

Trabalho de conclusão de curso
BOINC + R: Executando rotinas de
bioinformática em grades oportunistas

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

8 de fevereiro de 2010

Sumário

I	Parte objetiva	3
1	Introdução	3
1.1	Programação paralela e distribuída	3
1.2	Computação oportunista	3
1.3	Linguagens Interpretadas	4
1.3.1	Linguagem R	4
1.4	Linguagens interpretadas e computação distribuída	5
1.5	Solução multiplataforma	5
1.6	Utilização do <i>BOINC</i> com o <i>R</i>	6
2	Conceitos e tecnologias utilizadas	7
2.1	BOINC	7
2.1.1	Funcionamento do BOINC	8
2.1.2	Wrapper	9
2.2	R	10
2.3	Utilização do <i>BOINC</i> para o processamento de rotinas na linguagem <i>R</i>	13
3	Atividades realizadas	14
4	Resultados e produtos obtidos	15
4.1	Implementação	15
4.1.1	Sistema Linux	15
4.1.2	Sistema Windows	15
4.2	Discussão	17
4.2.1	Vantagens	17
4.2.2	Desvantagens	17
4.3	Instalação da rede	18
4.4	Testes	19
5	Conclusões	20
II	Parte subjetiva	26
6	Desafios e frustrações encontrados	26
7	Disciplinas relevantes para o trabalho	26

8	Conceitos Aplicados	27
9	Continuação do trabalho	27
10	Agradecimentos	28

Parte I

Parte objetiva

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 mais famosos: 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 mas custando de um décimo a um terço do preço. Outro exemplo de computação em grade são os projetos de computação oportunista ou voluntária, não necessariamente usam computadores de forma dedicada mas utilizam o tempo ocioso do computador.

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 com um apelo humanitário (ou científico).

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 Mersenne¹. Após este projeto, nasceram vários outros. Entre os pro-

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

jetos mais destacados podemos citar o *Folding At Home*, que investiga o enro-lamento de proteínas e que pode ajudar o desenvolvimentos de pesquisas contra Câncer, doença de Huntington, entre outras. Outro projeto notável de computação voluntária, e de grande importância para este trabalho, é o *SETI@Home* que atraiu centenas de milhares de voluntários de vida inteligente fora da Terra. Como um produto deste projeto, nasceu o *middleware BOINC* que 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 simplesmente interpretados. Enquanto no primeiro caso há a figura de um compilador, que transforma o código fonte em código de máquina para ele poder então ser executado, no segundo caso, há a figura de um interpretador que não converte o programa para código de máquina, mas sim o interpreta diretamente.

Linguagens interpretadas possuem vantagens e desvantagens sobre as compiladas: 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 será de fundamental importância para este trabalho.

1.3.1 Linguagem R

A linguagem *R* é uma linguagem interpretada bastante utilizada no desenvolvimento de rotinas de bioinformática e na análise de dados. Muitas funções normalmente utilizadas em análises de dados vêm incluídas na linguagem como por exemplo cálculo da média, desvio padrão, ajuste de curva, entre outras. A geração de gráficos do *R* também é bastante elaborada e é possível gerar gráficos dos mais diversos tipos. Também há implementações de algoritmos mais elaborados de estatística como por exemplo algoritmos de *clustering* que também são bastante usados em bioinformática.

Outro ponto interessante da linguagem *R* são os pacotes extras que se pode baixar. Ao todo, no repositório oficial, são 2076 pacotes, para os mais diversos propósitos que podem ser desde funções mais específicas para análise como por exemplo o *Bayesclust*², como bibliotecas para geração de gráficos específicos ou

²Disponível em <http://cran.r-project.org/web/packages/bayesclust/>

utilizando alguma biblioteca externa. como por exemplo o *rggobi*³.

Com maior destaque para área de bioinformática, o artigo [GCB⁺04] fala sobre o *Bioconductor*, um projeto que propõe o fornecimento de ferramentas para a análise de dados desta área. O projeto foi iniciado em 2001 e a grande maioria de seus pacotes são para o ambiente de programação da linguagem *R*.

1.4 Linguagens interpretadas e computação distribuída

Embora já existam ótimas soluções para a execução distribuída de programas compilados, como por exemplo o MPI⁴, 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 middleware *Condor*, que é uma solução bastante conhecida para execução de programas em grades. Um outro trabalho que relaciona o *R* com computação distribuída é o [RAD09] que utiliza o middleware *Alchemi*, cuja finalidade é fazer processamento em grade e é baseado na tecnologia *.NET* da Microsoft, e a interface *COM* junto com o pacote do *R RCom*.

O artigo [GGdVS08] fala sobre a utilização do Middleware de computação voluntária *BOINC* como solução para computação em grade na Universidade de Extremadura na Espanha e dentre os programas executados na grade, haviam programas em *R*. Porém isso foi somente instalado em redes de computadores cujo sistema operacional é o *Linux*.

1.5 Solução multiplataforma

Embora o ambiente *Linux* seja muito utilizado no ambiente acadêmico e em ambientes de desenvolvimento de softwares, é muito difícil encontrá-lo em um ambiente doméstico ou de trabalho cotidiano. Embora a cada ano o número de usuários deste sistema cresça, percebemos que o sistema normalmente utilizado é o *Microsoft Windows*. O artigo [RAD09] dispõe uma solução usando o *middleware Alchemi*, baseado na tecnologia da Microsoft *.NET* que é executado no sistema operacional *Microsoft Windows*. O artigo [GGdVS08] porém utiliza uma rede com computadores rodando *Linux*

Porém como temos muitos computadores com sistemas *Linux* e *Windows* na rede *CEC* do IME-USP, pensamos que o ideal seria termos uma solução que pudesse utilizar os dois sistemas. Como tanto o *BOINC* como o *R* estão disponíveis

index.html

³Disponível em <http://cran.r-project.org/web/packages/rggobi/index.html>

⁴disponível em <http://www.mcs.anl.gov/research/projects/mpi/>

para os dois sistemas, um dos objetivos deste trabalho é utilizar os dois sistemas na grade.

