onde n é o número de genes e p é o número de tecidos sendo avaliados. Cada elemento da matriz $g_{i,j}$ é a empressão no gene i no tecido j. As expressões podem ser absolutas ou relativas a expressão de outro tecido ou conjunto de tecidos. No caso do conjunto de dados utilizados neste trabalho, o experimento é a comparação de amostras de Leucemia Linfoblástica Aguda (ALL) com amostras de Leucemia Mielóide Aguda (AML). A expressão é mensurada usando arrays oligonucleotídeos de alta densidade Affymetrix contendo $p = 6817$

genes. O conjunto de dados compreende 47 tecidos ALL e 25 tecidos AML. O conjunto de dados foi preparado seguindo os passos descritos em (REF), isto é, (i) limiares: piso de 100 e teto de 16000; (ii) filtragem: exclusão de genes com $max/mim \leq 5$ e $max - mim \leq 500$, onde max e mim se referem, respectivamente, aos níveis de expressão máximos e mínimos de um gene em particular frente as amostras de mRNA; (iii) transformação logarítmica a base 10. O conjunto de dados resultante é uma matriz de expressão gênica de 72×3051, isto é, depois da preparação o número de genes é 3051.

Existem diversas técnicas na área de Reconhecimentos de Padrões para encontrar classificadores através de aprendizado supervisionado. Por motivos estatísticos e computacionais vamos utilizar O Discriminante Linear de Fisher (FLDA) para encontrar pares de genes que classificam perfeitamente os tecidos por tipo de câncer.

Seja x um ponto no plano formado pelos níveis de expressão de um par de genes (g_i, g_j) e y seja tal que $y = wx$, onde w é um vetor de pesos que melhor separa as projeções dos dois tipos de câncer (ALL e AML). A idéia por trás do FLDA é encontrar através do cálculo das projeções para um hiperplano (ou uma linha, neste caso), o vetor w que maximiza a razão da matriz de covariância inter classes S_B para a matriz de covariância intra classes S_W definida por:

$$\mathbf{S}_B = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^T \tag{5.1}$$

e

$$\mathbf{S}_W = \sum_{n \in C_1} (\mathbf{x}_n - \mathbf{m}_1)(\mathbf{x}_n - \mathbf{m}_1)^T + \sum_{n \in C_2} (\mathbf{x}_n - \mathbf{m}_2)(\mathbf{x}_n - \mathbf{m}_2)^T \tag{5.2}$$

onde: $\mathbf{m}_1 = \frac{1}{N_1} \sum_{n \in C_1} \mathbf{x}_n$ e $\mathbf{m}_2 = \frac{1}{N_2} \sum_{n \in C_2} \mathbf{x}_n$.

A solução deste problema é equivalente a inversão da matriz S_W multiplicada pela diferença das médias m_1 e m_2. Isto não é um discriminante, é apenas a direção do vetor normal da linha separadora entre os dois tipos de câncer o ponto de discriminação pode ser encontrado através de uma abordagem de Bayes ingênua.

O cálculo de um discriminante é bem rápido (cerca de 0.025s). Contudo há necessidade de calcular $C(3051, 2) = 4652775$ discriminantes e verificar se algum desses pares separam perfeitamente as amostras de câncer. Para cumprir esta tarefa pode-se dividir os trabalhos na a grade (devido sua independência) e apenas verificar os resultados. Existem 514 pares de genes que satisfazem o requisito de separar os tipos de câncer perfeitamente (A Figura . 5.1 mostra um exemplo destes pares).

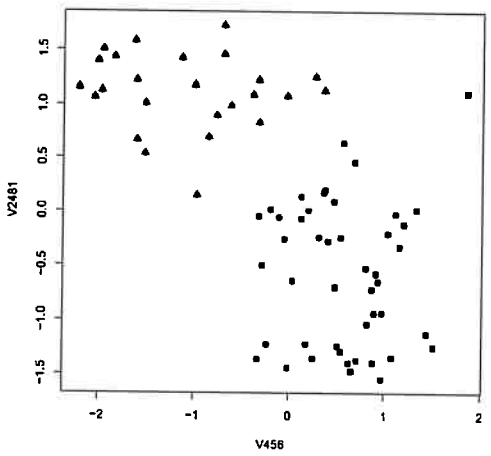

Figura 5.1: Um par de genes que separa perfeitamente os tipos de câncer (triângulos versus quadrados).

