Trabalho de Conclusão de Curso

Rodrigo L. M. Flores Orientador: Roberto Hirata Jr.

23 de novembro de 2009

Parte I

1 Introdução

O ponto de partida para este projeto é a necessidade de se fazer processamentos custosos de dados em bioinformática. Estes processamentos costumam ser combinatórios, o que os fazem demorar um bom tempo para serem executados em um computador comum, sendo necessário utilizar recursos computacionais de alta performance para obter os resultados em tempo hábil.

1.1 Programação paralela e distribuída

Uma abordagem clássica para se resolver esse problema é utilizar *clusters*, que são um grupo de computadores ligados entre si de modo a parecer ser um único computador muito mais potente. *Clusters* podem ser tanto máquinas específicas para isso, produzidas com um alto custo e com um hardware específico para otimizar seu desempenho, ou pode ser utilizado o processamento em grade: utilizando computadores pessoais que são produzidos em massa a um preço mais baixo e trabalhando em paralelo para fazer o processamento.

O projeto *Beowulf* é um dos exemplos de computação em grade: computadores pessoais baratos constituem uma grade dedicada que funciona como um super computador. De acordo com o projeto Beowulf, uma rede deste tipo provê o mesmo recurso computacional que um super computador custando de 3 a 10 vezes mais.

1.2 Computação oportunista

A computação oportunista é um tipo de computação distribuída no qual pessoas que possuem computadores podem doar processamento e armazenamento ocioso de suas máquinas. Estes projetos normalmente têm um objetivo bem definido.

O primeiro projeto de Computação voluntária foi o *Great Internet Mersenne Prime Search*, lançado em janeiro de 1996. Seu objetivo era encontrar números primos de Mersenne[1]. Em seguida houveram muitos outros projetos e alguns utilizam um apelo social para obter mais doadores de processamento, um exemplo disso é o o Folding At Home, que investiga o enrolamento de proteínas e que pode ajudar o desenvolvimentos de pesquisas contra Câncer, doença de Huntington, entre outras.

Um dos projetos mais notáveis de computação voluntária foi o *SETI@Home*. Os objetivos iniciais deste projeto eram:

[1]Isto é, números primos na forma $M_n = 2^n - 1$

- Provar a viabilidade e a praticidade de grades computacionais distribuídas;
- Fazer um trabalho útil e apoiar uma análise de observações para detectar vida inteligente fora da Terra.

Este projeto atraiu centenas de milhares de voluntários, mas só o primeiro objetivo teve sucesso: não foram encontrados sinais de vida inteligente fora da Terra. Como um produto do primeiro obejetivo, nasceu o *middleware BOINC* foi um dos produtos obtidos deste projeto e hoje é utilizado em diversos projetos.

1.3 Linguagens Interpretadas

Uma das possíveis divisões para linguagens de programação é se seus códigos são compilados para código de máquina ou se são simplismente interpretados. Enquanto no primeiro caso é necessária a utilização de um programa para transformar o código fonte em código de máquina para ele poder então ser executado, mo segundo caso, há a figura de um interpretador que não converte o programa para código de máquina, mas sim o interpreta e o executa.

Linguagens interpretadas possuem vantagens e desvantagens: embora elas sejam mais fáceis de serem multiplataforma (basta o interpretador estar disponível para aquela plataforma) e permitam escopo e tipagem dinâmica, também costumam ser menos eficientes que linguagens compiladas e a presença de um interpretador é obrigatória para sua execução.

Dentre as linguagens interpretadas, uma que adquiriu destaque na área de estatística e bioinformática é a linguagem *R*, que também pode ser utilizado como um ambiente. Esta linguagem já vem com as distribuições estatísticas mais usuais e possui uma extensibilidade poderosa, com bibliotecas que podem ser baixadas facilmente.

1.4 Linguagens interpretadas e computação distribuída

Embora já existam ótimas soluções para a execução distribuída de programas compilados como o MPI[2], não se fala muito em soluções para execução distribuída de programas em linguagem interpretada. Para a linguagem *R* há um pacote chamado *gridR* que permite o uso do R com o *Condor*, um middleware para execução de programas em grades. Um outro trabalho que relaciona o R com computação distribuída é o [RAD09]: é utilizado o middleware baseado na tecnologia .NET *Alchemi* e a interface *COM*, junto com o pacote do *RCom*.