1.6 Utilização do *BOINC* com o *R*

Embora a *API* do *BOINC* não esteja disponível para ser utilizada em programas escritos na linguagem *R*, a utilização destas duas tecnologias juntas mostra algumas vantagens. Como ambas as tecnologias são de código aberto, não se gasta na compra de licenças. A possibilidade de se executar rotinas inalteradas e de instalar pacotes do *R* sob demanda utilizando o *wrapper* também foi um fator decisivo na escolha. A utilização bem sucedida do processamento de rotinas em *R* na grade descrita no artigo [GGdVS08] também foi um fator determinante nesta decisão, além da disponibilidade multiplataforma de ambas as tecnologias.

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*. Dentre os conceitos estudados, podemos destacar a computação em grade.

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: 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 até hoje.

Dentre os diversos motivos para a utilização do *BOINC*, baseados no artigo [GGdVS08], podemos destacar:

- **Mais utilizado** - Quando comparado com outros *middlewares* semelhantes como o *middleware Condor* ou o *Xtremweb*, o *BOINC* é mais utilizado e há pacotes para o *BOINC* nas distribuições *Linux* mais populares;
- **Amplamente utilizado** - O *BOINC* é utilizado em diversas áreas como previsão do tempo, física, astrofísica, biologia, entre outras;
- **Suporte da comunidade** - Baseado no espírito de ajuda mútua existente em comunidades de software livre, é possível ter dúvidas esclarecidas quanto ao funcionamento do *BOINC* de maneira fácil e desburocratizada. Por ser um projeto cujas listas de discussão e canais de *chat* são movimentados é bem comum alguns problemas serem resolvidos em questão de poucos dias;
- **Estrutura simples** - O *BOINC* possui uma estrutura simples de comunicação: um servidor que armazena e distribui os trabalhos a serem feitos e os clientes que tem o papel de processar os trabalhos;

- **Documentação completa** - A página oficial do BOINC possui muita documentação e muitos tutoriais explicando cada detalhe da instalação e configuração de um projeto. Há também páginas não oficiais, como o Wiki não oficial do *BOINC*⁵, que também servem de ajuda.

2.1.1 Funcionamento do BOINC

Cada unidade de processamento no BOINC é chamada de *workunit* e é constituída de arquivos executáveis e arquivos de entrada. Depois de processado, os arquivos de saída gerados são enviados para o servidor que normalmente armazena estas saídas 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.

⁵cujo endereço é <http://www.boinc-wiki.info>

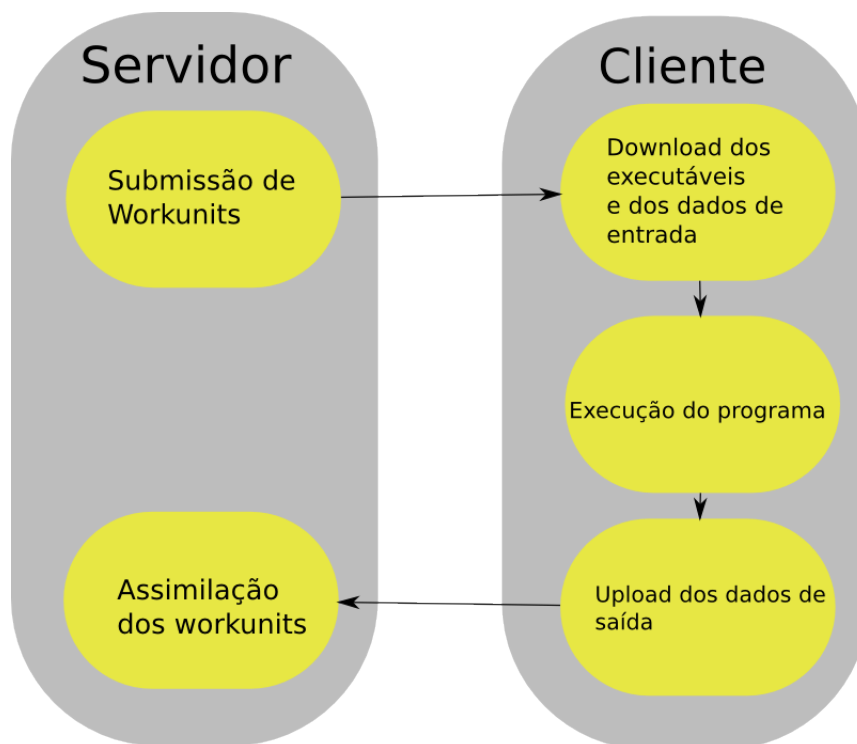


Figura 1: Funcionamento do BOINC

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 [MBK09] que utiliza um shell para a execução dos aplicativos. Os diagramas de funcionamento do *BOINC* com um programa escrito com a *API* e com o wrapper podem ser vistas nas figuras 2 e 3 respectivamente.

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* perceber se a execução do programa foi feita com sucesso, mas não é responsabilidade dele verificar se os pré-requisitos que o programa necessita estão presentes no sistema (se isso por ventura acontecer, o arquivo de saída não aparecerá no servidor, porém a execução será dada como correta no banco de dados).