5.2 Preparação do ambiente

Para realizarmos os experimentos dois laboratórios didáticos foram preparados para executar uma grade Alchemi. Cada laboratório tem 20 computadores sendo: 20 Intel Celeron D330 de 2.66GHz com 256Mb RAM em um deles (denominado Lab1) e 20 Intel Celeron D315 de 2.25GHz com 256Mb RAM em outro (denominado Lab2). Ambos laboratórios rodam Windows XP SP2 e pelo menos com os seguintes pacotes instalados em cada máquina: R (versão 2.6.0), Servidor RDCOM (versão 1.8.3) e Alchemi (versão 1.0.6). As estações em cada laboratório são conectadas a um *hub ethernet* (10 Mbps) e os dois *hubs* estão conectados em cascata e conectados por um *switch* à rede administrativa. Durante os testes todas as estações estavam dedicadas ao experimento, isto é, não havia compartilhamento de ciclos entre outros usuários ou aplicações. Apesar de que o Middle-R possa ser utilizado de modo oportunista, foi configurado um ambiente dedicado para o experimento para melhor aferição do desempenho.

FALAR DO STAGING

5.3 Testes no modelo de jobs

Para testar a estabilidade do desempenho da solução, um subconjunto dos pares a serem testados foi escolhido (5000 pares de genes selecionados ao acaso) como parte de

um job que foi testado em três experimentos diferentes com um, dez e vinte nós.

O desempenho da aplicação da grade foi comparada com uma máquina *stand alone* (Lab1) que executou o script R para todos os 4652775 testes (mesmo conjunto de dados), isto é, fora da infraestrutura da grade. O experimento foi repetido usando uma grade com cinco, dez, vinte e quarenta nós. Para todos os experimentos exceto o último (40 nós), apenas as estações do Lab1 foram utilizadas. Inicialmente, o número de pares a ser testado foi dividido em 1000 jobs (aproximadamente 4.653 testes por job). A Tabela 5.1 mostra os resultados para cada um dos testes. Um efeito superescalar foi notado na comparação dos tempos de 5 nós contra 10 nós.

Nós	Tempo
$\star$	43:39:19
5	15:49:22
10	7:43:16
20	4:18:45
40	2:12:14

Tabela 5.1: Comparação de tempo entre os experimentos com grades de 5, 10, 20 e 40 verificando todos os pares divididos em 1000 *jobs*. A coluna Tempo está respresentada no formato HH:MM:SS.
($\star$) Tempo medido fora da infraestrutura da grade.

Para reduzir o impacto da comunicação entre o Alchemi e os nós, um segundo experimento foi feito. Desta vez o número de jobs é igual ao número de nós do experimento e cada job testa $4652775/N$, onde $N = 5, 10, 20, 40$. Neste segundo experimento, ao invés de pequenos jobs e muita comunicação na grade, foram utilizados jobs maiores e N comunicações.

A Tabela 5.2 mostra os resultados para cada um dos testes. Comparando a Tabela 5.1, o tempo total foi reduzido em 0,88% para 5 nós, 3,74% para 20 nós e 7,69% para 40 nós. Contudo, o tempo de execução para 10 nós não mostrou o efeito super escalar. Com estes dois experimentos é possível observar uma espécie de troca entre comunicação entre jobs e efeitos de disk swap de grandes jobs. Foram feitos alguns testes adicionais para verificar o sobre custo de comunicação. Em todos os casos exceto com 10 nós correspondeu a diferença de tempo. Assim sendo, é suposto que a diferença dos 10 nós é devido ao *disk swap*.

Nós	Tempo
$\star$	43:39:19
5	15:41:02
10	8:06:33
20	4:09:05
40	2:02:04

Tabela 5.2: Comparação de tempo entre experimentos com grades de 5, 10, 20 e 40 nós vericando todos os pares divididos em N *jobs* onde $N = \{5, 10, 20, 40\}$. A coluna Tempo está respresentada no formato HH:MM:SS.
($\star$) Tempo medido fora da infraestrutura da grade.

5.4 Testes de estabilidade

Todos os experimentos tiveram números de jobs igual ao número de nós e foram repetidos 100 vezes. Para uma grade de um nó, o tempo máximo de execução para este experimento foi de 2 minutos e 10 segundos, o tempo mínimo de execução foi de 2 minutos e 7 segundos e o desvio padrão foi de 0.828 segundos. Para a grade de dez nós, o tempo mínimo de execução foi de 2 minutos e 2 segundos, o tempo máximo de execução foi de 2 minutos e 14 segundos e o desvio padrão foi de 1,81 segundos. Para a grade de vinte nós, o tempo mínimo de execução foi 2 minutos e 8 segundos, o tempo máximo de execução foi 2 minutos e 16 segundos e o desvio padrão foi de 1,68 segundos.