O artigo [GGdVS08] fala sobre a utilização do Middleware de computação voluntária BOINC como solução para computação em grade na Universidade de

[2]disponível em `http://www.mcs.anl.gov/research/projects/mpi/`

Extremadura na Espanha. Dentre os programas executados na grade, haviam programas em R. Porém isso foi somente instalado em redes de computadores com computadores cujo sistema operacional é o Linux.

Seguindo este feito, este trabalho tem a intenção de construir algo semelhante nos laboratórios da rede CEC do IME/USP. Utilizando não somente os computadores executando Linux, mas também os computadores cujo sistema operacional é o Microsoft Windows já que estes são uma parte considerável da rede.

2 Conceitos e tecnologias utilizadas

O desenvolvimento do projeto incluiu diversas tecnologias, sendo as principais a linguagem de Programação *R* e o middleware para computação voluntária *Boinc*.

2.1 BOINC

O BOINC, cujo nome é uma sigla para *Berkeley Open Infrastructure for Network Computing*, é um middleware para computação em grade e voluntária e foi criado na Universidade de Berkeley, Califórnia, Estados Unidos.

Inicialmente, o projeto consistia em gerenciar o projeto *SETI@HOME* que possuía dois objetivos:

- Provar a viabilidade e a praticidade do conceito "computação em grade distribuída";
- Fazer um trabalho científico útil fazendo uma análise observacional para detectar vida inteligente fora da Terra.

O primeiro objetivo foi concluído com sucesso e o resultado é o *BOINC*. O segundo falhou: nenhuma evidência de vida inteligente fora da Terra foi encontrada.

2.1.1 Funcionamento do BOINC

Cada unidade de processamento no Boinc é chamada de *workunit* e é constituida de arquivos executáveis e arquivos de entrada. Depois de processado, os arquivos de saída gerados são enviados para o servidor que normalmente armazena estes arquivos em um banco de dados ou em um arquivo.

Para gerar um workunit são necessários dois arquivos XML, um deles detalhando a entrada e o outro detalhando a saída. Para facilitar a escrita do programa, precisamos escrever para cada arquivo um nome lógico que ao enviar e receber

o cliente renomeia o arquivo. Por exemplo, temos um programa que lê um arquivo chamado `input` e escreve no arquivo `output`, para podermos ter muitos arquivos de entrada com nomes diferentes, quando criamos uma *workunit*, o servidor coloca um nome único e semelhante ao da workunit nos arquivos de entrada e saída que serão renomeados pelo cliente para o nome lógico.

O processamento é realizado pelo cliente: o arquivo binário é executado e enquanto ele é executado há um checkpoint que permite em caso de interrupções retomar o processamento de um determinado ponto. Finalizado o processamento, na próxima atualização o cliente avisará ao servidor que o processamento foi finalizado. Um diagrama do funcionamento pode ser visto na figura 1.