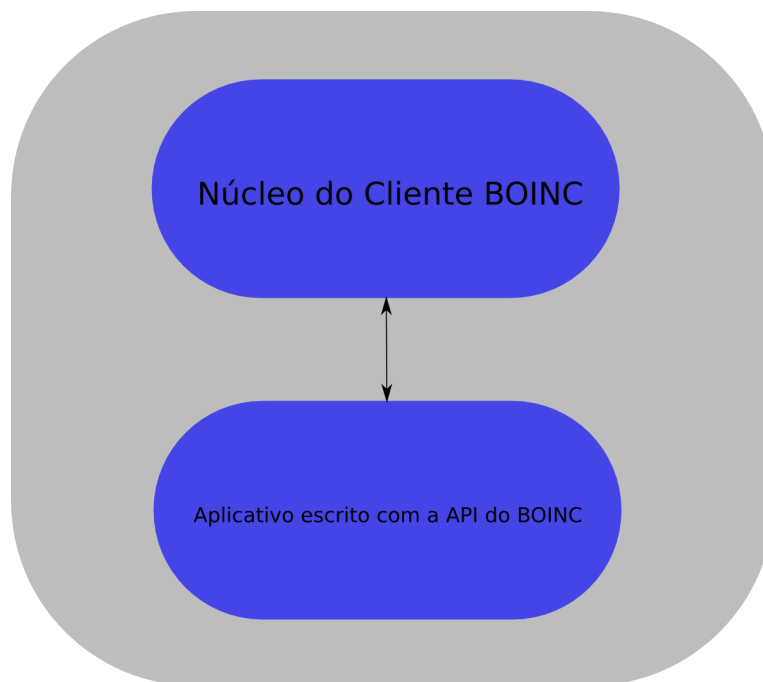


Figura 2: Funcionamento do BOINC utilizando uma aplicação que utiliza sua API

2.2 R

A linguagem de programação estatística *R* é uma implementação da linguagem *S* e foi criada por Ross Ihaka e Robert Gentleman na Universidade de Auckland, da Nova Zelândia. A linguagem e ambiente para cálculos estatísticos é considerada padrão [Van09] na área de análise de dados e além do ambiente acadêmico, em-

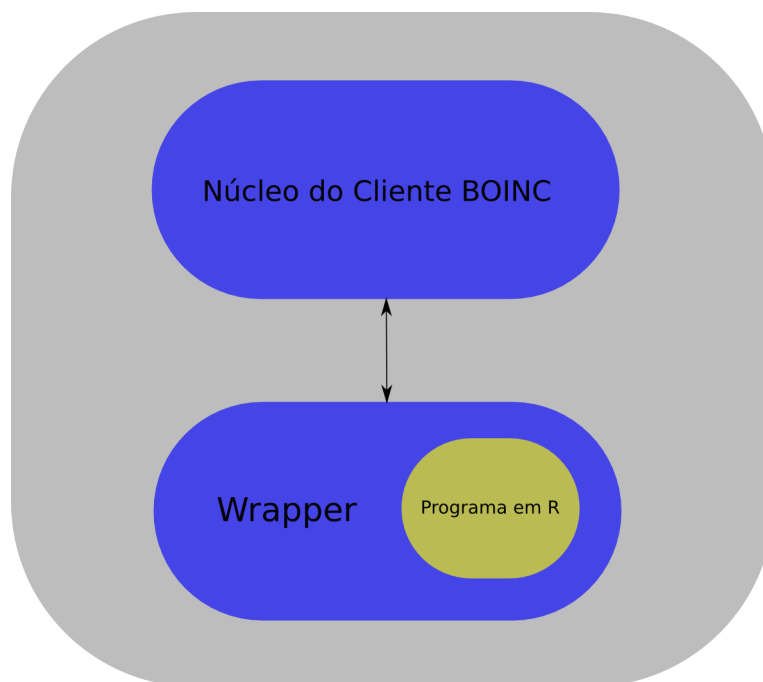


Figura 3: Funcionamento do BOINC utilizando uma aplicação que utiliza o *wrapper*

presas como *Google*, *Pfizer* e *Merck* a utilizam em seus processos de mineração de dados.

Dentre as facilidades que seu uso proporciona, podemos citar as seguintes:

- Grande quantidade de funções estatísticas de uso cotidiano na análise de dados como cálculo de média, desvio padrão, ajuste de curvas, análise de séries temporais. Há também funções mais complexas como por exemplo implementações de algoritmos de *clustering*.
- Utilizando o *R* é possível fazer operações em tabelas semelhantes a que normalmente são feitas em bancos de dados relacionais como seleção de linhas em tabelas que atendem certas condições.
- O *R* possui também rotinas de manipulação de matrizes e resolução de sistemas lineares, que são essenciais em qualquer tarefa de cálculo numérico.
- Gráficos de alta qualidade dos mais variados tipos e para os mais variados propósitos com uma qualidade alta também podem ser feitos com o *R*.
- A facilidade de se criar e de se obter pacotes, que nada mais são que conjuntos de rotinas agrupadas, é outro fator importante: é muito simples criar

pacotes e documentá-los. Para disponibilizá-los, pode se submeter um pedido de aprovação de para disponibilização no repositório oficial (<http://cran-r.c3sl.ufpr.br/>) do *R*, que possui 2076 pacotes para os mais diversos fins. O repositório possui *mirrors* espalhados por todo o mundo, inclusive no Brasil.

Além destes motivos, o artigo [GCB⁺04] dispõe outros motivos para a utilização do *R* e de seu pacote para análise de dados de de bioinformática *Bioconductor*.

- **Transparência** - Muitas metodologias de análise na área de biologia e bioinformática computacional são extremamente complexas e muitas etapas são necessárias na conversão de informação bruta (como por exemplo imagens escaneadas de *microarray*). Não se sabe a priori, como as análises podem ser sensíveis a estes fatores, e portanto trabalhos referenciados na área normalmente expõe todo o processo. O uso de mesmo ambiente e das mesmas ferramentas facilita bastante esta transparência;
- **Reprodutibilidade** - Experimentos na área de biologia molecular devem publicar listas de ingredientes e algoritmos para criar substâncias e processos. Um resultado só pode ser verificado se existir uma obediência a um protocolo. Seguindo esta linha, a mineração de dados também deve ser bem descrita e tanto o código fonte como os dados nos quais a análise é baseada são normalmente publicados junto com um trabalho nesta área. Utilizar um mesmo ambiente, que pode ser obtido gratuitamente, para qualquer plataforma, com pacotes facilmente extensíveis também facilita este processo.
- **Eficiência do desenvolvimento** - Se o pacote foi por ventura desenvolvido para alguma necessidade, ele pode ser publicado, melhorado, estudado por outros cientistas, e pode ter seu leque de funcionalidades aumentado caso seu uso siga padrões de código aberto. Para isso é necessário que esteja bem documentado. Tanto o *R* como o *Bioconductor* são softwares de código aberto e desfrutam desta qualidade.
- **Prototipagem** - Como o *R* é uma linguagem em um nível mais alto que outras linguagens como *C* ou *FORTRAN*, programar novas rotinas é bem mais simples. Mesmo não tendo uma execução tão eficiente como em outras linguagens, pode ser utilizado como protótipo, para depois ser implementado em uma linguagem mais eficiente.