Nós	Max.	Min.	σ
1	00:02:10	00:02:07	0,828s
10	00:02:14	00:02:08	1,812s
20	00:02:16	00:02:08	1,680s

Tabela 5.3: Comparação dos tempos (em segundos) máximos, mínimos e o desvio padrão dos testes de estabilidade.

5.5 Discussão dos Resultados

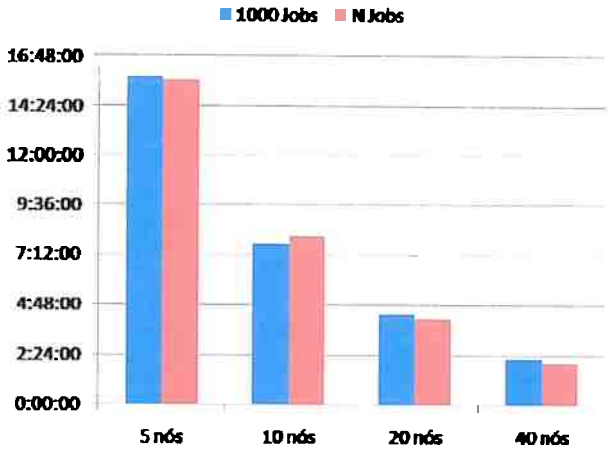

Figura 5.2: Resultados dos experimentos executados contendo 1000 *jobs* versus os experimentos contendo N *jobs*. O efeito superescalar pode ser notado no experimento com 10 nós.

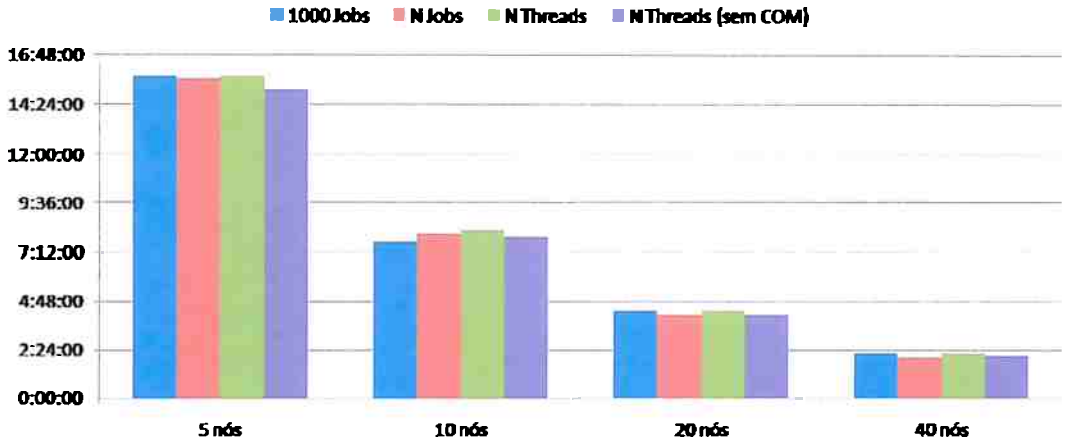

Figura 5.3: Resultados obtidos comparados.

CAPÍTULO 6

Conclusão

Existem várias soluções de grade para ambientes Unix e existem alguns pacotes para facilitar a distribuição de aplicações R sob o Unix. Sob o Windows, não existiam soluções "de prateleira"para distribuir o R quando este projeto foi iniciado. Felizmente, utilizando o Alchemi, um middleware de grade feito sobre a plataforma Microsoft .NET, foi possível projetar e implementar o MiddleR, um middleware de nível de usuário para possibilitar que aplicações R possam ser executas em um ambiente de grade.

Existe uma importante diferença entre a solução proposta neste trabalho e os pacotes existentes até então para paralelizar o R: devido ao fato do MiddleR utilizar o Alchemi, um middleware de grade, pode-se escalar a solução para uma grande rede de computadores distribuído através de domínios administrativos diferentes, o que seria muito difícil ou até impossível utilizando outras soluções. A solução proposta foi implantada em laboratórios didáticos de duas instituições educacionais, de modo a não interferir nas atividades normais dos usuários (que podem continuar utilizando a estação sem notar degradação de performance).