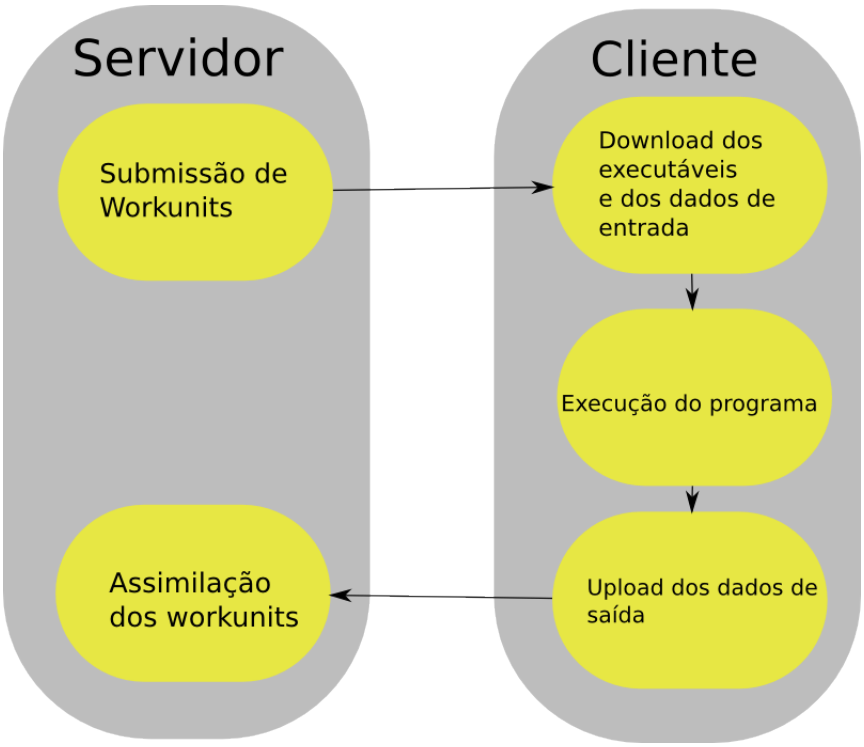

Figura 1: Funcionamento do Boinc

2.1.2 Wrapper

O *Wrapper* é um programa escrito utilizando a *api* do *BOINC*, cujo objetivo é executar aplicações legadas, i.e. aplicações que não utilizam a API do *BOINC*, utilizando o *BOINC*. Há uma versão do Wrapper distribuída junto com o *BOINC* que utiliza um arquivo XML, mas existe uma outra opção descrita no artigo que utiliza um shell para a execução dos aplicativos.

O arquivo XML de execução tem a seguinte estrutura:

```
<job_desc>
    <task>
        <application>foobar</application>
        [ <stdin_filename>stdin_file</stdin_filename> ]
        [ <stdout_filename>stdout_file</stdout_filename> ]
        [ <stderr_filename>stderr_file</stderr_filename> ]
        [ <command_line>--foo bar</command_line> ]
    </task>
    [ ... ]
</job_desc>
```

Neste XML, o único campo obrigatório é o *application*, que é a aplicação que será executada e pode ser distribuída junto com a aplicação ou já existir no computador que o cliente estará instalado (para este segundo caso é necessário informar o caminho inteiro do executável). É possível ter mais de uma tag task, e o wrapper as executará sequencialmente. É de responsabilidade do *Wrapper*

2.2 R

A linguagem de programação estatística R é uma implementação da linguagem S e foi criada por John Ihaka e Robert Gentleman na Universidade de Auckland, da Nova Zelândia. Hoje, a linguagem R é uma linguagem considerada padrão entre estatísticos no desenvolvimento de softwares estatísticos e na análise de dados. Há também um ambiente onde podemos utilizar a linguagem em um interpretador e gerar gráficos de alta qualidade.

Por padrão, as distribuições estatísticas mais populares podem ser utilizadas e é muito mais simples que em linguagens como C, Java a manipulação de matrizes e tabelas de dados.Outro ponto importante na linguagem R é a extensibilidade: é muito simples instalar bibliotecas. Hoje em dia, há cerca de 2000 bibliotecas no repositório principal (chamado de *CRAN*, sigla de *Comprehensible R Archive Network*) com as mais diversas funções (desde bibliotecas de gráficos específicos até métodos estatísticos mais complexos). Outro repositório bastante utilizado é o Bioconductor, que provê rotinas bastante utilizadas para o processamento de dados da área de bioinformática.

3 Atividades realizadas

O início do projeto deu-se ainda em 2008, com a visita ao colégio Rainha da Paz na Lapa, onde o aluno de mestrado do *IME* Rodrigo Assirati Dias mantém uma

grade de computadores com o middleware *Alchemi* citada no trabalho . Nesta visita foi possível esclarecer dúvidas, entender o funcionamento da grade e receber algumas dicas quanto à manutenção da grade. Após esse encontro, começou-se a buscar alternativas para o a computação de alta performance com o *R*. Um primeiro pacote encontrado que fazia esta função foi o *GridR*, que permite submeter rotinas do *R* para *clusters*, máquinas remotas e grades. Um dos arcabouços possíveis para o uso deste pacote é o Condor , desenvolvido pela *University of Wisconsin-Madison* e é bastante utilizado em empresas de grande porte como a *NASA* e pode ser executado tanto em sistemas baseados em *UNIX*, como em sistemas *Windows*. Seguindo esta busca, encontramos o *middleware BOINC*, no artigo sendo utilizado para um propósito semelhante em um trabalho na Universidade de Extremadura, na Espanha e decidimos que a abordagem seria interessante para nosso trabalho.