2.3 Utilização do *BOINC* para o processamento de rotinas na linguagem *R*

Dentre as possibilidades de processamento em grade para a linguagem *R*, escolhamos o *BOINC* para o processamento em grade. Dentre os motivos, podemos citar:

- **Código aberto** - Ambas as tecnologias são de código aberto e possuem licenças que permitem a utilização das duas ferramentas para qualquer propósito, além da obtenção gratuita e da possibilidade de estudo do código fonte das duas tecnologias. Outro ponto forte em comum entre elas é a presença de uma comunidade ativa, com fóruns, listas de discussão e canais de IRC;
- **Multiplataforma** - Em redes de universidades e empresas é muito comum a utilização de sistemas tanto *Linux* como *Windows*. Em ambientes nos quais há muito desenvolvimento de programas, o *Linux* acaba sendo bastante utilizado, já para um usuário cotidiano o *Windows* acaba sendo a opção mais comum. Ter uma solução que utilize tanto computadores com sistemas *Linux* como computadores com sistemas *Windows* é essencial para sua utilização. Como o *R* e o *BOINC* já tem versões para estes dois sistemas, assim como o *wrapper*, fica então possível a utilização multiplataforma da grade. Lembrando que a utilização multiplataforma deve funcionar para um mesmo projeto e sem distinção de workunits, de modo a ser indiferente à quem submete a utilização de qual sistema para a computação.
- **Execução inalterada de aplicações escritas em *R*** - Utilizando o *wrapper* é possível executar, sem precisar alterar um byte do código, programas compilados. Utilizando como programa compilado o interpretador do *R*, podemos executar rotinas do *R* utilizando o *wrapper*. Isso poupa o trabalho de reescrever códigos já previamente escritos e em funcionamento;
- **Possibilidade de se utilizar pacotes do *R*** - Como o *R* possui uma notável extensibilidade com mais de 2000 pacotes, é de se esperar que um ambiente para executar rotinas nesta linguagem também seja capaz de automaticamente instalar pacotes do *R*, caso isto seja necessário. Utilizando o *BOINC* e o *wrapper* é possível utilizar estes pacotes, sendo apenas necessário verificar se eles estão instalados e colocar no programa o comando de instalação caso não estejam.

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 [RAD09]. 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 Condor⁶, 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 [GGdVS08] 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, o qual foi realizado em janeiro de 2010. No momento, 69 máquinas possuem fazem parte do grade. Este número dificilmente aumentará, já que as máquinas *Windows* que terão o *Boinc* instalado já possuem o sistema *Linux* em outra partição.

⁶Disponível em <http://www.cs.wisc.edu/condor/>

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 Implementação

4.1.1 Sistema Linux

Para o ambiente *Linux*, o processo foi relativamente simples: 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 no início do arquivo:

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

Isso faz um sistema *Linux* chamar o interpretador *Rscript* para interpretar o arquivo e assim fazer a interpretação do 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 popularmente de *shebang*. Esta implementação foi relativamente simples: o BOINC possuía um tutorial para executarmos programas compilados na grade e só o truque do shebang foi necessário.

Porém, para termos a mesma solução em ambos os sistemas, utilizamos a solução para *Windows* no sistema *Linux*.

4.1.2 Sistema Windows

Como o sistema *Windows* não permite utilizar script 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* precisa ser compilado para o *Linux* com o caminho para o interpretador correto. O programa também não faz uma verificação para perceber se o *R* está instalado, dando um resultado de conclusão positiva no banco, mas sem arquivos de saída.

A utilização do *Runner* foi necessária 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. Um diagrama exemplificando esse funcionamento pode ser visto na figura 4. Para a execução ficar multiplataforma, foi necessário fazer uma pequena alteração no *wrapper* na versão para *Linux* modificando o nome do arquivo XML do wrapper, já que em ambos os sistemas é necessária a utilização de um arquivo XML próprio.

A implementação no Windows foi bastante difícil: existem muitos pequenos detalhes que acabam atrapalhando um pouco. Um dos exemplos foi especificar o caminho do executável: para o Windows acessar um diretório específico devemos “truncar” os nomes de diretórios maiores que 8 caracteres, sendo assim apontamos o caminho, trocando o oitavo e nono caractere do diretório por ~1 e desprezando os seguintes. Alguns outros problemas foram relacionados à falta de mensagens de erro do BOINC mais amigáveis e à falta de verificação das entradas: um XML mal formatado causou um erro no processamento do *workunit*, quando isso deveria ser acusado na submissão do *workunit*. Outro problema foi um *Bug* nas configurações de compilação do *wrapper* no Microsoft Visual Studio Express que fazia referência a bibliotecas desnecessárias e inexistentes para a compilação do *wrapper* e isso só foi resolvido pelos desenvolvedores do BOINC algumas semanas depois.