Nós utilizamos a grade em alguns de nossos problemas de bioinformática, em sua maioria análise estatística de dados de microarray para encontrar marcadores moleculares robustos (classificadores gênicos de poucos genes para distinção de tecidos com câncer e sem câncer, ou para distinção entre tipos de câncer). Nós pudemos resolver um importante problema relacionado à reprodutibilidade científica que é controlar as versões dos dados e dos pacotes do R instalados em um ambiente distribuído.

6.1 Trabalhos Futuros

Como trabalhos futuros, pretendemos explorar mais profundamente os efeitos dos tamanhos dos *jobs* no tempo total de execução, utilizando diferentes configurações como estações heterogêneas e redes mais rápidas.

Pretendemos ainda implementar soluções similares nas plataformas encontradas mais recentemente, especificamente o BOINC e o Condor. As soluções serão implantadas nos laboratório do Centro de Ensino de Computação (CEC) do Institudo de Matemática e Estatística, onde serão feitos testes com a mesma aplicação de exemplo para que posteriormente possam ser comparados com os resultados obtidos neste trabalho.

Referências Bibliográficas

[1] *Centro de inovação*, Último Acesso em Julho de 2009. Citado na(s) página(s) 5

[2] *.net framework developer center*, Último Acesso em Fevereiro de 2009. Citado na(s) página(s) 32

[3] *.net languages list*, Último Acesso em Fevereiro de 2009. Citado na(s) página(s) 32

[4] R. Ranjan A. Luther, R. Buyya e S. Venugopal, Alchemi: A .NET Based Enterprise Grid Computing System, *Proceedings of the 6th International Conference on Internet Computing (ICOMP'05)* (Las Vegas, USA), Junho de 2005. Citado na(s) página(s) 11

[5] C. Adams, S. Farrell, T. Kause e T. Mononen, *Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)*, RFC 4210, Setembro de 2005. Citado na(s) página(s) 23

[6] Gustavo Alonso, Fabio Casati, Harumi Kuno e Vijay Machiraju, *Web Services: Concepts, Architectures and Applications*, Springer, Novembro de 2003. Citado na(s) página(s) 11, 14

[7] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky e Dan Werthimer, Seti@home: an experiment in public-resource computing, *Commun. ACM* **45** (2002), no. 11, 56–61. Citado na(s) página(s) 14

[8] Parvin Asadzadeh, Rajkumar Buyya, Chun Ling Kei, Deepa Nayar e Srikumar Venugopal, Global Grids and Software Toolkits: A study of four grid middleware technologies, *High-Performance Computing: Paradigm and Infrastructure*, Wiley, November de 2005. Citado na(s) página(s) 4

[9] Thomas Baier e Eric Neuwirth, *R COM Client Interface and internal COM Server*, 2006, R package version 1.4. Citado na(s) página(s) 28

[10] Tim Banks, *Web Services Resource Framework (WSRF) - Primer v1.2*, Committee Draft 02, Maio de 2006, wsrf-primer-1.2-primer-cd-02. Citado na(s) página(s) 15, 23

[11] R. Buyya e S. Venugopal, The Gridbus toolkit for service oriented grid and utility computing: an overview and status report, *Grid Economics and Business Models, 2004. GECON 2004. 1st IEEE International Workshop on*, 2004, pp. 19–66. Citado na(s) página(s) 6, 17

[12] Rajkumar Buyya, Economic-based distributed resource management and scheduling for grid computing, *CoRR* **cs.DC/0204048** (2002). Citado na(s) página(s) 5

[13] Microsoft Corporation, *Web Services Enhancements (WSE)*. Citado na(s) página(s) 23

[14] T. Dierks e E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2*, RFC 5246, Agosto de 2008. Citado na(s) página(s) 23

[15] Evgenia Dimitriadou, Kurt Hornik, Friedrich Leisch, David Meyer, e Andreas Weingessel, *e1071: Misc functions of the department of statistics (e1071), tu wien*, 2006, R package version 1.5-13. Citado na(s) página(s) 49

[16] Jack J. Dongarra, Jack J. Dongarra, Hans W. Meuer, Hans W. Meuer, Erich Strohmaier e Erich Strohmaier, *Top500 supercomputer sites*, Tech. report, Supercomputer, 1996. Citado na(s) página(s) 5