Escolhido o *middleware* nos focamos na instalação do servidor. A própria página do *BOINC* possui um guia de instalação do servidor do *middleware* no sistema *Debian GNU*
Linux e por esta distribuição *Linux* ser bastante conhecida por sua estabilidade, foi instalado este sistema no servidor. Instalado o servidor, o foco foi em ter uma aplicação em R executando remotamente em uma grade de computadores. Este processo no sistema Linux foi relativamente simples: utilizando o "truque do *shebang*" é possível colocar um script como executável e o *wrapper* executá-lo como se fosse um arquivo compilado. Já para o sistema Windows o trabalho foi mais complicado: havia um bug nas configurações de compilação do *wrapper* e até perceber isso atrasou bastante o andamento do projeto. Passado isso, foi necessário utilizar um programa escrito em C, que apenas executava o interpretador junto com o arquivo com a rotina em *R*.

Finalizado esta parte, nos focamos na aplicação a ser executada na grade. Para isso foi criado um programa em *R* que apenas acessava um arquivo e fazia alguns cálculos custosos. Isto foi então configurado para o mesmo programa poder ser executado tanto em sistemas Windows como em sistemas Linux. Paralelamente a isso, foi pedido para a administração da rede do *CEC* para instalar o *BOINC* nas máquinas da rede.

4 Resultados e produtos obtidos

O principal resultado deste trabalho foi fazer o *Boinc* funcionar com rotinas em *R*, tanto nos ambientes *Linux* como no ambiente *Windows*

4.1 Sistema Linux

Para o ambiente *Linux*, foi relativamente simples o processo: dado um arquivo com rotinas do *R* a serem executadas, é somente necessário alterar a permissão do arquivo para executável (via `chmod +x arquivo.R`) e adicionar a seguinte linha na primeira linha do arquivo:

```
#!/usr/bin/Rscript
```

Isso faz um sistema *Linux* chamar o interpretador *Rscript* para interpretar as rotinas em *R* e assim executar o arquivo. Esta solução permite não só que rotinas em *R* sejam executadas, mas sim qualquer script que tenha seu interpretador descrito na primeira linha.

A esta solução, demos o nome de *Truque do Shebang*, pelos caracteres `#!` serem chamados de *shebang*. Porém, para termos a mesma solução em ambos os sistemas, utilizamos a solução para Windows no sistema Linux.

4.2 Sistema Windows

Como o sistema Windows não permite utilizar scripts utilizando o *shebang*, foi necessário utilizar um programa compilado escrito na linguagem C, que chamamos de *Runner*, que usando a função `system`, chama o interpretador com o arquivo.R como parâmetro.

Esta solução permite inclusive que usemos scripts diferentes de R a cada vez que criamos um *workunit*, assim como adicionar arquivos extra que por ventura fossem necessários para o processamento. Esta maneira também funciona no *Linux*, só que o *Runner* precisar ser compilado para o *Linux* com o caminho para o interpretador correto. O programa não faz uma verificação para perceber se o *R* está instalado.

A utilização do *Runner* foi necessário devido ao *Wrapper* não perceber corretamente que o interpretador foi executado sem erros e mesmo em execuções sem erros, o *wrapper* recebia um valor de retorno do interpretador diferente de zero, o que ele percebia como um erro e marcava o *workunit* como inválido.

4.3 Discussão

4.3.1 Vantagens

As principais vantagens no uso do *Boinc* para o processamento em grade são:

- **Facilidade de se adicionar novos nós** - A instalação do BOINC em ambos os sistemas Linux e Windows é simples e fácil de ser feita e nenhuma ação

no servidor é necessária a cada instalação de nós. Além disso, é muito simples fazer a replicação de configurações, tanto para o processamento, quanto para a conexão com o servidor para os computadores;

- **Processamento multiplataforma** - Para a grade funcionar na plataforma só são necessárias duas coisas: que o BOINC e o *R* estejam disponível para a plataforma. As plataformas mais comuns (Linux 32 e 64 bits e Microsoft Windows) têm ambos os projetos disponíveis;