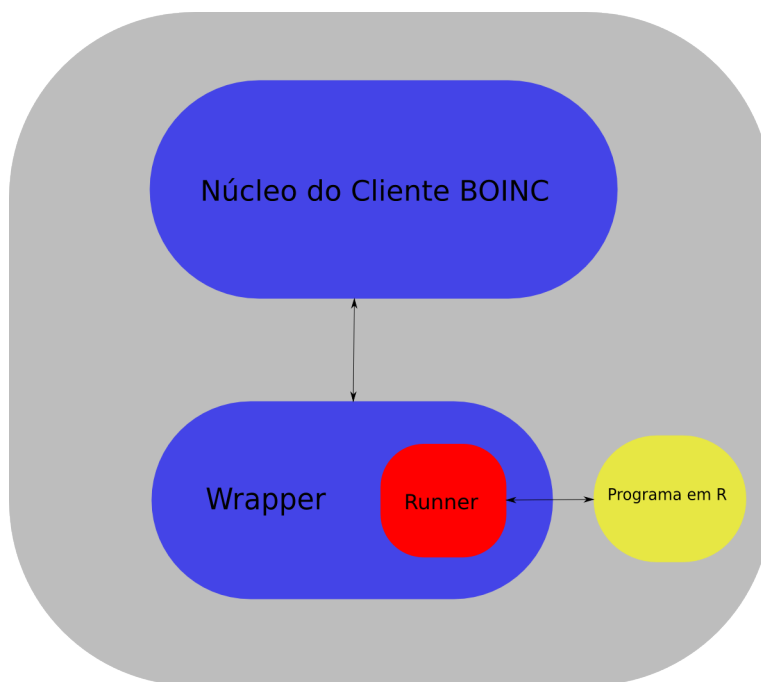


Figura 4: Diagrama do funcionamento do BOINC com o *Runner* e *Wrapper*

4.2 Discussão

4.2.1 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 e 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 após 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, assim como a frequência com que ele usará a rede.
- **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).

4.2.2 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 [RAD09] 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.

4.3 Instalação da rede

A instalação para o sistema Linux foi concluída no laboratório CEC do IME-USP. No momento possuímos 69 máquinas com o sistema Linux funcionando plenamente e 2 máquinas com algum problema na instalação dos pacotes necessários para a computação. Primeiramente, o *BOINC* foi instalado em 48 computadores (e foi neste ambiente que os testes foram executados) e algumas semanas depois, ele foi instalado em mais 21 computadores.

No momento está sendo feito a instalação no sistema *Windows* que contará com 16 máquinas que possuem um dual-boot com máquinas com o sistema Linux já presentes na grade. Esta instalação se mostrou um pouco mais complexa, devido a configurações que devem ser feitas no momento da instalação (e cuja necessidade é pouco clara na documentação).

Somente três alterações foram feitas na configuração dos clientes:

- Execução das tarefas apenas quando o computador estiver ocioso por *10min*. Por padrão, o *Boinc* é executado sempre, inclusive quando um usuário esteja usando. Como nossa intenção é executar no tempo ocioso da rede, escolhemos essa configuração.
- Conexão com o servidor sempre que possível. Como normalmente a distância de um cliente para um servidor é longa, a conexão com o servidor é feito apenas após um período determinado de tempo (por padrão *0.10dia*). Reduzimos esse valor para zero, tornando a transmissão de resultados sempre que possível devido ao baixo volume de dados transmitidos.

- *Buffer* de tarefas adicionais ajustada para zero. Por padrão, o cliente do *Boinc* assume o processamento de várias tarefas de uma vez, e devolvendo o resultado de todas estas ao mesmo tempo. Porém para distribuir melhor as tarefas nos nós e devido a comunicação com o servidor ser frequente (como especificado no item anterior) reduzimos o acúmulo de tarefas com o servidor para zero.

4.4 Testes

Para testar a grade foi criado um script na linguagem *R* que apenas fazia contas sem significado por, aproximadamente, $4min20s$ “dedicados” (isto é, executados diretamente em um dos computadores da grade). Foi utilizada redundância que enviava, para cada *workunit*, duas tarefas a serem processadas.

Foram feitos dois testes: um para certificar que as instalações nas máquinas foram bem sucedidas, e outro para calcular a média de tempo de execução sem interrupções. No primeiro teste, foram enviadas 1000 tarefas, as quais utilizaram as 48 máquinas que tinham o *BOINC* no momento do teste. Algumas execuções em determinadas máquinas foram bastante demoradas (chegando a alguns milhares de segundos) ou não terminaram, provavelmente devido a algum usuário que interrompeu o processo, ou a alguma falha na configuração.

No segundo teste, o que se buscou foi o tempo médio de retorno. Devido a utilização do CEC no final do mês de janeiro e no início de fevereiro para cursos de verão (no qual um deles utilizou o sistema Windows em uma das salas), duas máquinas fizeram o processamento das 300 tarefas. O tempo médio de processamento das tarefas foi de $9min2.167s$, com um desvio padrão de $14.486s$. Um histograma desta execução pode ser encontrado na figura 5. Um dos computadores (que chamarei de 1) teve um desempenho pouco superior ao outro tendo uma média de $8min50.130s$ (e um desvio padrão de $6.543s$) contra uma média de $9min16.404s$ (com um desvio padrão de $5.544s$) do outro computador (chamado de 2). Os histogramas do tempo de execução nos computadores podem ser encontrados na figura 6 e 7 para os computadores 1 e 2, respectivamente. O computador 1 processou 154 tarefas, contra 146 do computador 2. Todas execuções foram feitas com sucesso e todos os resultados gerados estão presentes no diretório de upload dos resultados.

O tempo total de processamento das tarefas foi de $14h31min41s$. Caso as tarefas fossem processadas em série, em um computador e de maneira direta (sem o *BOINC*) o tempo de execução seria de, aproximadamente, $21h40min$ o que nos dá um *speed-up* próximo de 1.5.