[17] Jack J. Dongarra, Rolf Hempel, Anthony J. G. Hey e David W. Walker, *Mathematical Sciences Section: A PROPOSAL FOR A USER-LEVEL, MESSAGE PASSING INTERFACE IN A DISTRIBUTED MEMORY ENVIRONMENT*, Fevereiro de 1993. Citado na(s) página(s) 5, 23

[18] Todd Tannenbaum Douglas Thain e Miron Livny, Distributed computing in practice: The condor experience, *Concurrency and Computation: Practice and Experience*, vol. 17, 2005, pp. 323–356. Citado na(s) página(s) 7

[19] Jeff Duntemann, *Assembly language: step-by-step*, John Wiley & Sons, Inc., New York, NY, USA, 1992. Citado na(s) página(s) 37

[20] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach e T. Berners-Lee, *Hypertext Transfer Protocol – HTTP/1.1*, 1999. Citado na(s) página(s) 22

[21] Global Grid Forum, *Open Grid Services Infrastructure (OGSI) Version 1.0*, Junho de 2003, GFD-R-P15 (Proposed Recomendation). Citado na(s) página(s) 15, 23

[22] I. Foster, J. Geisler, B. Nickless, W. Smith e S. Tuecke, Software infrastructure for the I-WAY high-performance distributed computing experiment, *High-Performance Distributed Computing, International Symposium on* **0** (1996), 562. Citado na(s) página(s) 3

[23] I. Foster e C. Kesselman, Globus: A toolkit-based grid architecture, *The Grid: Blueprint for a Future Computing Infrastructure*, MORGAN-KAUFMANN, 1998, pp. 259–278. Citado na(s) página(s) 4

[24] I. Foster e C. Kesselman (eds.), *The Grid 2: Blueprint for a Future Computing Infrastructure*, 2ª ed. ed., MORGAN-KAUFMANN, 2003. Citado na(s) página(s) 3

[25] I. Foster, C. Kesselman, J. Nick e S. Tuecke, *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*, 2002. Citado na(s) página(s) 15, 23

[26] Ian Foster, *Designing and building parallel programs: Concepts and tools for parallel software engineering*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. Citado na(s) página(s) 4, 21

[27] Ian Foster, Carl Kesselman, Gene Tsudik e Steven Tuecke, A security architecture for computational grids, *CCS '98: Proceedings of the 5th ACM conference on Computer and communications security* (New York, NY, USA), ACM, 1998, pp. 83–92. Citado na(s) página(s) 23

[28] N. Freed e N. Borenstein, *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, 1996. Citado na(s) página(s) 22

[29] Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides, *Design patterns: elements of reusable object-oriented software*, Addison-Wesley Professional, 1995. Citado na(s) página(s) 19

[30] A. Geist, A. Beguelin, Jack Dongarra, W. Jiang, R. Manchek e V. Sunderam, *PVM Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge, Mass., 1994. Citado na(s) página(s) 5, 23

[31] Robert Gentleman, Reproducible Research: A Bioinformatics Case Study, *Statistical Applications in Genetics and Molecular Biology* **4** (2005), no. 1, Article 2. Citado na(s) página(s) 48

[32] Robert C Gentleman, Vincent J. Carey, Douglas M. Bates, Ben Bolstad, Marcel Dettling, Sandrine Dudoit, Byron Ellis andLaurent Gautier, Yongchao Ge, Jeff Gentry, Kurt Hornik, Torsten Hothorn, Wolfgang Huber, Stefano Iacus, Rafael Irizarryand

Friedrich Leisch Cheng Li, Martin Maechler, Anthony J. Rossiniand Gunther Sawitzki, Colin Smith, Gordon Smyth, Luke Tierneyand Jean Y. H. Yang e Jianhua Zhang, Bioconductor: Open software development for computational biology and bioinformatics, *Genome Biology* **5** (2004), R80. Citado na(s) página(s) 28

[33] Andrei Goldchleger, Fabio Kon, Alfredo Goldman, Marcelo Finger e Germano Capistrano Bezerra, Integrade: Object-oriented grid middleware leveraging idle computing power of desktop machines, *Concurrency and Computation: Practice and Experience* **16** (2004), no. 5, 449–459. Citado na(s) página(s) 4