- **Código aberto** - A utilização de dois softwares com código aberto facilita bastante: a busca de bugs se torna possível, a gratuidade dos softwares e a grande quantidade de documentação, muitas vezes produzidas por pessoas que não necessariamente são da equipe de desenvolvimento do *BOINC*. A mentalidade de ajuda mútua, existente nas listas de discussão na no canal de IRC do projeto também é de grande ajuda;

- **Execução invisível ao usuário** - Com o *BOINC* configurado para isso, um usuário comum da rede nem ao menos percebe a existência de um processamento em grade. É possível configurar o *BOINC* para só começar a execução com o computador ocioso há um número arbitrário de minutos. Também é possível configurar para o processamento só acontecer em determinadas faixas de horários e dias da semana. Outra configuração interessante é a determinação do máximo de memória *RAM* e de espaço em disco para a execução.

- **Solução funciona para qualquer linguagem de script** - De maneira análoga, é possível executar qualquer programa escrito em linguagem interpretada com o BOINC utilizando esta mesma solução. Como comentado antes, só é necessário que exista uma versão do interpretador para as plataformas necessárias (o que é comum para as linguagens mais utilizadas como *PERL*, *Python* e *Ruby* e as plataformas mais comuns).

4.3.2 Desvantagens

As principais desvantagens são:

- **Necessidade de se ter o *R* instalado** - O *R* não é uma linguagem instalada por padrão nas distribuições Linux mais populares, nem no *Windows*. Então, a adição de um nó só pode ser feita se o *R* for também instalado.

- **Falta de checkpoints** - Utilizando um aplicativo feito com a *API* do *BOINC* é possível se ter *Checkpoints*, que são uma maneira de uma aplicação feita com o *BOINC* reiniciar o processamento não do início, mas sim de um

determinado ponto. Utilizando o *Wrapper* e o *R*, perdemos esse recurso. A computação de rotinas longas se torna mais difícil e pouco recomendada, já que a cada interrupção o processamento é reiniciado.

- **Falta de "compromisso" fixo dos clientes** - Diferentemente da rede citada no artigo não há a figura de um computador *Manager* que gerencia as máquinas, atualizando a qualquer momento, mas sim um servidor que apenas envia e recebe as tarefas e a iniciativa de computação fica com os computadores da grade.

4.4 Andamento da rede

5 Conclusões

Referências

[GGdVS08] D.L. Gonzalez, G.G. Gil, F.F. de Vega, and B. Segal, *Centralized boinc resources manager for institutional networks*, April 2008, pp. 1–8.

[RAD09] Roberto Hirata Jr. Rodrigo A. Dias, *Middle-r - a user level middleware for statistical computing*, VII Workshop on Grid Computing and Applications (2009).

Parte II
Parte subjetiva

6 Desafios e frustrações encontrados

O curso de bacharelado em ciência da computação é um curso bastante denso e dificilmente temos tempo para fazer todas as tarefas de todas as disciplinas. Então acredito que o meu maior desafio nestes anos de IME foi conciliar todas as tarefas e disciplinas, e infelizmente descartando alguma as vezes.

7 Disciplinas relevantes para o trabalho

Diversas disciplinas foram relevantes para este trabalho:

- **MAC110 -**
- **MAC122 -**
- **MAC211 - Laboratório de programação I** Ferramentas como versionamento de código, processamento de texto e o makefile foram essenciais para a elaboração deste trabalho de forma indireta e contribuiram com a boa qualidade do mesmo.
- **MAC242 - Laboratório de programação II** Linguagens de script facilitam bastante o trabalho de tarefas repetitivas e o boa parte do que sei sobre este tipo de linguagem eu aprendi neste curso.
- **MAC338 - Análise de algoritmos** Este curso contribuiu indiretamente com minha formação como cientista da computação i e muitos dos conceitos aprendidos neste curso ajudaram o entendimento melhor de algoritmos e soluções.
- **Programação paralela e distribuída**
- **MAC422 - Sistemas Operacionais** Saber como um programa é executado, como o sistema gerencia essas execução e como a tabela de processos funciona é essencial quando se trabalha com uma grade de computadores.