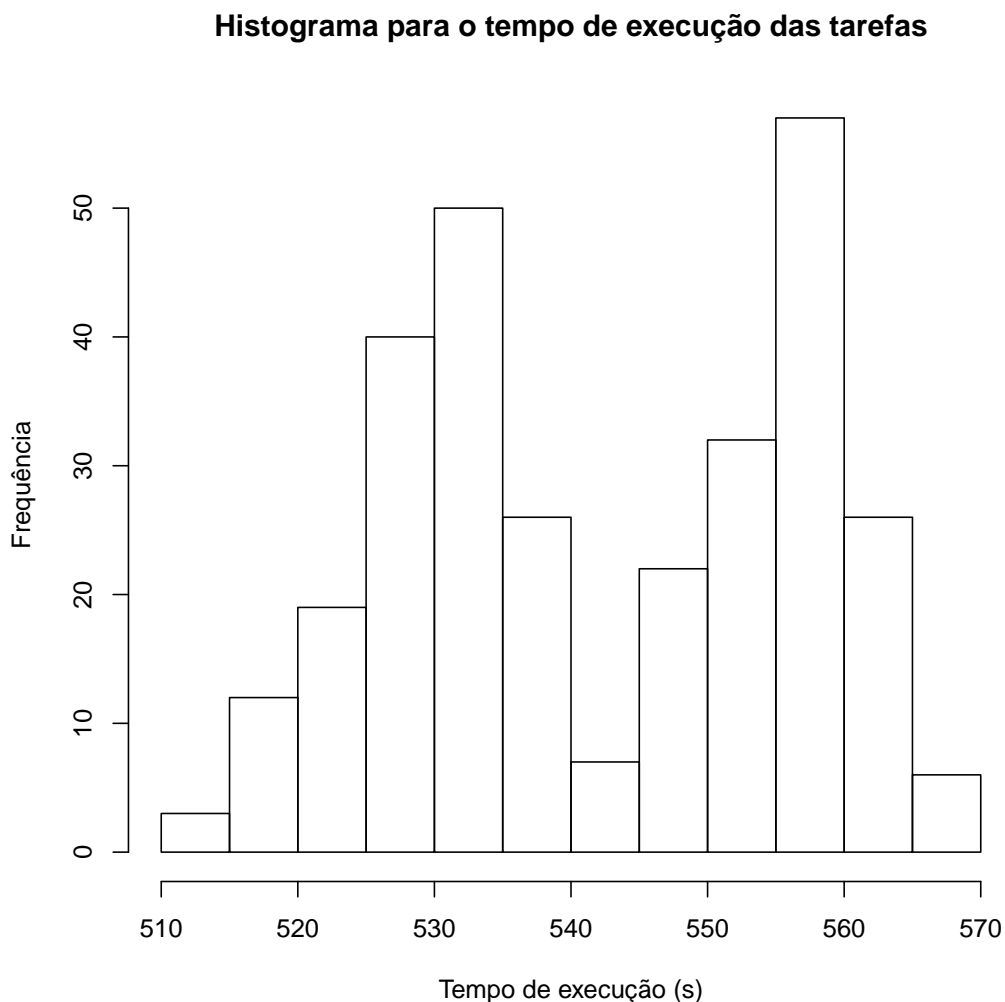


Figura 5: Histograma dos tempos de execução das tarefas

5 Conclusões

A principal conclusão deste trabalho é que é possível a utilização do *BOINC* para o processamento de rotinas de bioinformática escritas na linguagem *R*. A utilização multiplataforma também foi de grande utilidade já que podemos incluir praticamente qualquer tipo de computador existente em redes. O custo total de implementação foi somente a inutilização de uma máquina, utilizada como servidor da rede. Embora o *BOINC* seja um projeto consolidado e possua uma documentação com bastante conteúdo, as mensagens de erro, principalmente do *Wrapper*, são

Histograma dos tempos de execução das tarefas na máquina 1

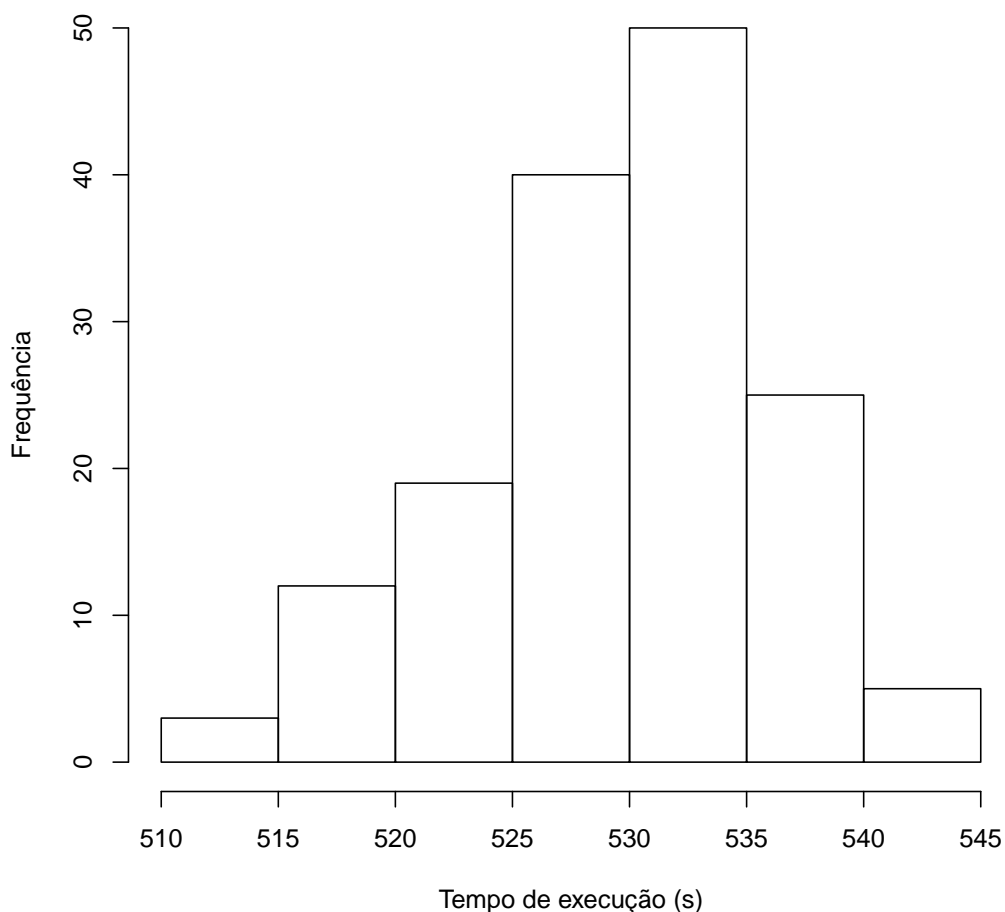


Figura 6: Histograma dos tempos de execução das tarefas no computador 1

muito pouco informativas, o que tornou o desenvolvimento deste projeto bastante trabalhoso.