[34] Daniel Lombraña González, Francisco Fernández de Vega, L. Trujillo, G. Olague, M. Cárdenas, L. Araújo, P. Castillo, K. Sharman e A. Silva, Interpreted applications within boinc infrastructure, *Ibergrid 2008. 2nd Iberian Grid Infrastructure Conference Proceedings*, 2008, pp. 261–272. Citado na(s) página(s) 7

[35] A.S. Grimshaw e W.A. Wulf, Legion: A view from 50,000 feet, *5th International Symposium on High Performance Distributed Computing (HPDC '96)* (Syracuse, NY, USA), August de 1996. Citado na(s) página(s) 4

[36] Marty Humphrey e Mary R. Thompson, Security implications of typical grid computing usage scenarios, *HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing* (Washington, DC, USA), IEEE Computer Society, 2001, p. 95. Citado na(s) página(s) 23

[37] J. Kohl e C. Neurnan, *The Kerberos Network Authentication Service (V5)*, Internet RFC 1510, Setembro de 1993. Citado na(s) página(s) 23

[38] Stefan M. Larson, Christopher D. Snow, Michael Shirts, Vijay S. P e Vijay S. Pande, *Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology.* Citado na(s) página(s) 14

[39] Na Li e A. J. Rossini, *rpvm: R interface to PVM (Parallel Virtual Machine)*, 2006, R package version 1.0.1. Citado na(s) página(s) 6, 28

[40] M. Litzkow, M. Livny e M. Mutka, Condor: A hunter of idle workstations, *Proceedings of the 8th International Conference of Distributed Computing Systems (ICDCS)*, 1988. Citado na(s) página(s) 4, 7

[41] Akshay Luther, Rajkumar Buyya, Rajiv Ranjan e Srikumar Venugopal, *Alchemi: A .NET-based Grid Computing Framework and its Integration into Global Grids*, Technical Report GRIDS-TR-2003-8, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, Dezembro de 2003. Citado na(s) página(s) 11

[42] ———, High Performance Computing: Paradigm and Infrastructure, ch. Peer-to-Peer Grid Computing and a .NET-based Alchemi Framework, Wiley Press, New Jersey, USA, Junho de 2005. Citado na(s) página(s) 7, 11, 12

[43] Scott McLean, Kim Williams e James Naftel, *Microsoft .Net Remoting*, Microsoft Press, Redmond, WA, USA, 2002. Citado na(s) página(s) 11

[44] Erik Meijer e Jim Miller, *Technical overview of the Common Language Runtime*, Tech. report, Microsoft Corp., 2000. Citado na(s) página(s) 32

[45] Adam Nathan, *.net and com: The complete interoperability guide*, Pearson Education, 2002. Citado na(s) página(s) 36

[46] E. O'Tuathail e M. Rose, *Using the Simple Object Access Protocol (SOAP) in Blocks Extensible Exchange Protocol (BEEP)*, 2002. Citado na(s) página(s) 22, 34

[47] David S. Platt, *Introducing microsoft .net, second edition*, Microsoft Press, Redmond, WA, USA, 2002. Citado na(s) página(s) 32

[48] R Development Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2006, ISBN 3-900051-07-0. Citado na(s) página(s) 27

[49] Ingo Rammer, *Advanced .net remoting*, Apress, Berkely, CA, USA, 2002. Citado na(s) página(s) 34

[50] Ward Rosenberry, David Kenney e Gerry Fisher, *Understanding dce*, O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1992. Citado na(s) página(s) 30

[51] A. Rossini, L. Tierney e N. Li, Simple parallel statistical computing in R, *UW Biostatistics working paper series* (2003), Paper 193. Citado na(s) página(s) 6, 28

[52] D B Skillicorn, J M D Hill e W F McColl, *Questions and Answers on BSP*, Technical Report PRG-TR-15-96, Oxford University Computing Laboratory, Agosto de 1996. Citado na(s) página(s) 23

[53] J. A. Smith e S. K. Shrivastava, A System for Fault-Tolerant Execution of Data and Compute Intensive Programs over a Network of Workstations, *Lecture Notes in Computer Science* **1123** (1996), 487. Citado na(s) página(s) 21

[54] Marc Snir e Steve Otto, *MPI-The Complete Reference: The MPI Core*, MIT Press, Cambridge, MA, USA, 1998. Citado na(s) página(s) 5, 23

[55] Hao Yu, *Rmpi: Interface (Wrapper) to MPI (Message-Passing Interface)*, 2006, R package version 0.5-3. Citado na(s) página(s) 6, 28