A instalação do *BOINC* e do *R* na rede do CEC também foi bastante simples pois estes estão disponíveis como pacotes do Debian (que é a distribuição Linux que é executada nos laboratórios do CEC). O *deploy* da aplicação de teste também foi simples, já que o *BOINC* possui um utilitário de linha de comando e com apenas um comando (apontando o servidor e a chave de acesso) o *deploy* da aplicação é feito.

Para a continuação do projeto há várias sugestões:

Histograma dos tempos de execução das tarefas na máquina 2

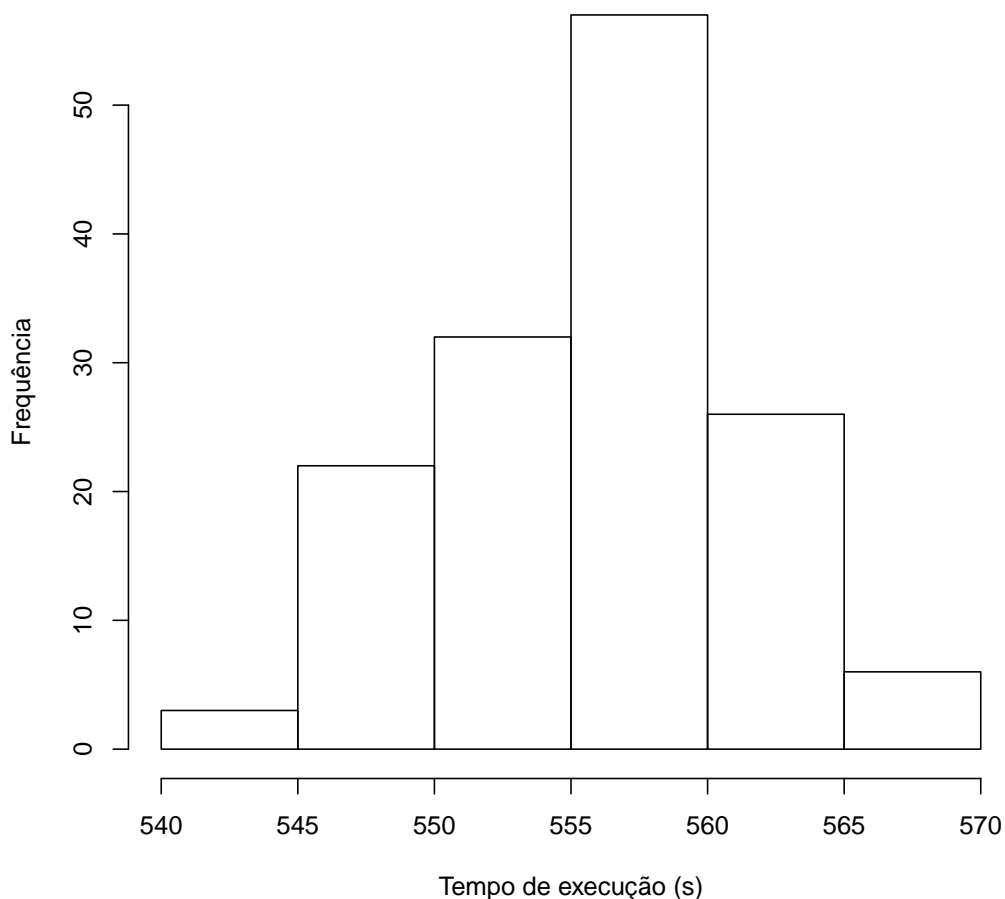


Figura 7: Histograma dos tempos de execução das tarefas no computador 2

- *Benchmark* da rede e comparação com grade descrita no artigo [RAD09] Para determinar a viabilidade, seria interessante estabelecermos a comparação com outra alternativa.
- Comparação com grades “alugadas”: hoje já existe oportunidade de se fazer esse tipo de processamento. em grades alugadas como a oferecida pela empresa *Amazon*. Como os computadores na rede consomem energia elétrica seria interessante comparar o gasto da energia elétrica com o gasto em uma grade “alugada”.

- Analisar o desempenho nas máquinas com Windows e com Linux. Seria interessante analisar o Benchmark da grade em ambos os sistemas e determinar qual das duas plataformas é mais propícia para o processamento;
- Utilização de máquinas virtuais: como feito no artigo [GGdVS08], podemos utilizar máquinas virtuais, que são iniciadas em cada nó e é feito o processamento. Sem ter que se preocupar com a instalação do *R* em todas as máquinas

Referências

- [boi] *Página de documentação do wrapper do boinc*, <http://boinc.berkeley.edu/trac/wiki/WrapperApp>.
- [GCB⁺04] Robert C. Gentleman, Vincent J. Carey, Douglas M. Bates, Ben Bolstad, Marcel Dettling, Sandrine Dudoit, Byron Ellis, Laurent Gautier, Yongchao Ge, Jeff Gentry, Kurt Hornik, Torsten Hothorn, Wolfgang Huber, Stefano Iacus, Rafael Irizarry, Friedrich Leisch, Cheng Li, Martin Maechler, Anthony J. Rossini, Gunther Sawitzki, Colin Smith, Gordon Smyth, Luke Tierney, Jean Y. Yang, and Jianhua Zhang, *Bioconductor: open software development for computational biology and bioinformatics.*, *Genome biology* **5** (2004), no. 10, R80+.
- [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.
- [MBK09] Attila Csaba Marosi, Zoltan Balaton, and Peter Kacsuk, *Genwrapper: A generic wrapper for running legacy applications on desktop grids*, *Parallel and Distributed Processing Symposium, International* **0** (2009), 1–6.
- [RAD09] Roberto Hirata Jr. Rodrigo A. Dias, *Middle-r - a user level middleware for statistical computing*, VII Workshop on Grid Computing and Applications, Recife - Brazil (2009).
- [Van09] Ashlee Vance, *Data analysts captivated by r's power*, http://www.nytimes.com/2009/01/07/technology/business-computing/07program.html?_r=2, 2009.
- [Wik09a] Wikipedia, *Boinc — Wikipedia, the free encyclopedia*, http://en.wikipedia.org/w/index.php?title=Berkeley_Open_Infrastructure_for_Network_Computing&oldid=326896451, 2009, [Online; accessed 23-November-2009].
- [Wik09b] _____, *Interpreted language — Wikipedia, the free encyclopedia*, http://en.wikipedia.org/w/index.php?title=Interpreted_language&oldid=323979791, 2009, [Online; accessed 23-November-2009].

- [Wik09c] ———, *R (programming language)* — *Wikipedia, the free encyclopedia*, [http://en.wikipedia.org/w/index.php?title=R_\(programming_language\)&oldid=326801637](http://en.wikipedia.org/w/index.php?title=R_(programming_language)&oldid=326801637), 2009, [Online; accessed 23-November-2009].
- [Wik09d] ———, *Volunteer computing* — *Wikipedia, the free encyclopedia*, http://en.wikipedia.org/w/index.php?title=Volunteer_computing&oldid=325674089, 2009, [Online; accessed 23-November-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 em alguns momentos (principalmente finais de semestre).

A falta de aprendizado de orientação a objeto também foi uma frustração: como este assunto é muito requisitado por empresas e projetos, não aprendê-lo foi bastante frustrante. Há também pouco enfoque a tecnologias, deveriam haver mais chances de aprendermos tecnologias e desenvolvermos mais projetos.

Para algumas disciplinas também que pedem projetos e trabalhos únicos que são entregues no decorrer do semestre, falta uma coisa muito importante: um *Feedback* constante do professor/monitor para saber se estamos no caminho certo. No decorrer do curso conclui algumas matérias com trabalhos que eu acreditei estar bom, mas com uma nota abaixo do esperado. Fazer um trabalho de conclusão de curso também é uma escolha muito feliz: serve para coroarmos nosso aprendizado e aplicá-los utilizando os temas que mais gostamos.

A figura de um orientador e de um tutor também foi de suma importância: é sempre interessante ter alguém que se possa receber conselhos de matérias e de optativas. Desenvolver também um projeto de iniciação científica também foi importante, para meu desenvolvimento e acredito que para qualquer aluno seria uma experiência válida.

7 Disciplinas relevantes para o trabalho

Diversas disciplinas foram relevantes para este trabalho:

- **MAC122 - Principio e desenvolvimento de algoritmos** - Este curso fornece uma base importante para as outras matérias e ajuda a melhorar o raciocínio para elaborarmos algoritmos.
- **MAC323 - Estruturas de Dados** - Aprender as diversas estruturas de dados, seu uso e a eficiência é essencial para qualquer trabalho que envolva ciência da computação.

- **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 contribuíram 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 e muitos dos conceitos aprendidos neste curso ajudaram o entendimento melhor de algoritmos e soluções.
- **MAC431 - Programação paralela e distribuída** Aprender a utilidade de programação de alta performance foi de bastante utilidade neste trabalho.
- **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.
- **IBI5031 - Reconhecimento de padrões** - Neste curso tive a oportunidade de aprender algoritmos e fazer análises de dados, que são normalmente feitas em pesquisas de bioinformática. Também programei bastante na linguagem *R* neste curso e percebi sua importância

8 Conceitos Aplicados

Dentre os conceitos aprendidos no curso utilizados no trabalho pode-se citar a computação distribuída, aprendida no curso de computação paralela e distribuída. Saber que alguns problemas só podem ser resolvidos de uma exaustiva, também foi importante e esse foi um conceito aprendido no curso de Análise de Algoritmos.

9 Continuação do trabalho

A continuação do trabalho será escrever um artigo falando sobre a grade para ser publicado na página do *BOINC* e em alguma publicação. Outro trabalho interessante decorrente deste seria melhorar o *Wrapper* e o *BOINC* em geral para fazê-los terem mensagens de erros mais claras.

10 Agradecimentos

Este trabalho foi feito com a ajuda de diversas pessoas. Indiretamente, meus pais, familiares, amigos e namorada foram de extrema importância no apoio, amizade e carinho.

Diretamente, agradeço à CNPq pela bolsa que me permitiu a dedicação exclusiva a este projeto. Agradeço também ao meu orientador, Prof. Dr. Roberto Hirata Jr. que me apoiou e me ajudou bastante na elaboração deste trabalho. Para a obtenção do servidor, agradeço ao Prof. Dr. Roberto Marcondes César Junior, e aos administradores da Rede Vision, David Pires e Rodrigo Bernardo Pimentel, que me ajudaram quando na substituição do servidor com um problema de *hardware*. Agradeço também ao aluno de pós-graduação Rodrigo Assirati Dias, que me ajudou e me deu diversas dicas com relação à manutenção e a instalação da grade. Na parte do *BOINC* agradeço à Nicolás Alvarez e Yoyo, que me ajudaram a resolver os bugs e a implementar a solução descrita pelo trabalho. Agradeço também o Prof. Dr. Alfredo Goldman pela disponibilização da rede *CEC* e aos administradores da rede, Jânio Matsuura e Gislaine Olivi pela instalação da grade.